J. Parallel Distrib. Comput. 73 (2013) 371–382

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Building a reliable and high-performance content-based publish/subscribe system

Yaxiong Zhao [a,*,1], Jie Wu [b]

[a] Amazon.com Inc, Seattle, USA

[b] Department of Computer and Information Sciences, Temple University Philadelphia, USA

ARTICLE INFO

Article history:
Received 22 June 2012
Received in revised form
17 October 2012
Accepted 20 December 2012
Available online 29 December 2012

Keywords:
Content-based publish/subscribe
Fault tolerance
Reliable distributed system
High-performance system design

ABSTRACT

Provisioning reliability in a high-performance content-based publish/subscribe system is a challenging problem. The inherent complexity of content-based routing makes message loss detection and recovery, and network state recovery extremely complicated. Existing proposals either try to reduce the complexity of handling failures in a traditional network architecture, which only partially address the problem, or rely on robust network architectures that can gracefully tolerate failures, but perform less efficiently than the traditional architectures. In this paper, we present a hybrid network architecture for reliable and high-performance content-based publish/subscribe. Two overlay networks, a high-performance one with moderate fault tolerance and a highly-robust one with sufficient performance, work together to guarantee the performance of normal operations and reliability in the presence of failures. Our design exploits the fact that, in a high-performance content-based publish/subscribe system, subscriptions are broadcast to all brokers, to facilitate efficient backup routing when failures occur, which incurs a minimal overhead. Per-hop reliability is used to gracefully detect and recover lost messages that are caused by transit errors. Two backup routing methods based on DHT routing are proposed. Extensive simulation experiments are conducted. The results demonstrate the superior performance of our system compared to other state-of-the-art proposals.

1. Introduction

Content-based publish/subscribe (CBPS) systems as a communication substrate have many advantages compared to traditional connection-oriented architectures, including better flexibility, anonymity, straightforward support for content caching, efficient support for complex communication patterns, etc. The most popular designs organize brokers into an overlay network, and let *subscribers* express their interests by sending *subscriptions* to *brokers*. Routing tables are configured accordingly by each broker so that published *messages* can be routed to the interested subscribers. Subscriptions in CBPS systems are generated from a multi-dimensional description model that grants the users a high-degree of expressiveness in expressing complex application requirements.

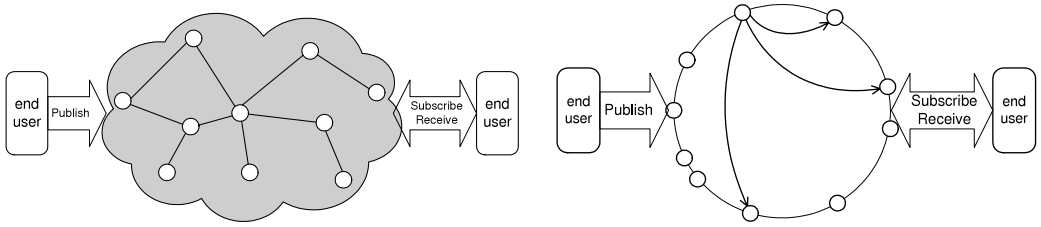
There are generally two architectures for a CBPS system as shown in Fig. 1. The first architecture, called *Broadcast-based* CBPS (B-CBPS) in this paper, employs a mesh-based overlay network of brokers to connect publishers with subscribers. In this architecture, subscriptions are broadcast and routing paths are obtained by reversing the broadcast tree. The second architecture aims at supporting large-scale and dynamic CBPS systems without central coordinations. P2P technology is key in this architecture, so it is called *P2P-based* CBPS (P-CBPS) in this paper. B-CBPS can be highly-efficient and optimized, and is used in various performance-critical scenarios [21]. To avoid the complexity of reconfiguring brokers' routing tables when failures occur, backup and/or replication techniques can be employed. However, this approach causes considerable overhead and costs in maintaining a replica of a broker's states, which results in increased hardware, software, and manpower costs. On the other hand, P-CBPS offers significantly higher reliability compared to B-CBPS because of its robust network architecture. However, the performance of P-CBPS is inferior to B-CBPS as is demonstrated in many previous studies [14,25,36], which hinders its uses in performance-critical scenarios.

Because messages are forwarded based on the filtering of brokers, communication in a CBPS system is fundamentally dynamic and distributed, and there are no predefined endpoints for forwarding messages. Additionally, users can revoke their interests by *unsubscribing*, which further complicates the communication patterns and increases the dynamism. Due to these issues, conventional techniques for network reliability do not work in a CBPS

* Corresponding author.
E-mail addresses: justicezyx@gmail.com, zhaoyaxi@amazon.com (Y. Zhao), jiewu@temple.edu (J. Wu).

[1] This work is done while Yaxiong Zhao is a PhD student in Temple University.

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.12.014



(a) B-CBPS architecture. (b) P-CBPS architecture.

Fig. 1. High level illustration of the two architectures of CBPS systems.

system. A lot of research has been done to address this issue [2,3,6–8,10,15–17,22]. However, existing proposals either rely on replication, which requires additional resources, or trade performance degradation for better reliability. Our goal in this paper, therefore, is to explore the design space of a high-performance CBPS system with a high-degree of reliability.

We observe that the fragile network structure of B-CBPS, which is key to achieving high performance message forwarding, hinders efficient fault-tolerance. The network structure itself lends little resources to cope with errors and failures. On the other hand, P2P style systems provide far better reliability and fault-tolerance because of their inherent resilient network structure, which inevitably results in an inferior performance compared to B-CBPS. In this paper, we take a mixed design approach. The basic idea, however, is to combine the two structures. We show that overlapping a P2P-based fault-tolerant layer on top of the B-CBPS infrastructure requires minimal overhead, both in storage and bandwidth consumption, while providing significantly higher reliability than the B-CBPS alone. A key enabling fact of this result is that, in B-CBPS, subscriptions are broadcast to all brokers. Because all brokers are aware of all others' subscriptions, they do not need to retransmit subscriptions to establish backup routing when a failure occurs, which provides a smooth transition from the high-performance B-CBPS infrastructure to the P2P routing layer. This smoothness can be crucial to many real-world applications, for example, financial trading systems [31] and distributed messaging middleware [21], which require minimal disruptions. Additionally, it simplifies the maintenance of the P2P routing layer and reduces overheads.

The first design question of the system involves how the system can detect message loss. Transit errors or failures of overlay links and brokers can result in dropped messages. As proven in previous work [3], loss detection in CBPS systems is complicated. This is because each message is filtered by brokers and the subscribers that should receive the message cannot be determined without actually delivering it. In [8], a method that requires each message to piggyback matched subscriptions is proposed. It is effective but may cause a large overhead since the number of matched subscriptions of popular messages can be high, and the verification process requires examining the piggybacked subscriptions, which can significantly delay the delivery of the message. In [3], a lightweight non-intrusive technique for best-effort CBPS systems is presented. This technique is simple and effective, but does not fully address the lost message detection problem because it only supports probabilistic detection. In this paper, we propose the use of a per-hop loss detection and recovery scheme, which avoids unnecessary overhead and guarantees the detection of message loss.

Another problem lies in efficiently maintaining the P2P routing layer. Existing techniques for P2P-based CBPS are designed for large-scale systems that are distributed across a wide geographic range, and require complex mapping between the content space and the P2P identifier space. Their performance cannot meet the requirements of many performance-critical applications. We design a light-weight P2P routing protocol that does not need complex processing. In order to achieve a good delivery performance, the protocol utilizes a unique feature of the B-CBPS infrastructure, i.e., consistent subscriptions on all brokers. In the presence of persistent or long-term failures, which would require the reconfiguration of the B-CBPS infrastructure to provide the optimal performance, our system falls back to the conventional topology reconfiguration procedures [22]. This design makes sure that the system performs consistently in all situations and maintains the optimal performance in the long run.

Our main contribution in this paper is a hybrid CBPS system that seamlessly combines two network architectures, namely B-CBPS and P-CBPS, and demonstrates the effectiveness of the hybrid design. Other important contributions include:

- We present a light-weight loss detection scheme that fully detects all lost messages with minimal overhead.
- We design a P2P routing scheme based on content space mapping, which provides sufficient performance and a smooth transition from the B-CBPS infrastructure to the P-CBPS layer.
- We propose a failure recovery protocol for topological reconfiguration in the presence of long-term or persistent failures.
- We perform extensive simulation studies to verify the performance of our designs. The results demonstrate the effectiveness and efficiency of our system.

The rest of the paper is organized as follows. Section 2 presents the assumptions that are used in the discussion and an overview of the system. Section 3 presents the detailed designs, including loss detection, P2P routing, and routing layer switching. Section 4 sketches the implementation of our system. Section 5 presents the results of the performance evaluation from both simulations and testbed experiments. Section 6 discusses related work, and Section 7 concludes this paper.

2. Problem settings and overview of design

This section presents the important settings of our research context and an overview of the design.

2.1. Problem settings

The system is comprised of *brokers* and *end users*, which are collectively called *nodes*. Usually, brokers are processes running on powerful hardware, and end users are running on PCs. Each broker serves as a portal for a large number of end users, usually on the order of thousands or more. We assume that each end user only connects to one broker. In practice, multi-homing can be applied, but the overall design still works. Our problem settings are fundamentally different from a pure P2P-based CBPS system, where all end users are considered equal in their processing capability and functionality. Our settings are more like those P2P systems that have *super peers* [1].

Similar to the design in [2], each broker implements a module called *subscription endpoint* or *subend*, which is called *subscription processor* in this paper. A subscription processor processes incoming subscriptions from end users and other brokers to

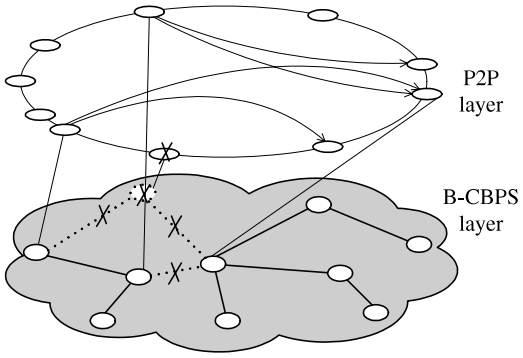


Fig. 2. High level illustration of the two-layer hybrid routing structure.

configure the routing table, and stores the routing table by using a reliable storage service. Therefore, it is assumed that a broker could regain its routing table after recovering from a failure. This assumption is different from some previous research [17,16]. Our rationale is that reliable storage services are generally available and cheap in practice, for example, with Cloud storage services. Therefore, in this paper, we will not consider the problem of recovering the routing tables of failed brokers.

Each broker also implements a *publish endpoint* or *pubend*. Pubend maintains a persistent record of all delivered messages to the end users that directly connect with it. Same as subend, pubend also provides reliable storage service. End users can recover history messages from a broker if needed. Therefore, the problem of providing durable publications [3] is not considered in this paper.

From an abstract point of view, end users could be *publishers* and *subscribers*. We assume that each end user can simultaneously publish messages and subscribe to certain content. In simulations, this setting is simulated with a uniform publish rate for all end users. Additionally, we acknowledge that in practice, it is most likely that only a few information sources will publish messages, whereas all other end users publish a much smaller amount of messages. This setting is also simulated in the performance evaluation.

The subscription processing follows the conventional approach [5,6], where subscriptions are broadcast to all brokers, and a dissemination tree is formed on the reverse path of the broadcast tree. Because the subscriptions of end users are summarized by brokers, the source of the subscriptions that are processed in the broker network only reflect the ID of the originating brokers. Messages are routed between brokers and then dispatched by pubend based on the original information recorded by the subend. We will use the *source* of a subscription to refer to the broker from which the subscription originated, instead of the end user that actually sent the subscription. Therefore, most discussions in the rest of this paper will not consider end users.

Our designs address two types of failures. The first, random transient failures that cause message loss. This type of failures are persistent for a short time frame, usually in a sub-second scale, which are best recovered by using message retransmission instead of complex topology reconfiguration protocols. Such failures are expected to happen frequently because Internet services are moving to Cloud where services are collectively provisioned by a large number of distributed and virtualized servers running on physical machines. Virtualized servers could be brought back quickly which makes the transient failures possible. The second, persistent failures. These errors are usually caused by physical servers down and virtual servers could not be initialized on other physical servers. Bringing physical servers back needs a few minutes. Using message retransmission is unable to gracefully mask the latency experienced by users. In this case, a topology reconfiguration protocol is necessary. In rare situations, physical servers could be permanently down, which requires a long-term topology reconfiguration plan. These situations are all required to be considered in the designs.

2.2. Overview of design

Our basic design is illustrated in Fig. 2. Two routing layers are used. At the bottom, a high-performance distributed tree-based infrastructure is used, which is called the *B-CBPS infrastructure*. This layer can be constructed manually, which may intentionally optimize the end-to-end delay and throughput. Manual configuration also makes it straightforward to implement specific policies, which are specified by outside business logic. Tree-based content routing is employed in B-CBPS. Each broker broadcasts the subscriptions that it receives from end users and records itself in the subscription as the source. When subscriptions are received by other brokers, duplicate subscriptions from the same source will be discarded, and new ones will be stored in the routing table. The routing table records the broker from which the subscriptions are received. This process makes sure that the paths with the minimum latency to the same broker will be used. In message publishing, end users push messages to a broker, and the broker checks its routing table to determine where to forward the message.

On top of the B-CBPS infrastructure, a P2P routing layer is created. In this paper, we use Chord [29], which is proven to offer the best overall routing performance and reliability [13]. A broker's domain name and port number are used as a hash key to compute its ID in the Chord ring. Overlay links are created by using Chord protocol. The P2P routing layer is not used for forwarding messages if no error or failure occurs. Maintenance is still needed for maintaining valid states of the P2P routing. Its details are in Section 3.3.3. When failures occur, traffic will be automatically forwarded into the P2P layer. Specific designs are presented in Section 3.3. Because of DHT's inherent fault tolerance, messages could be delivered in the presence of serious network failures. Our design also allows the B-CBPS infrastructure to reconfigure when failures are persistent or network partitions occur, which optimizes the long-term performance of the system. The reliability of this P2P layer is key to supporting the overall reliability of the entire system. Previous work on DHT reliability has already verified its superior fault-tolerance [13,29].

3. Design

This section elaborates on three major components of our system: message loss detection, p2p routing, and routing layer switch. Message loss detection and recovery help the system to recover from transit or short-term failures. P2P routing addresses efficient routing in the presence of link or node failures that cause network partitioning. It helps brokers that are affected by failures form a temporary routing structure to perform routing efficiently. The routing layer switch makes sure that P2P routing and the B-CBPS infrastructure switch smoothly when failures occur and are resolved.

3.1. Message loss detection and recovery

This section presents the design of the per-hop reliability based on sequence numbers. We then discuss the lost message recovery mechanism, which is based on sliding window transmission.

3.1.1. Per-hop reliability based on sequence numbers

In the presence of transit errors, message loss can occur. Attaching sequence umbers to every message at the publishers is insufficient for detecting message loss in a CBPS system. The inability is resulted from the fact that the sequence numbers assigned at

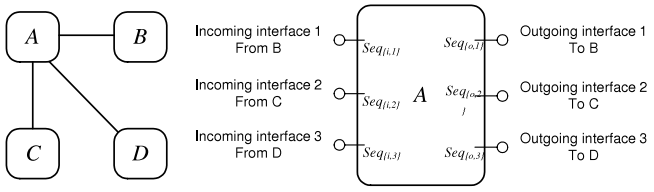


Fig. 3. High level illustration of the configuration of the sequence numbers associated with the logical interfaces of a broker.

a publisher do not reflect the sequence at which the subscribers would receive. Because messages are filtered by brokers distributedly, there will be gaps between the sequence numbers received at the subscriber. Therefore, the subscriber cannot distinguish such gaps from the gaps caused by message loss, which makes the sequence number useless in detecting message loss.

Even though sequence numbers do not work for end-to-end message loss detection, it still works in a hop-by-hop scenario. The subscriptions sent from a broker to another neighboring broker are consistent. If a message is sent from a broker to another neighboring broker, the message's sequence is consistent on both brokers. In other words, between two neighboring brokers, for any message being sent from one broker to the other, the two brokers have a consistent view of the sending and the reception of the message. If both brokers negotiate a common inception sequence number, the broker that is receiving messages could detect a loss by comparing the message's sequence number to the next expected one. A broker maintains the sequence number for each of its egress interfaces to neighboring brokers. The sequence number of an interface indicates the next expected sequence number that should be assigned to an outgoing message or should be received from an incoming message.

Take the configuration in Fig. 3 as an example: broker *A* maintains sequence numbers for each incoming and outgoing interface. When messages are acknowledged, the sequence number associated with the interface is incremented by 1. This mechanism is supposed to detect message loss caused by transient failures. Timeout will trigger routing switch or topology reconfiguration to bypass persistent failures.

3.1.2. Lost message recovery

Although the mechanism present in the previous section is a straightforward application of the per-hop reliability scheme, there is a small disparity caused by the unique properties of CBPS systems. As discussed before, a message is forwarded according to the routing table of the broker, so the message will potentially be forwarded to multiple neighboring brokers through different interfaces. This requires the broker to maintain multiple sequence numbers for each sent message. Only when all of the acknowledgments from the next-hop brokers are received can the sending broker remove the message from its buffer. Therefore, a broker needs to maintain a variable number of sequence numbers for each message and compare them with an equal number of acknowledgments.

3.2. Routing layer switch protocol

With per-hop loss detection and recovery, the end-to-end reliability is guaranteed under any condition where no link or node failure occurs. We will simply use failure to refer to the link and node failures that cause disconnections in the network. Our basic idea for handling topology reconfigurations caused by failures is to let the brokers that are affected by any failure switch to a P2P routing network. This idea is illustrated in Fig. 2. In Fig. 2, link and node failures cause brokers to be disconnected. The affected brokers then forward messages in the P2P routing layer. Due to the better fault-tolerance of the P2P network, routing paths still exist. The details of the P2P routing protocol used in our system will be given in Section 3.3. In this section, we describe how to switch between the two routing layers.

It is arguable that the system should immediately resolve failures and reconfigure system states to preserve normal operations. This used to be vital because recovering a failed server can take excessive time. However, Cloud computing technology, which is enjoying a fast adoption rate worldwide, changes this situation. That is, a broken server could be brought back quickly, either by replicating virtual machines or by migrating workloads to a new physical server. This operation could be completed in a relatively short time, for example, less than a minute. Such a short time is still long enough to noticeably affect system performance. On the other hand, because the time is short, it makes the recovery of the broker network very expensive. As discussed in [22], in order for the broker network to regain its valid state after a topology change, the subscriptions that are associated with the affected brokers need to be repropagated throughout the network, which could be the entire network in the worst case. This operation involves multiple brokers and can potentially take a much longer time in the case of cascading failures. Even worse, when the broken brokers come back, the same operation needs to be performed again, which is quite a waste of resources. Additionally, the brokers cannot forward messages to subscribers that are affected by the failure until the reconfiguration is completed, which incurs an excessive delay.

This motivates us to employ a two-layer routing switch approach. Firstly, our routing switch protocol does not require the system to immediately respond to the failures in the B-CBPS routing layer. Instead, the traffic is redirected into the P2P routing layer. With an almost always available P2P routing layer, the above complicated operations could be avoided if failures are transit. In our preliminary design, we set a timeout value to the 90% percentile value of the down time of a broker. After the timer expires, a routing table reconfiguration will be executed to reconfigure network states. After the B-CBPS infrastructure is recovered and settled down, the affected brokers will switch back to the B-CBPS routing layer. This scheme avoids the unnecessary network reconfigurations and simplifies the overall system design.

If no failure occurs, a broker would never forward messages into the P2P routing layer. A broker starts forwarding messages to the P2P layer only when it detects failures in its neighboring brokers (link or broker failures). We use *switching brokers* to refer to the brokers that are required to send or receive messages to or from the P2P routing layer. When switching brokers, the subscriptions that are affected by the failed brokers need to be identified. There are two types: the sending and receiving subscriptions. The sending subscriptions are those that are received from the failed brokers; the receiving are those that receive messages from the failed brokers. The sending brokers will forward messages that should have been directed to the failed brokers to the P2P layer; on the other hand, the receiving brokers register their subscriptions that should have received messages from the failed brokers. These ideas are illustrated in Fig. 4. Broker *A*'s connection to *D* is broken. The interface to *D* will be marked as *repair*. The messages sent through a repair interface should be directed to the P2P layer.

3.3. P2P routing in the presence of failures

During the time that the P2P routing layer is being used, its performance should be good enough to support normal system operations. We strive to strike a desirable trade-off between reliability and performance and also provide means to adjust the trade-off. Existing P2P-based CBPS designs have already explored a wide range of the design space. We include a CBPS routing

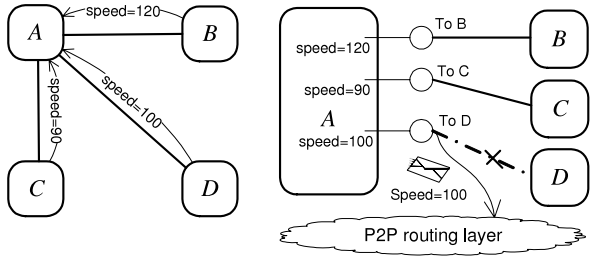


Fig. 4. The routing table of broker *A* is configured to forward the message to the matched brokers of each subscription. Traffic will be redirected to the P2P routing layer if failure occurs.

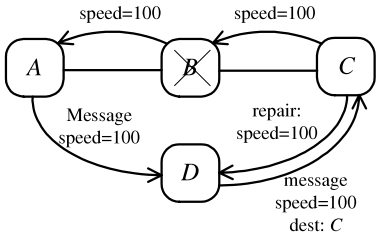


Fig. 5. Illustration of the rendezvous-based repair. Broker *B* failed. Broker *C* and *D* are the affected brokers. Broker *C* requests repair from the rendezvous broker *D*. *D* then forwards messages to *C* by using DHT routing. Arrows indicate the direction of message forwarding.

protocol based on Chord [29] as a comparison in the performance evaluation section. However, all of these proposals are designed for a large-scale system scattered over a wide geographic range, which has a high degree of dynamics, user churn especially. As a result, most of them are designed to address this problem instead of providing highly-efficient message forwarding. This motivates us to investigate new techniques that better serve our purpose in the design.

The content mapping mechanism is the method of mapping from the content space of the CBPS system to the identifier space of the P2P network. It is used in most existing P2P-based CBPS systems. Certainly, it is straightforward to apply this method in our design. However, this approach may result in performance degradation. We would like to discuss it in detail here because each broker in our system collects the subscriptions from a large number of subscribers. In our plan, each broker should be able to serve 100,000 users on average. The users' interests can be diverse, so the subscriptions collected at each broker will generally cover a very large fraction of the entire content space. When failures occur, affected brokers need to inform all of the rendezvous brokers that are responsible for handling the part of the content space that corresponds to the subscriptions of the affected brokers. Since the subscriptions generally occupy a large fraction of the entire content space, it is likely that a large fraction of the brokers will be the rendezvous brokers. Therefore, the affected brokers need to contact many brokers. The messages, on the other hand, only need to be forwarded to the corresponding broker, which is guaranteed to be one of the rendezvous brokers. In conclusion, the problem is that the time it takes for the affected brokers to contact all of the rendezvous brokers may be long.

3.3.1. A rendezvous-based approach

One simple and effective solution is to take a pro-active approach in forwarding the messages. When a message is directed to the P2P routing layer, the rendezvous brokers will broadcast the message to all matched brokers, regardless of the possible waste. The rendezvous brokers are mapped from the subscriptions that require repair by the DHT protocol. Fig. 5 shows a very simple network configuration, where broker *D* is mapped by the subscription of *speed* = 100 and requires repair by broker *C*.

After a short time, the notification of the affected brokers will finally arrive at the rendezvous brokers. Then, the rendezvous brokers seize broadcasting and use P2P routing to transmit the message to the affected brokers. This method comes at the cost of wasted bandwidth and duplicate messages because not all recipient nodes need the messages. A timer is used to reduce the bandwidth waste by setting up a timeout to stop broadcasting. Additionally, duplicate messages can be easily removed by comparing the sequence numbers assigned by the publishers. Also, note that broadcast here means to forward the message to all of the matched brokers; it actually does not broadcast to all brokers in the network. The rationale for using this concept is the fact that not all matched brokers are affected by the failure, but they still receive messages from the P2P layer.

When a broker detects a failure, if it decides to forward the messages to the P2P routing layer, it sets a *repair* flag in the message to indicate that this message should be transmitted in the P2P routing layer by the rendezvous broker. When the rendezvous broker receives the message, it first checks its routing table to find the matched brokers of the message. Part of the matched brokers can be reached by the B-CBPS infrastructure, and this information can be found readily by examining the original broker of each subscription in the routing table. This set of brokers are called the *downstream brokers*. The rendezvous broker then removes the downstream brokers from the matched brokers. The resultant brokers' IDs are called the *destination set* and are appended to the message. This message will be forwarded to the nearest broker in the destination set. The first broker that receives the message in the destination set will strip its ID from the destination set and forward the message to the next nearest broker. This process continues until all brokers in the destination set are reached. It is possible that some brokers in the destination set are down. In that case, the intermediate brokers in the P2P routing layer can detect this and automatically remove those brokers from the destination set.

The timeout of stopping a broadcast is set as follows. It should be at least as large as the time for any affected brokers to notify the rendezvous broker. This time is equivalent to the maximum path latency in the P2P network. We derive the maximum path latency by multiplying the average per-hop link latency by the maximum path length in hop counts. The former could be estimated locally. The maximum path length in a DHT-based P2P network can be calculated analytically, as demonstrated in previous work [13]. Or, we could do a sampling by using a random walk. When the time expires, the broadcast stops and the rendezvous broker will start forwarding the repair messages to the affected brokers. The benefit of the rendezvous-based method is that the number of brokers that require repair messages is small.

3.3.2. A multicast-based approach

Since subscriptions are broadcast, each broker is aware of the sources of the received subscriptions. Fig. 6 illustrates this fact. Usually, content matching returns the interface associated with the subscription. When a failure occurs, the interface connected to the failed link and brokers will be marked as *repair*. For these interfaces, the sources of the subscriptions will be collected and set as the destination set of the repair message. As shown in Fig. 6, when a new message directed to interface *B* is received, broker *A* appends the source of the matched subscription, *D*, to the destination set of the message. The message is then directed to the P2P layer. Just like the previous rendezvous-based approach, brokers forward the message starting with the nearest the destination.

Note that this approach is similar to MEDYM [4]. The difference is that our design uses P2P routing to provide maximum reliability,

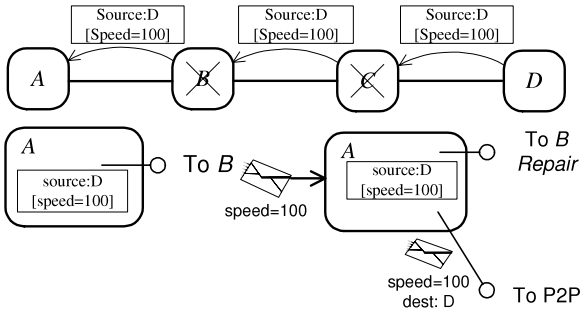


Fig. 6. Illustration of the multicast-based repair. Broker *A* knows the source of the subscription. When *A* detects that *B* failed, the interface to *B* is marked *repair*. Messages directed to the interface will be redirected to the P2P layer.

whereas MEDYM uses overlay unicast and multicast to optimize delivery performance, which does not address the reliability issue. We would also like to point out that our design could be straightforwardly integrated with most broadcast-based CBPS systems. More details about the integration with existing systems are given in Section 4.

3.3.3. Maintenance of P2P routing layer

The overhead of our design lies mainly in the maintenance of the P2P routing layer. Section 2.2 indicates that building P2P routing on top of a B-CBPS infrastructure requires minimum overhead because subscriptions do not need to be transmitted. The B-CBPS protocol makes sure that all brokers store subscriptions that originated from all other brokers. Each broker obtains an ID by hashing its domain name and port number. The basic DHT is built from the IDs of the brokers. Note that this process does not concern subscriptions. A content-to-ID mapping protocol is required in the rendezvous-based repair protocol. We could borrow the designs from previous P2P-based CBPS systems [14,24,30,36]. Multicast-based repair, on the contrary, does not need such a mapping protocol. A failure detector is needed to keep the statuses of the neighboring brokers up-to-date, which is also required by the B-CBPS.

3.4. Message buffer and loss recovery

Per-hop loss detection and recovery has a limitation when broker failures occur. If a neighboring broker fails, previously-acknowledged messages may be lost before they could be successfully delivered to the next-hop broker. It is necessary to buffer acknowledged messages for a certain time. The time is determined by the maximal tolerable failure in the system. Analogous to the sliding window transmission, the buffer size is determined by the one-way delay for a message to traverse n brokers. Fig. 7 illustrates this idea. Here, n is the maximal brokers that can fail simultaneously given that any lost message can be recovered. Suppose that the delay of reaching the $(n+1)$th broker is σ, and the outgoing bandwidth of a broker is B, the size of the message buffer in bytes should be $\sigma \times B$. Because messages are of variable size, the actual count of messages stored in the message buffer could vary. To ensure a high probability of loss recovery, the total size of the buffered messages should be larger than $\sigma \times B$. Note that the above analysis applies for every outgoing interface of a broker, because the message buffer needs to be maintained for all of the outgoing interfaces of a broker.

The messages held by the message buffer will be forwarded during the failures. As is discussed above, the message buffer is designed for worst-case scenarios where σ brokers fail simultaneously. Usually, not all messages in the buffer are lost. We employ

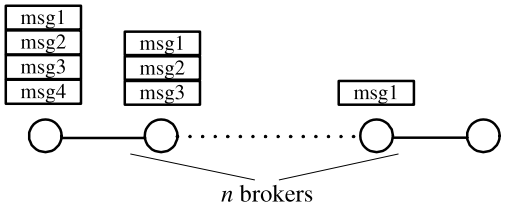


Fig. 7. Illustration of how to determine the size of the message buffer.

a mechanism called *message replay* to avoid unnecessary transmissions. Because of the per-hop loss detection and recovery, if a message is successfully delivered to subscribers, all messages that are older than that one are delivered too. Based on this observation, in a message replay, the messages in the message buffer are retransmitted from the newest to oldest. When a subscriber receives a duplicate message, it reports to the rendezvous broker (in rendezvous-based routing) or the source broker (in multicast-based routing). When the notification is received, the rendezvous broker or the source broker will seize the message replay process to the subscriber. After all messages in the message buffer are retransmitted, or all affected subscribers acknowledge that the replay process should stop, the source broker will send new messages. The message replay does not require explicit signaling to indicate its beginning because the rendezvous brokers and the subscribers can detect it automatically by checking decreasing sequence numbers in the repair messages that are being received from the P2P routing layer. Similarly, the transmissions of new messages are detected by observing the increasing sequence numbers that are larger than the previously recorded numbers in the message replay process.

Message replay avoids unnecessary retransmissions and saves bandwidth. However, it complicates operations. In certain scenarios where this complexity negatively affects the overall performance, or the users prefer a simpler scheme, message replay can be disabled. Instead, all messages in the buffer could be retransmitted.

When a failed broker is recovered, the lost messages that have not been delivered to it can be recovered straightforwardly. As is discussed before, message buffer guarantees that all messages has not been delivered to the next hop broker will be buffered. Per-hop sequence numbers let the recovered broker easily identify lost messages. It can directly request those messages from its upstream brokers.

4. Prototype implementation

We implement a prototype on PlanetLab [28]. This section presents important implementation details and the evaluation results.

4.1. Development details

We use Python/Twisted [32], an event-driven networking development framework written in Python, in the implementation. An event-driven framework is highly efficient and offers clean abstraction to model the application requirements of complex networked applications. More details are available on the website referenced above.

A high-level description of the software architecture of a broker is given in Fig. 8. As discussed in Section 3, brokers are responsible for the majority of the system functions. Fig. 8 shows four important functions: *message forwarding and dispatching*, *subscription processing*, *neighbor state management*, and *message reception*. These functions are completed by various internal modules of the broker, including the content-based routing module, failure detector, publication endpoint, and subscription endpoint. All of these

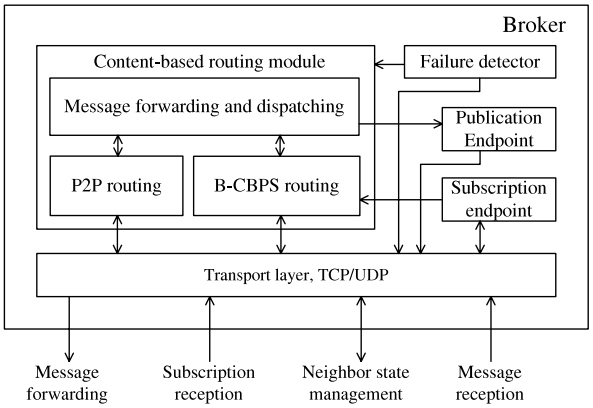


Fig. 8. High-level software architecture of a broker.

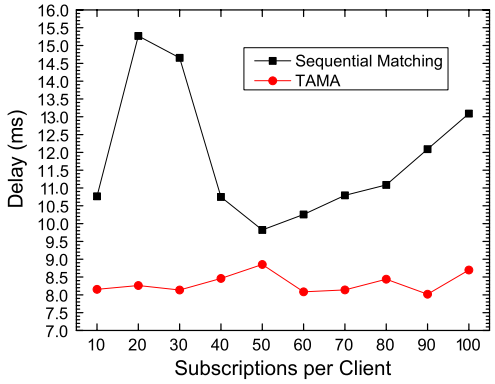


Fig. 9. The delay of the delivered messages on the PlanetLab testbed.

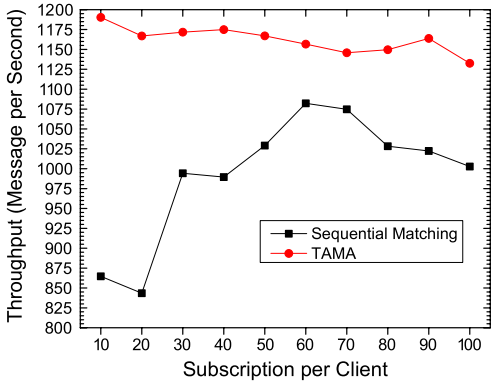


Fig. 10. The throughput on the PlanetLab testbed.

modules rely on a transport layer protocol, and TCP is used in the prototype. End users or clients are much simpler. They implement three modules: *publish*, *receive*, and *subscribe*, which are responsible for pushing and receiving messages to and from brokers and sending subscriptions to brokers.

In the source code, a broker has two components: a *peer manager* and a *client manager*. The peer manager is responsible from message forwarding, subscription processing, neighbor state management. The client manager is responsible for message reception and dispatching. The complete source code is available on [27]. Our design can be integrated with most broadcast-based CBPS systems. The efficiency of our design is best with broadcast-based content routing because subscriptions do not need to be forwarded when maintaining P2P routing tables. If, however, it is implemented with hierarchical CBPS systems, like in [10], some subscriptions need to be installed by the P2P routing protocol.

4.2. Prototype evaluation

We allocate 70 PlanetLab nodes as brokers, which are all located in North America. Clients are implemented as local processes on the broker node. On each of the PlanetLab nodes, 100 client processes are allocated. Clients are connected with the broker that resides on the same physical machine, which are identified through different port numbers. Each client subscribes to the local broker with at least 100 subscriptions, and the amount changes to model different network scales. All subscriptions only have 1 attribute. Note that this is not a realistic setting. The reason is to accurately model the impact of the number of subscriptions on the system performance. That is, it is necessary to check all subscriptions to match an incoming message, which cannot be filtered by examining the schema of the subscription and the message.

Each broker node connects with 6 nodes, which have the lowest communication delay with the broker node, as the neighbors in the B-CBPS infrastructure. The same number of finger links are used in the P-CBPS layer. We study the delay and throughput of our system. All subscriptions are intervals that randomly picked from a global limit of [0, 10,000]. The lower and upper bounds of an interval is obtained by sorting two random numbers drawn from [0, 10,000]. Each message has no actual payload and only has the content description and a timestamp. Note that this setting avoids the impact of transmission delay on the overall network performance. The evaluation is done like this: after all clients subscribe to brokers, a client is selected to publish data. Due to the symmetric of the network, this suffices to evaluate the overall network performance.

The delay and throughput of our system without failure are presented in Figs. 9 and 10. The hardware and network configurations of PlanetLab servers are powerful. As presented in the Fig. 9, the delay of delivering a message is low. We present the results of the two matching techniques: sequential matching, which is the naive approach that examines all subscriptions one by one and stops when an interface is matched; and TAMA, which is the approximate matching method presented in [35]. TAMA slightly reduce the overall delay because of its faster matching. We note a spike in the delay of the sequential matching. This is because when the amount of subscriptions in the whole network is small, brokers have to examine almost all subscriptions to determine whether or not to forward a message. This causes the brokers to examine more subscriptions when the subscriptions amount increases. When the subscription amount increases beyond a threshold, the shortcut effect, i.e. the matching process stops early once a subscription is matched for an interface, reduces the number of subscriptions need to be examined. The delay still increase afterwards due to the sheer increase of the amount of subscriptions.

The throughput is presented in Fig. 10. We also include the results of two matching techniques. Similar as Fig. 9, TAMA performs better than sequential matching. The correspondence between delay and throughput is not strictly well regulated. In this test, we let a client inject as many messages as possible to its connected broker, and then measure the delivered messages of that client. Since we let messages to have only a timestamp the primary bottleneck for throughput is the processing speed of the broker and the underlying routing infrastructure. There is a small drop in the throughput for the sequential matching method, which is corresponding to the spike in the delay as shown in the previous figure. Due to the fact that the subscriptions are generated in a purely random fashion, the test results show some irregularity between 60 and 70 subscriptions per client. We believe this is most

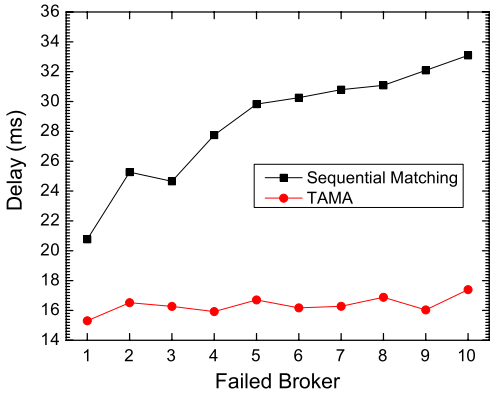


Fig. 11. The delay of delivered messages on the PlanetLab testbed with growing number of failed brokers.

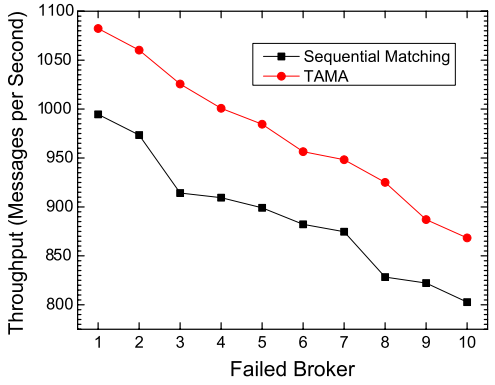


Fig. 12. The throughput on the PlanetLab testbed with growing number of failed brokers.

likely a result of the underlying physical network status and the randomness of the subscriptions.

Figs. 11 and 12 present the delay and the throughput of two matching techniques with broker failures. The number of failed brokers is selected from 1 through 10. The results are consistent with the analysis we gave previously.

5. Performance evaluation

Simulations are used to evaluate our designs. We implement an abstract network simulator that is able to capture the properties of the transport layer, which is much simpler to implement than a comprehensive packet-level simulator. In this section, we first present the details of the settings of the simulations and then discuss the details of the simulation results.

5.1. Simulation settings

A customized simulation program written with C++ is used. We use the topology generation process described in [33]. An AS-level (Autonomous System) topology is generalized from real AS-level connection statistics based on the method given in [9], and then a per-AS router topology is populated for each AS by using the method from [18]. This complete Internet topology is not directly used in the simulation because of its excessive complexity, but is used as a basis to derive an inter-AS topology. Inter-AS links are 10 Gbps; inner-AS links are 1 Gbps; and users connect with the Internet with 15 Mbps LAN, which is consistent with the mainstream ISP service specifications. The bandwidths and latencies between end users and brokers are derived from the generated topology and link connections. Propagation delays are omitted.

Each broker is assigned to an AS, so that the connections between brokers are implicitly determined by the AS-level topology. The number of brokers varies from 100 to 1000 to simulate different scales of the system. 1000 end users are assigned to each broker. Each end user subscribes to one subscription that corresponds to a small range in a one-dimensional attribute space and is drawn uniformly. Subscriptions are pre-assigned and keep static during the entire simulation. The width of the range is set to be $\frac{1}{1000}$ of the entire value space of the attribute. Considering the large amount of end users in the entire system, there are, on average, 100 to 1000 subscribers for each message to be delivered to, for the network of 100 to 1000 brokers. We let each end user send messages at an average rate of one message per hour. This rate is implemented as an exponential inter-transmission time that has an mean value of one hour. We also vary this rate to simulate different traffic loads on the system. Each message is 1 MB in size.

Methods in Comparison: We compare our designs with two other methods: B-CBPS-Reconfig and P-CBPS. The former stands for the B-CBPS infrastructure with the reconfiguration protocol in [16], which reconfigures the routing tables of all brokers in the network when failures occur. There are optimizations made to accelerate the reconfiguration process and to reduce the induced signaling overhead. P-CBPS is a Chord DHT network. We make sure that the B-CBPS infrastructure and the P-CBPS layer of our system is identical to the B-CPBS-Reconfig and P-CBPS topologies used in comparison. We also use Rendezvous and Multicast to represent the two P2P routing schemes.

5.2. Throughput

We first look at the throughput of the system in the presence of failures. We record 100 s of running states of the system. The delivered messages in every second are accumulated to calculate the throughput. We only consider the unsaturated case where the network bandwidth is enough to accommodate all traffic loads. There are 100 brokers in the system. We randomly remove a broker from the network and let it rejoin 5 s later, at the 10th and 70th second. Note that topology reconfiguration is not executed by our design. Fig. 13 presents the achieved throughput at every second. Throughput is disrupted by the failure. All protocols perform closely. P-CBPS clearly has the lowest amount of disruptions, and B-CBPS-Reconfig has the highest amount of disruptions. This is because B-CBPS-Reconfig will seize transmission when a failure occurs and will only resume after reconfiguration is completed. In a network of 100 brokers, the time used to complete reconfiguration could be acceptable, as demonstrated in the experiments.

To better understand the disruption of failures on throughput in a large-scale network, we change the count of brokers to 1000 and repeat the above experiments. To avoid saturating the network, we reduce the size of the message to 100 kB, so that the overall throughput is similar to the previous experiment. We randomly remove 10 brokers at the 10th and 70th second and let them rejoin the network 5 s later. This is to ensure the same fraction of failures as in the network of 100 brokers. The results are shown in Fig. 14. B-CBPS-Reconfig has much higher delays and much longer disruptions compared to other protocols. Since the network becomes larger, the time used to reconfigure the routing tables of brokers becomes much longer than a smaller network. Whereas for our system, the reconfiguration process is shielded from the end users by forwarding traffic on the P2P routing layer. The disruptions in our system are caused by the initial setup time of the P2P routing and the longer delays of forwarding messages on the P2P layer.

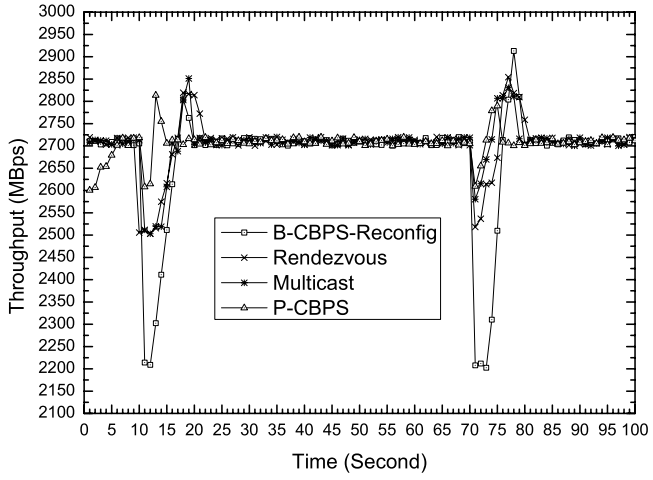


Fig. 13. The aggregate throughput of the system calculated in a 1 s time window in a network of 100 brokers using different protocols.

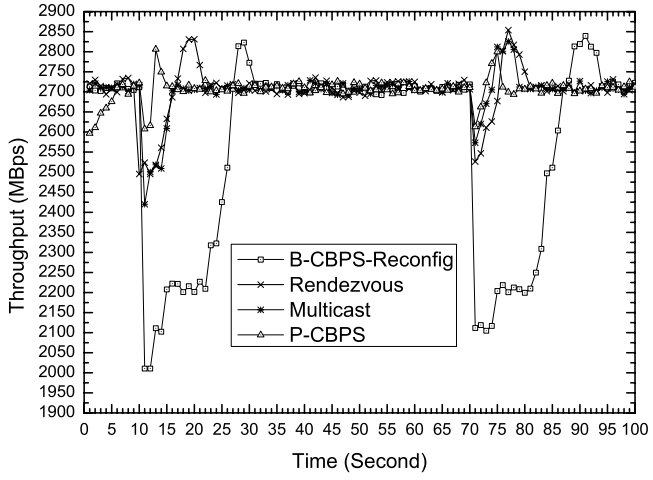


Fig. 14. The aggregate throughput of the system calculated in a 1 s time window a large network of 1000 brokers using different protocols.

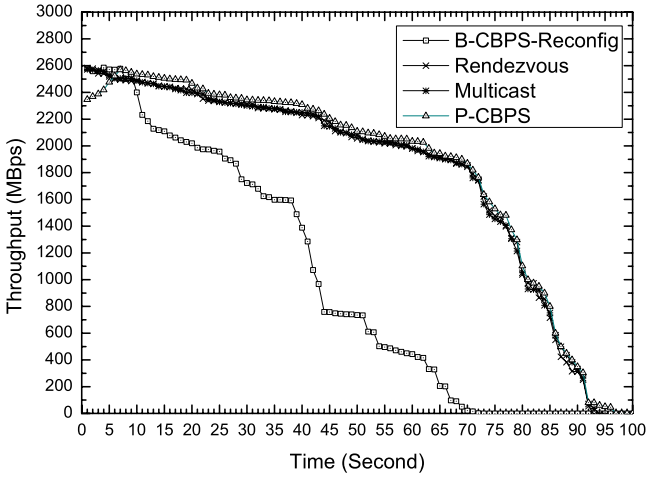


Fig. 15. The average delay calculated in a 1 s time window change with broker failures. 100 brokers. Each second a broker is removed from the network starting from the 10th second.

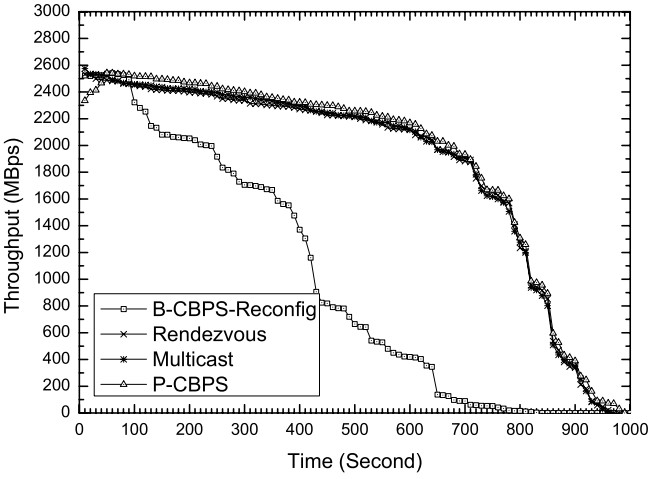


Fig. 16. The average delay calculated in a 10 s time window change with broker failures. 1000 brokers. Each second a broker is removed from the network starting from the 10th second.

We then measure the achieved throughput when a fraction of the brokers fail. Starting from the 10th second, a broker will be permanently removed from the network in every 10 s. The network has 100 brokers. Similar to the above, we measure the achieved throughput in a 1 s time window. The results are shown in Figs. 15 and 16. Fig. 15 shows the results for a network of 100 brokers. Our system and P-CBPS perform very closely. P-CBPS works best because the failures cause less path failures, so that more traffic will be forwarded normally without being disrupted. On the other hand, since our system tries to switch back to B-CBPS after a timeout, which is set to 5 s in this experiment, a fraction of the bandwidth is wasted on exchanging control messages. Additionally, with more brokers being removed, more traffic will be forwarded on the P2P layer, which has longer routing paths and also reduces the short-term throughput. As we can see in the figure, the difference between our system and P-CBPS almost disappears after the 75th second. This is because, after a sufficiently long time, the effects of long routing paths in the P2P routing layer are settled. Also, because a large fraction of the brokers are already removed, most of the traffic will be forwarded on the P2P routing layer, which makes our system perform almost identically to P-CBPS.

B-CBPS-Reconfig has a inferior performance compared to other protocols. This obviously results from the fragile network structure, which causes more disruptions and cannot be repaired quickly. There are large drops in throughput at the 10th, 45th, 50th, and 62nd second, which is resulted from network partitioning caused by removing brokers.

We also notice that at the beginning of the simulation, P-CBPS has a noticeable performance gap compared to other protocols. This is because the delay of forwarding messages in a P2P network is large; some of the traffic is not accumulated in real-time, which makes the throughput less than others. From the beginning to the 10th second, its throughput is also larger than the others, for some of the traffic is accumulated and added into the throughput, which results in a larger throughput. We also would like to point out that, the delay of delivered messages using P-CBPS is larger than the others; details will be given in the next section.

We also repeat the above experiment in a network of 1000 brokers. The results are presented in Fig. 16. The message size is reduced to 100 kB, and the time scale is prolonged to 1000 s. We remove brokers starting at the 100th second. The results are very similar to those in Fig. 15. Our analysis of Fig. 15 also applies here.

5.3. Delay

We then study the delay of the delivered messages of the different protocols. Figs. 17 and 18 are the corresponding delays of

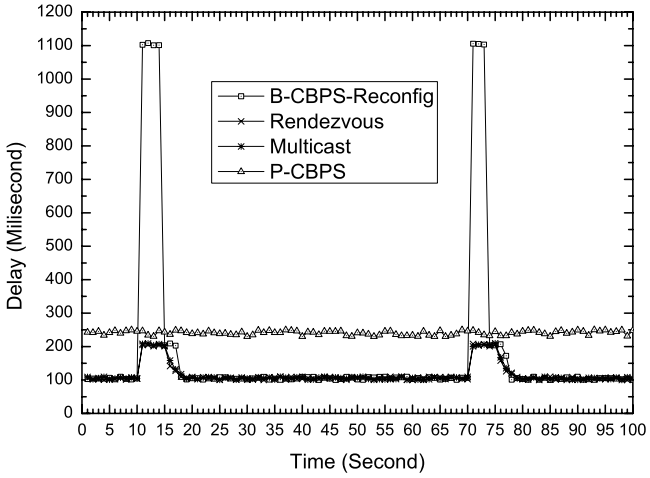


Fig. 17. The average delay of the delivered messages calculated in a 1 s time window in a network of 100 brokers using different protocols.

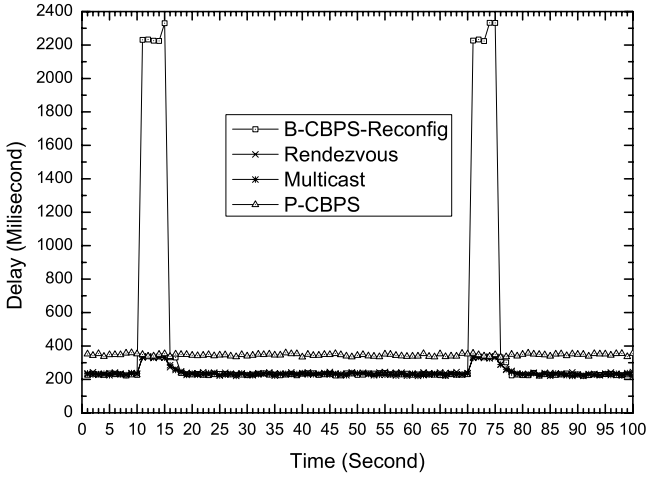


Fig. 18. The average delay of the delivered messages calculated in a 1 s time window in a network of 1000 brokers using different protocols.

the figures in the last section. At the end of every second, we find the maximal delay of all of the messages received by subscribers in the last second. In normal operations, the delays achieved by P-CBPS the largest, which is caused by the relatively longer routing paths in P2P routing. When a failure occurs, our system quickly adapts the routing protocol to maintain a low delay. On the contrary, B-CBPS-Reconfig needs to wait until the reconfiguration is completed. The resultant delay is a order of magnitude larger than the other protocols.

The delay of P-CBPS is much larger than the other protocols in normal operations. This makes it not a good choice when a small delivery delay is required by the application. However, this gap is reduced in a larger network, as shown in Fig. 18. This fact suggests that P2P-based routing would outperform broadcast-based routing when the network scale is beyond a certain threshold. For the application scenarios considered in this paper, this will not happen.

Note also that the delays of Rendezvous-based and Multicast-based P2P routing are very close. This is because both methods react to failures without any delay and rely on the same P2P routing structure for forwarding messages. There is a small gap between P-CBPS and our methods. Note that our P2P routing methods are not exactly the same as normal P2P routing. Multiple messages can be forwarded on a single path despite multiple

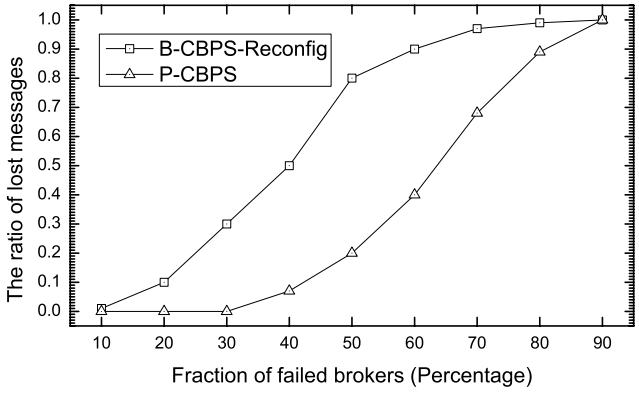


Fig. 19. The ratio of lost messages in a network of 100 brokers. The percentage of failed brokers varies from 10% to 90%.

different destinations. In pure P-CBPS, messages sent to different destinations will be forwarded on distinctive paths, which causes excessive delay.

5.4. Lost messages and overhead

Fig. 19 presents the ratio of lost messages when brokers failed. Since our system relies on the P2P routing layer to achieve maximal fault-tolerance, the ratio of the lost messages to the overall messages that should be delivered will be the same as P-CBPS. Therefore, we use P-CBPS to represent our system and the pure P-CBPS system. The figure shows that P2P-based routing reduces message loss by a large gap. This result complies with the conclusion in [13].

We only consider the overhead of the two routing methods used in our system. The overhead of other protocols involves different settings and is inappropriate to compare together. Rendezvous-based routing forwards duplicate messages to subscribers. By measuring the results of the previous experiments, we found that, on average, 30% more messages are transmitted by using Rendezvous-based routing compared to Multicast-based routing.

6. Related work

CBPS systems can be implemented on top of broadcast-based overlays [4,5,11,23] and P2P-based overlays [14,24,30,36]. Efficient routing protocol design is an important problem in broadcast-based overlays. The designs of [5,11,23] use a flat overlay network of brokers and broadcast-based dissemination. They offer distinctive reliability guarantees. DHT-based CBPS systems [14,24, 30,36] propose various mapping schemes between the content space and the ID space. Specifically, Chord [29] has been used in [24,30,36]. Such systems target large-scale wide-area systems, which emphasizes the scalability and resilience of the system instead of the delivery performance.

Reliability in CBPS systems has been studied in a lot of previous research, both in broadcast-based overlays [2,3,7,8,10,15–17,20, 22,26,34] and P2P-based overlays [12,25]. In [8], the epidemic algorithm is employed to disseminate potentially lost messages to end users. The epidemic algorithm is simple to implement, but wastes bandwidth because duplicate messages will be transmitted. Additionally, it only recovers part of the lost messages. In [10], a hierarchical multi-layer overlay network is used to coordinate reliable transmissions in a wide area network. The reliability is supported by constructing a multi-parent distribution tree and reliability-aware routing. Their approach is centralized and complicated, which makes it less favorable in practice. In [2], the authors present an abstract log-based reliable message delivery

method. It requires complex processing and reliable brokers, which limits its performance. In [16], the authors propose to cache topology information to repair routing paths. This approach causes quadratic storage overhead and excessive delays in forwarding messages.

The reliability of the P2P-based CBPS systems is implemented by using DHT to scalably replicate brokers [25] or by utilizing the inherent robustness and fault-tolerance of the P2P overlay [12]. The reliability of DHT-based P2P networks is a well-studied topic. In [13], the impact of different DHT routing structures on routing resilience is studied. Their results verify that Chord [29] has the best overall performance in terms of routing efficiency and resilience. In [19], the authors analyze the graph theoretical properties of many DHT-based P2P routing algorithms.

7. Conclusion

This paper presents the design and evaluation of a hybrid CBPS system that provides a high degree of reliability and high performance. We observe that the designs of the broadcast-based and P2P-based CBPS systems have distinctive goals that optimize different metrics. Specifically, delivery performance is good in B-CBPS whereas reliability and robustness are maximized in P-CBPS. Our system works by utilizing these two architectures alternatively. The system achieves a good performance in normal operations and in the presence of failures, both short-term and long-term failures, by switching to the appropriate routing structure that works best for the traffic demands. The basic idea is to combine a broadcast-based infrastructure with a P2P-based backup routing structure. Due to the inherent high reliability and fault resilience of DHT-based P2P routing, the system is able to route messages in the presence of failures with a very small performance degradation and avoid complex operations to regain the correct states, which also greatly reduces the overall overhead. Two P2P-based routing protocols are proposed to handle failures with simple operations and minimal overheads. Extensive simulation experiments are carried out to evaluate our designs. The results verify our claim that the system achieves good performance across a wide range of failure scenarios and only incurs a small amount of overhead. Our future work will involve implementing the system on PlanetLab to study its performance in a real-world environment.

Acknowledgments

This research was supported in part by NSF grants ECCS 1128209, CNS 10655444, CCF 1028167, CNS 0948184, and CCF 0830289, and Research Program of China (973) No. 2009CB320705.

References

[1] B. Beverly Yang, H. Garcia-Molina, Designing a super-peer network, in: Proceedings of 2003 International Conference on Data Engineering, 2003, pp. 49–60. doi:http://dx.doi.org/10.1109/ICDE.2003.1260781.

[2] S. Bhola, R.E. Strom, S. Bagchi, Y. Zhao, J.S. Auerbach, Exactly-once delivery in a content-based publish–subscribe system, in: Proceedings of the 2002 International Conference on Dependable Systems and Networks, DSN'02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 7–16. URL http://dl.acm.org/citation.cfm?id=647883.738567.

[3] S. Bhola, Y. Zhao, J. Auerbach, Scalably supporting durable subscriptions in a publish/subscribe system, in: Dependable Systems and Networks, International Conference on 0, 2003, 57. doi: http://doi.ieeecomputersociety.org/10.1109/DSN.2003.1209916.

[4] F. Cao, J.P. Singh, Medym: match-early with dynamic multicast for content-based publish–subscribe networks, in: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware, Middleware'05, Springer-Verlag New York, Inc., New York, NY, USA, 2005, pp. 292–313. URL http://dl.acm.org/citation.cfm?id=1515890.1515905.

[5] A. Carzaniga, D.S. Rosenblum, A.L. Wolf, Design and evaluation of a wide-area event notification service, ACM Trans. Comput. Syst. 19 (2001) 332–383. URL http://doi.acm.org/10.1145/380749.380767.

[6] A. Carzaniga, A.L. Wolf, Forwarding in a content-based network, in: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM'03, ACM, New York, NY, USA, 2003, pp. 163–174. doi: http://doi.acm.org/10.1145/863955.863975.

[7] R. Chand, P. Felber, Xnet: a reliable content-based publish/subscribe system, in: Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, SRDS'04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 264–273. URL http://dl.acm.org/citation.cfm?id=1032662.1034371.

[8] P. Costa, M. Migliavacca, G.P. Picco, G. Cugola, Introducing reliability in content-based publish–subscribe through epidemic algorithms, in: Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, DEBS'03, ACM, New York, NY, USA, 2003, pp. 1–8. doi: http://doi.acm.org/10.1145/966618.966629.

[9] X. Dimitropoulos, D. Krioukov, A. Vahdat, G. Riley, Graph annotations in modeling complex network topologies, ACM Transactions on Modeling and Computer Simulation (TOMACS) 4 (2009) 17.

[10] C. Esposito, D. Cotroneo, A. Gokhale, Reliable publish/subscribe middleware for time-sensitive internet-scale applications, in: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS'09, ACM, New York, NY, USA, 2009, pp. 16:1–16:12. doi: http://doi.acm.org/10.1145/1619258.1619280.

[11] E. Fidler, H.A. Jacobsen, G. Li, S. Mankovski, The padres distributed publish/subscribe system, in: In 8th International Conference on Feature Interactions in Telecommunications and Software Systems, 2005, pp. 12–30.

[12] A.E., M. Gradinariu, A.K. Datta, G. Simon, A. Virgillito, A semantic overlay for self-* peer-to-peer publish/subscribe, in: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, ICDCS'06, IEEE Computer Society, Washington, DC, USA, 2006, p. 22. doi: http://dx.doi.org/10.1109/ICDCS.2006.12.

[13] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoica, The impact of DHT routing geometry on resilience and proximity, in: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM'03, ACM, New York, NY, USA, 2003, pp. 381–394. doi: http://doi.acm.org/10.1145/863955.863998.

[14] A. Gupta, O.D. Sahin, D. Agrawal, A.E. Abbadi, Meghdoot: content-based publish/subscribe over p2p networks, in: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, Middleware'04, Springer-Verlag New York, Inc., New York, NY, USA, 2004, pp. 254–273. URL http://dl.acm.org/citation.cfm?id=1045658.1045677.

[15] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, A fast and robust content-based publish/subscribe architecture, in: Proceedings of the 2008 Seventh IEEE International Symposium on Network Computing and Applications, IEEE Computer Society, Washington, DC, USA, 2008, pp. 52–59. URL http://dl.acm.org/citation.cfm?id=1443222.1443610.

[16] R.S. Kazemzadeh, H.-A. Jacobsen, Reliable and highly available distributed publish/subscribe service, in: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems, IEEE Computer Society, Washington, DC, USA, 2009, pp. 41–50. URL http://dl.acm.org/citation.cfm?id=1637865.1638331.

[17] R.S. Kazemzadeh, H.-A. Jacobsen, Partition-tolerant distributed publish/subscribe systems, Reliable Distributed Systems, in: IEEE Symposium on 0, 2011, pp. 101–110. doi: http://doi.ieeecomputersociety.org/10.1109/SRDS.2011.21.

[18] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, M. Bogu?, Hyperbolic geometry of complex networks, Phys. Rev. E (036106).

[19] D. Loguinov, A. Kumar, V. Rai, S. Ganesh, Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience, in: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM'03, ACM, New York, NY, USA, 2003, pp. 395–406. doi: http://doi.acm.org/10.1145/863955.863999.

[20] G. Muhl, M.A. Jaeger, K. Herrmann, T. Weis, A. Ulbrich, L. Fiege, Self-stabilizing publish/subscribe systems: algorithms and evaluation, in: 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2006, in: LNCS, vol. 4269, Springer, 2005, pp. 233–238.

[21] G. Pardo-Castellote, Omg data-distribution service: architectural overview, in: Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDCSW'03, IEEE Computer Society, Washington, DC, USA, 2003, p. 200. URL http://dl.acm.org/citation.cfm?id=839280.840571.

[22] G.P. Picco, G. Cugola, A.L. Murphy, Efficient content-based event dispatching in the presence of topological reconfiguration, in: Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDCS'03, IEEE Computer Society, Washington, DC, USA, 2003, p. 234. URL http://dl.acm.org/citation.cfm?id=850929.851945.

[23] P.R. Pietzuch, J. Bacon, Hermes: A distributed event-based middleware architecture, in: Proceedings of the 22nd International Conference on Distributed Computing Systems, ICDCSW'02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 611–618. URL http://dl.acm.org/citation.cfm?id=646854.708058.

[24] P.R. Pietzuch, J. Bacon, Peer-to-peer overlay broker networks in an event-based middleware, in: Proceedings of the 2nd International Workshop on Distributed Event-based Systems, DEBS'03, ACM, New York, NY, USA, 2003, pp. 1–8. doi: http://doi.acm.org/10.1145/966618.966628.

[25] M.R. Selim, Y. Goto, J. Cheng, A replication oriented approach to event based middleware over structured peer to peer networks, in: Proceedings of the 5th International Workshop on Middleware for Pervasive and Ad-hoc Computing: Held at the ACM/IFIP/USENIX 8th International Middleware Conference, MPAC'07, ACM, New York, NY, USA, 2007, pp. 61–66. doi: http://doi.acm.org/10.1145/1376866.1376877.

[26] A.C. Snoeren, K. Conley, D.K. Gifford, Mesh-based content routing using xml, SIGOPS Oper. Syst. Rev. 35 (2001) 160–173.
doi: http://doi.acm.org/10.1145/502059.502050.
[27] Source code of the hybrid CBPS architecture.
https://github.com/justicezyx/cbps.
[28] N. Spring, L. Peterson, A. Bavier, V. Pai, Using planetlab for network research: myths, realities, and best practices, SIGOPS Oper. Syst. Rev. 40 (2006) 17–24.
doi: http://doi.acm.org/10.1145/1113361.1113368.
[29] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Trans. Netw. 11 (2003) 17–32.
doi: http://dx.doi.org/10.1109/TNET.2002.808407.
[30] D.K. Tam, R. Azimi, H.-A. Jacobsen, Building content-based publish/subscribe systems with distributed hash tables, in: K. Aberer, V. Kalogeraki, M. Koubarakis (Eds.), DBISP2P, in: Lecture Notes in Computer Science, vol. 2944, Springer, 2003, pp. 138–152.
URL http://dblp.uni-trier.de/db/conf/dbisp2p/dbisp2p2003.html#TamAJ03.
[31] Marketcetera, Open source trading platform, 2012.
URL http://www.marketcetera.com/.
[32] Twisted matrix labs. URL http://twistedmatrix.com/.
[33] CAIDA, Internet topology at router- and as-levels, and the dual router+as internet topology generator.
URL http://www.caida.org/research/topology/generator/.
[34] Y. Zhao, D. Sturman, S. Bhola, Subscription propagation in highly-available publish/subscribe middleware, in: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, Middleware'04, Springer-Verlag New York, Inc., New York, NY, USA, 2004, pp. 274–293.
URL http://dl.acm.org/citation.cfm?id=1045658.1045678.
[35] Y. Zhao, J. Wu, Towards approximate event processing in a large-scale content-based network, in: Distributed Computing Systems (ICDCS), 2011 31st International Conference on, 2011, pp. 790–799.
doi: http://dx.doi.org/10.1109/ICDCS.2011.67.
[36] Y. Zhu, Y. Hu, Ferry: a p2p-based architecture for content-based publish/subscribe services, IEEE Trans. Parallel Distrib. Syst. 18 (2007) 672–685.
doi: http://dx.doi.org/10.1109/TPDS.2007.1012.

Yaxiong Zhao received his B.E. degree in Transportation Engineering from Tongji University, Shanghai, China, in 2005; and M.E. degree in Software Engineering from Tsinghua University, Beijing, China, in 2008. He is currently a Ph.D. candidate in the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. His research interests are in the design and implementation of various novel application and protocols of computer networks. He has authored 9 papers on wireless sensor networks, delay tolerant networks, and content-based networks. His current research focuses on the efficient communication between human carried wireless devices and Internet-scale information-centric content dissemination systems.

Jie Wu (F'09) received his B.S. in computer engineering and M.S. in computer science from Shanghai University of Science and Technology (now Shanghai University), Shanghai, China, in 1982 and 1985, respectively, and his Ph.D. in computer engineering from Florida Atlantic University, Boca Raton, in 1989. Jie Wu is an ACM Distinguished Speaker and an IEEE Fellow. He is a Chair and a Professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. Prior to joining Temple University, he was a Program Director with the National Science Foundation, Arlington, VA. He has published more than 550 papers in various journals and conference proceedings. His research interests include wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. Dr. Wu has served as an IEEE Computer Society Distinguished Visitor and is the Chairman of the IEEE Technical Committee on Distributed Processing (TCDP). He serves on the Editorial Board of the IEEE TRANSACTIONS ON COMPUTERS and the Journal of Parallel and Distributed Computing. He is Program Co-Chair for IEEE INFOCOM 2011. He was also General Co-Chair for IEEE MASS 2006, IEEE IPDPS 2008, and DCOSS 2009.