The Hong Kong Polytechnic University

Department of Computing

A Distributed Publish-Subscribe Architecture For XML-Based Event Dissemination

by

Xiaochuan YU

A thesis submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

June 2013

CERTIFICATE OF ORIGINATLITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

. . . .

Xiaochuan YU

June 2013

Abstract

The aim of this study is to investigate how a pub-sub architecture can be efficiently empowered with XML-based filtering capability in a distributed environment. An XML based pub-sub system mainly focus on active, selective, asynchronous dissemination of timely, personalized and dynamic information represented and modelled by XML. It supports improved filtering flexibility and expressiveness as compared to its predecessor, the topic-based and content-based pub-sub systems. In recent years, XML based pub-sub shows increasing significance as the evolution of web 2.0 technologies and the succeeding proliferation of social network services demand a better information dissemination paradigm that is suitable for information-driven type of applications. However, many existing approaches in XML-based pub-sub do not operate efficiently and do not scale well in a wide-area environment, such as the Internet. In addition, most of the studies require dedicated infrastructure support and do not concern about fault-tolerance of the system. In this thesis, we introduced a series of novel approaches that contribute toward enabling an efficient distributed XML-based pub-sub system design that scale up to operate in a wide-area and large-scale environment. Our design possesses self-organizing capability for the overlay infrastructure and supports fault-tolerance that matches the dynamism of the underlying network. In our design, we combine the process of data filtering with routing where we exploit the structure coverage relation between XML representation of subscriptions and publications. Here, the subscriptions are forwarded selectively to only a few rendezvous nodes without the need to flood subscriptions across the entire network. At the rendezvous node, the incoming publications can be evaluated against subscriptions and delivered to subscribers if matched. To achieve this, we applied a bloom filter-based filtering approach that coalesced into the addressing scheme for the Key-based Routing to provide a scalable, flexible and robust pub-sub infrastructure. In addition, we developed a hypercube overlay as a multicasting infrastructure for efficient dissemination of publications. We further extended proposed architecture by developing a redundancy-based fault-tolerance strategy to enhance the robustness of the system considering the dynamism of the underlying network environment. We validate the efficiency and scalability of the proposed system through extensive experiments. It shows that the proposed system is able to balance the load among the peers and to prune out unmatched publication messages at the early stages along the dissemination path. It also shows that the proposed system scales well with increasing number of peers, subscriptions and publication events. We also evaluated the availability and effectiveness of the fault-tolerant capability in the case of nodes churning. The evaluation result shows that proposed approach can work very well in a dynamic network environment where nodes may join, leave and fail at times.

Publications

Journal Papers

- Xiaochuan Yu and Alvin Toong Shoon Chan. Towards robust XML dissemination in large-scale dynamic network environment. Submitted to *Journal of Computer and System Sciences*, 2013.

Conference Papers

- Xiaochuan Yu and Alvin Toong Shoon Chan. Hope: A fault-tolerant distributed Pub/Sub architecture for large-scale dynamic network environment. To appear in *Proceedings of the 12th IEEE International Conference on Ubiquitous Computing and Communications*, IUCC '13, July 2013.

- Xiaochuan Yu and Alvin Toong Shoon Chan. A hypercubic overlay using bloom-filter based addressing for a non-dedicated distributed tag-based pub/sub system. To appear in *Proceedings of the 11th IEEE International Symposium on Parallel and Distributed Processing with*

Applications, ISPA '13, July 2013.

- Xiaochuan Yu and Alvin Toong Shoon Chan. A hypercubic event-dissemination overlay using structure-aware addressing for distributed xml-based pub/sub system. In *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, HPCC '12, June 2012, pages 179-186.

- Xiaochuan Yu and Alvin Toong Shoon Chan. A time/space efficient xml filtering system for mobile environment. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01*, MDM '11, June 2011, pages 184-193.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor: Professor Alvin Chan, a man with great patience, motivation, enthusiasm, self-discipline and immense knowledge. It is all the inspirational, supportive, and patient guidance he gave me that helps me move towards a qualified PH.D and become an independent researcher.

I am also grateful to my parents, who cultivated my love of adventure, explore and my passion in pursuing the truth, all of which finds a place in this thesis.

Last but not least important, I would like to thank my uncle, who has always been a role model for me. I would not have decided to pursue a career in research if it were not for him.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The aim of this study is to investigate how a pub-sub architecture can be efficiently empowered with XML-based filtering capability in a distributed environment. In this chapter, in section 1.1, we first describe the *motivation* behind this research, followed in section 1.2 by a brief discussion on the *shortcomings of current approaches.* Then, in section 1.3, we provide a brief summary of our *contribution.* A brief introduction of the *methodology* adopted in this research is presented in section 1.4. We end this chapter by illustrating *the organization of this thesis* in section 1.5.

1.1 Motivation

As we enter the era of big data, the volume of information on the Internet is exploding due to the increasing number of Internet users, mobile devices, pervasive computing devices, and social networking activities.

As Internet services continue to infiltrate into the social fabric of our everyday lives, the need to efficiently deliver personalized information on an as-available basis is becoming increasingly important. In order to filter and deliver relevant data from the massive amount of messages to interested users or software components in a timely manner, architecting an efficient information dissemination system is important, to say the least. The architecture should intrinsically support the efficient collection and integration of the various kinds of data that are distributed among large sets of users and application services, while providing the capacity to actively disseminate personalized information on an as-available basis through a “push-based” approach.

The traditional *request-reply* paradigm, which has been widely adopted as the distributed computing paradigm of choice, is not able to address the aforementioned requirements. This is because the request-reply approach is inherently based on a “pull-based” approach, while selective information dissemination requires a “push-based” communication strategy. In order to achieve selective information delivery using a request-reply style of communication, a client has to continuously poll the server for any updated information. Although such a mode of operation works well in a local area network environment with a limited number of clients and servers, the scaling up of the system would lead to increasing performance penalties. Clearly, the performance degradation becomes especially severe in a wide-area network such as the Internet. For example, the well-known social microblogging service Twitter [65] has suffered significant availability issues from traffic overloads and hardware failures largely due to the inefficiency of this polling mechanism [47, 6]. More importantly, as the Internet continues to take root in services such as pervasive and social computing, among many others, the need for a communication paradigm that supports the efficient dissemination of traffic in the form of many-to-many communication is apparent. In such a communication environment, it is the norm rather than the exception that a client often does not get to decide which end-point server to connect to and receive messages from, as in a one-to-one request-reply paradigm. In this thesis, we argue that there is a need to adopt a *publish-subscribe* (*pub-sub*) communication paradigm that is robust enough to be used across myriads of Internet services, while providing the capacity for the system to scale-up to operate in an Internet-scale environment.

In pub-sub, users indicate their interests by submitting *subscriptions* to the system. Published information (i.e., *publications),* when available, is automatically propagated and delivered to the interested peers based on subscriptions that they have submitted. Pub-sub provides a more intuitive and efficient communication model by actively and selectively disseminating information based on user's interests through a "push-based" approach. More importantly, it enables loose coupling between the data source and the sink. In a dynamic environment where clients continuously join and leave, while servers may fail at times, pub-sub can effectively handle exchanges of data among a very large number of entities without requiring all of the information sources and sinks to be present in the network at the same time. This in turn enables the decoupling of application logic and communication, which can ease the application design process considerably.

1.2 Shortcomings of Current Approaches

The earliest pub-sub systems are mainly topic/channel-based [5, 44, 48]. This has subsequently led to the development of pub-sub systems that support content-based matching schemes [2], which are more flexible and expressive. The common adoption of XML as a standardized mark-up language for data interchanges has stimulated significant research interest in integrating and leveraging the expressiveness of XML data to effectively operate within the pub-sub paradigm. This has led to the development of XML-based pub-sub systems [45, 16, 58, 36, 22, 12, 34, 17, 37, 38, 9, 2]. In an XML-based pub-sub system, published events are represented in the form of XML documents. The subscriptions in these systems are expressed in a powerful filtering language such as XPath [66] or XQuery [67], which is used to specify constraints over both structure (using path expressions) and content (using value-based predicates). Compared to previous approaches, XML-based pub-sub provides better expressiveness in filter expression, as the structured XML representation can embed richer semantic information in users' subscriptions. It is also more flexible and exhibits scalability, as it does not require predefined topics or attributes. However and significantly, existing XML-based pub-sub systems suffer from a number of performance issues when applied to large-scale networks such as the Internet, not to mention the dynamic nature of such an environment. Among those issues, the following are the most crucial:

Filter Efficiency. Most studies [16, 58, 36, 22, 12, 34] on XML-based pub-sub have primarily focused on improving the semantic richness of filter expression and enhancing filtering precision with a trade-off in computation complexity. The overhead incurred in performing filtering to attain the desirable level of precision often resulted in computational inefficiency. In a distributed pub-sub system, filters are normally deployed at each broker in the system. The filtering and matching process is triggered each time an incoming message arrives at the broker. As a result, the filtering process is triggered repeatedly on every broker along the paths traversed by each publication. Hence, the computational load on each broker increases exponentially as the system scales up. Consequently, if the filtering process is not efficient enough, the brokers can easily become overloaded, leading to potential system failure.

Robustness. Studies on distributed XML filtering systems normally assume the use of a spanning tree based routing approach, which is widely used in conventional topic or content-based pub-sub systems. Normally, the tree is constructed by identifying subscriber nodes that share common subscriptions and are in close proximity to each other, while minimizing the messages transversal path to all the other subscribed nodes. The computation requirement for forming the tree dictates that the subscription among the nodes does not change frequently, and those nodes do not constantly enter or leave the subscription tree. As such, as the dynamism and the changes in the network topology increase, the system will suffer from a severe degradation in performance. Such an assumption is clearly invalidated in Internet applications where it is normal for potential nodes to join, leave, or fail, and subscription updates happen frequently.

Scalability. As mentioned, the spanning tree-based routing scheme is used by many existing distributed pub-sub systems for the dissemination of subscribed messages from publishers to subscribers. The scalability is severely limited due to the high computational requirement of establishing and maintaining the underlying tree structure and the flooding-alike propagation of subscriptions. Thus far, all of the studies in this area have focused only on applying XML-based pub-sub in a small network of brokers with a small to medium number of nodes. As pervasive computing over the Internet continues to grow, it is clear that future pub-sub systems must handle a far larger number of published messages and subscriptions that operate across a diverse number of applications. Therefore, it is not uncommon to expect pub-sub systems to support thousands if not millions of users.

Level of integration. To date, the studies in this area have treated the filtering and routing of publications separately. Hence, they have either focused solely on filtering while assuming that the routing issue has already been taken care of [16, 58, 36, 22, 12, 34], or have focused on the routing problem using existing filtering schemes [17, 37, 38]. However, treating the matching and routing processes separately can hardly lead to an optimal solution, as these two processes are tightly bound to each other. The routing scheme dictates the requirements of filtering scheme, while the filtering scheme will have an impact on the performance of the routing at large. Therefore, in a well-designed system, the filtering and routing processes should synergistically complement, rather than limit, each other.

Autonomy. Most studies [16, 58, 36, 22, 12, 34, 17] have required either a dedicated broker network or some dedicated hosts as part of the management facility, to provide centralized coordination and topology management services. As a distributed system scales, especially when the system spans over multiple administrative domains, difficulties might be encountered with direct administrative control. The risk of a single point failure occurring in a domain, which then propagates to other administrative domains, eventually rendering the network inoperable, is inherent in a system that requires a dedicated management facility. Hence, adopting a self-organizing distributed architecture, which requires no dedicated brokers or distinguished hosts, would be preferable with respect to reliability.

The outline of the above challenges and requirements have driven us to investigate how we can efficiently empower pub-sub architecture with the XML-based filtering capability in a distributed environment. In particular, our core task is to develop an XML-filtering system that can be easily deployed to a large collection of loosely maintained, heterogeneous and non-dedicated machines spread throughout the Internet.

1.3 Contribution

The main contributions of our work are in two areas: the *event matching*, and the *event routing*.

1.3.1 Contributions in Event Matching

The contributions with respect to event matching are:

- **A scalable and flexible distributed XML dissemination architecture.** We took both event matching and event routing into consideration, and carefully developed a distributed pub-sub system suitable for the large-scale dissemination of XML data.

- **The concept of separating parsing from the matching process.** We provide a novel architecture to relieve the computation burden of parsing from the matching process. This will enable our framework to be adapted in a computation-constrained environment such as a mobile environment.

- **Efficient structure representation by a bloom filter.** We provide a way of representing information on the structure by using a bloom filter, which originally only supported the representation of non-structured data sets.

- **Efficient matching method based on the representation of a bloom filter.** Using our method, the evaluation can be as simple as a bit-vector comparison.

1.3.2 Contributions in Event Routing

Our contributions in event routing are:

- **An addressing scheme that is optimized for the event-matching scheme.** We utilized the natural convergence between the event-matching scheme and the addressing scheme, thereby optimizing the event-routing process. The address can reveal certain structural information that can later facilitate the event-routing process.

- **Robust and scalable routing scheme.** We developed a robust and scalable routing scheme for Internet applications. Specifically, we developed a hypercube overlay by exploiting the structure-awareness of our addressing scheme. With this overlay, a multicast style of communication can be established for efficient message propagation. Our routing scheme can efficiently handle changes in topology caused by joining/leaving or failure of nodes, and also has the intrinsic capability to scale up to operate in an Internet-scale environment.

1.3.3 Contributions in Architectural Design

In our study, we have also made many contributions with regard to research methodology and architectural design, the most prominent being the following:

- **A holistic approach to handling matching and routing.** As far as we know, this is the first work to take a holistic approach to considering filtering and routing as two processes that may synergistically complement one another to support efficient message dissemination. In our approach, filtering is seamlessly coalesced into the routing scheme and can be accomplished seamlessly and efficiently as publications are propagated across the nodes.

- **An extensible multilayer architecture**. We proposed a multilayered event-dissemination architecture. With the multilayer design, flexibility and adaptability can be largely guaranteed. Modifications to support different levels of optimization and adaptation corresponding to specific layers are also ease to made.

1.4 Methodology

Our aim was to develop an XML-based pub-sub architecture suitable for applications in large-scale networks, specifically the Internet-scale applications. For the sake of effectiveness and efficiency in the evaluation of the proposed architecture through experiments, we adopted a simulation-based approach, facilitated with analysis where appropriate. The reasons behind this decision were manifold: i) Running large-scale experiments requires a substantial amount of network infrastructure which is ideally deployed in a controlled environment. ii) It would be difficult to run experiments on a scale larger than 10 fully controlled computers, since adequate infrastructure would be hard to obtain during the process of this research study. iii) Even if the infrastructure were available, technological faults independent of the proposed algorithm could have potentially slowed down the process of the research. Hence, it would be impractical to carry on real-world test, given the limited period of time available to conduct the proposed research. iv) The experiments, if implemented in an appropriate simulation environment, can be parallelized, which has been proven to be quite helpful by our later experience. v) Many simulation frameworks can provide a near-realistic environment which can easily lead to results as authentic as those in testbed experiments. In addition, reproducing the results is easy compare to real-world tests.

In our simulation, we mainly use OMNeT++ [62], an extensible, modular, component-based C++ simulation library and framework, combined with some analysis utilizing Matlab [61] when appropriate. Via detailed simulations, we have evaluated system performance extensively in terms of efficiency, scalability, overhead, and fault-tolerance capability in multi-faceted experiment setups. Evaluating the algorithms in a real-world scenario on platforms such as PlanetLab [63] would be an interesting next step, even though it may have drawbacks as mentioned earlier.

1.5 Organization of This Thesis

The topics and the organization of this thesis are depicted in Figure 1.1.

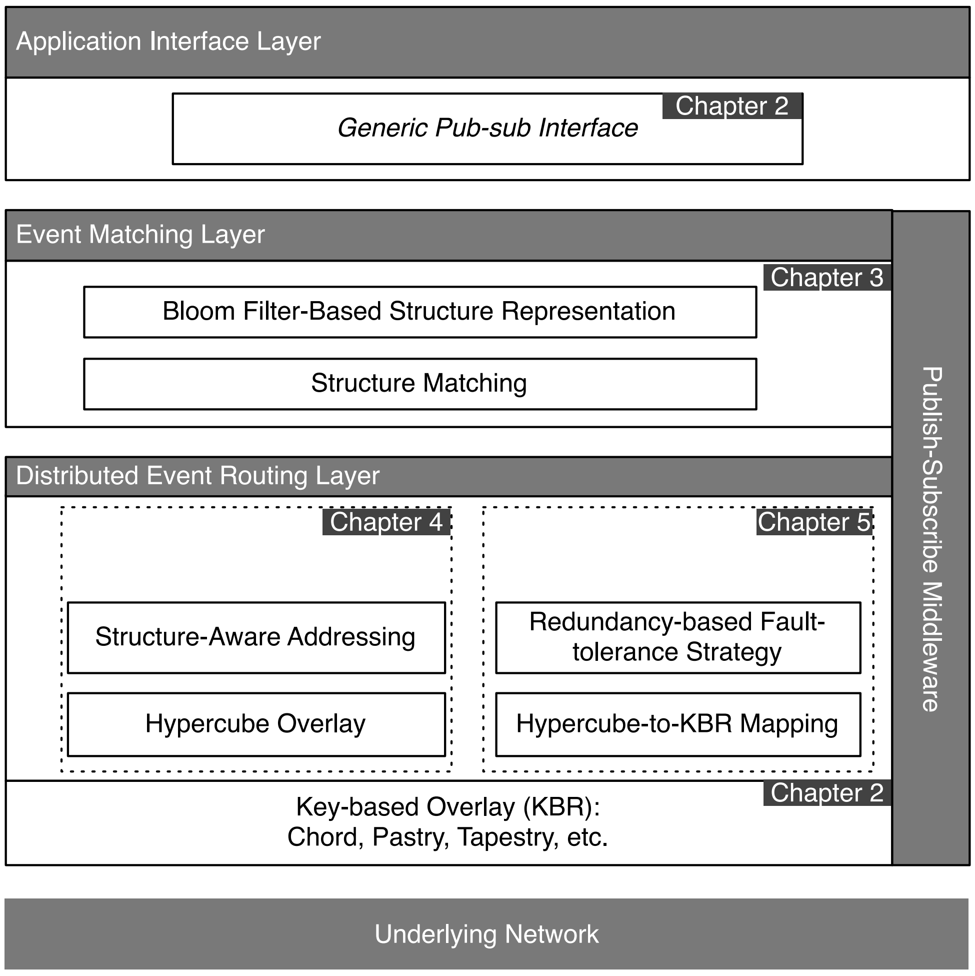

Application Interface Layer
Chapter 2
Generic Pub-sub Interface
Event Matching Layer
Chapter 3
Bloom Filter-Based Structure Representation
Structure Matching
Distributed Event Routing Layer
Chapter 4
Chapter 5
Structure-Aware Addressing
Hypercube Overlay
Redundancy-based Fault-tolerance Strategy
Hypercube-to-KBR Mapping
Chapter 2
Key-based Overlay (KBR): Chord, Pastry, Tapestry, etc.
Publish-Subscribe Middleware
Underlying Network

Figure 1.1: Topics and Organization of This Thesis in Three Layers Conforming Proposed Pub-Sub System Architecture

In Chapter 2, we provide a brief introduction to publish-subscribe system architecture and the key-based routing overlay, which forms the background of this thesis. After presenting the background, we divide our work into two parts, one is the event matching, and the other is the event routing, conforming to two major functional layers of a generic pub-sub system architecture.

In Chapter 3, we focus on the event matching. First, a highly compact bloom filter-based structure representation is introduced. Based on this representation method, we proposed a highly efficient XML-based event-matching scheme. An XML-filtering architecture is then proposed utilizing proposed event-matching scheme. Proposed architecture consists of a pre-processing component and a matching component. The former is used for document parsing and bloom filter creation, while the latter provides matching evaluation-related functions. One distinctive feature of this design is the separation of the parsing process and the matching process. This is because based on our study, the time imposed on parsing is a dominant factor of the total matching time cost. Therefore shifting the parsing process to the preprocessing component can tremendously speed up the overall matching in an XML-filtering system so as to relief the burden of parsing from the filtering engine. After the illustration of proposed architecture, we present the result of performance evaluation. Extensive experiments prove that the matching time can be significantly reduced due to the separation of matching and parsing. In addition, the space needed for structure index is tremendously reduced, attributed to the bloom filter-based compact data representation method.

In Chapter 4, we address the issues in distributed event routing. First, we present a structure-aware addressing scheme, which fully exploits the bloom filter-based structure representation proposed in Chapter 3. Using the structure-aware addressing, we adapted key-based routing approach to the event-dissemination architecture, and managed to inherit the advantages such as scalability and flexibility from the key-based routing overlay. To further improve the efficiency of the event routing process, we developed a hypercube overlay as a multicasting infrastructure for efficient dissemination of publications. Extensive experiment evaluation shows that the proposed system is able to balance the load among the peers and prune out unmatched publication messages at the early stages along the dissemination path. More importantly, the proposed system scales well with increasing number of peers, subscriptions and publication events.

In Chapter 5, we focus on the robustness of earlier presented event-dissemination overlay. By introducing a novel *hypercube-to-KBR mapping algorithm*, and a *redundancy-based fault-tolerance strategy*, we are able to provide a robust large-scale event-dissemination architecture with effective fault-tolerance capability. Our experiments reveals that proposed architecture works well not only in a large-scale application, but also in a dynamic environment where nodes may join, leave and fail at times.

In Chapter 6, we summarizes our research and presents our conclusions. Furthermore, we discuss problems which remain open and sketch areas for future work.

Chapter 2

Background

This chapter lays the foundation for an event-based system architecture. First, we illustrate the basics of the event-based system, followed by an introduction to *event matching* and *event routing* – two essential functional layers of an event notification service. After that, we will provide a brief overview of *Key-based Routing* technologies, on which the proposed distributed event-routing scheme is established.

2.1 Overview of the Event-based System

The proliferation of applications in mobile and pervasive computing has stimulated the evolution of communication paradigms. The traditional tightly coupled request-reply style can no longer satisfy the undirected and asynchronous communication requirements that new applications need. As a result, a more loosely coupled communication paradigm is proposed to fulfill this requirement. This paradigm is called *publish-subscribe* (*Pub-sub* for short, also known as the *event-based system*; hence in the rest of this thesis, the terms *pub-sub system* and *event-based system* are used interchangeably). In a pub-sub system, the message producer is not supposed to send the message to a specific message consumer. Rather, all messages are forwarded to the interested consumers by their subscriptions. Each consumer is required to express their interests using a certain machine-readable form as subscription, and to disseminate them throughout the pub-sub system as filters for the incoming messages. As a matter of fact, the addresses of the consumers are not explicitly defined; thus, the producer is not aware of the destination of the messages it has produced. Nor is the consumer aware of the source of the messages. In this way, pub-sub provides a communication method that decouples the message producer and consumer of the message. The pub-sub system can be implemented in either a centralized way or a distributed way. The distributed implementation provides good scalability and avoids the issue of a single point of failure, and thus has attracted much attention. How to filter the messages as well as how to disseminate the messages to the matched subscriber in a distributed manner are two major focuses of research on distributed pub-sub system.

An abstraction of the general architecture of a pub-sub system is shown in Figure 2.1. In a pub-sub system, subscribers are required to register their interests (also known as profiles) in the system as subscriptions. Upon receiving a message, the system, which is normally implemented as middleware, will filter the message according to the subscription and deliver the message to the matched subscribers. The interaction among the publisher, subscriber, and the pub-sub middle-ware can be realized by a set of op-

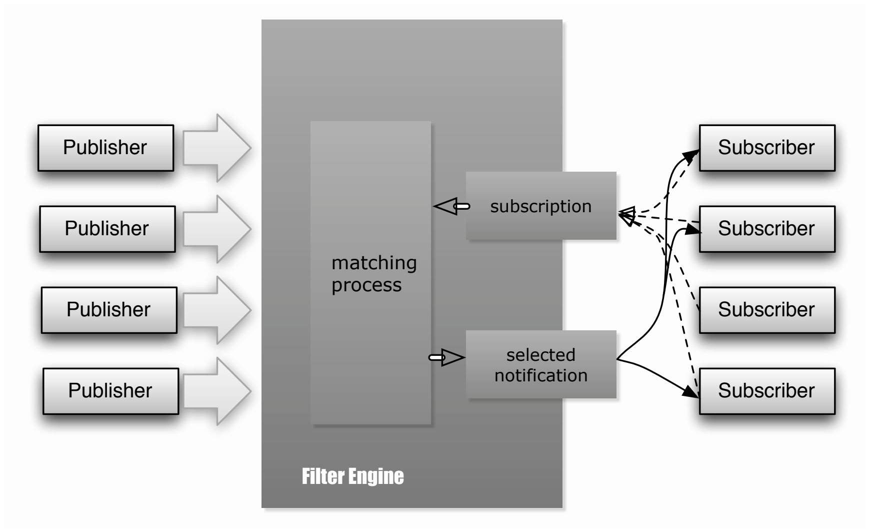


Figure 2.1: General Architecture of a Pub-Sub System

erations through standard interfaces (such as *publish*() *subscribe*() and *unsubscribe*()).

Normally, a pub-sub system consists of following components: 1) *events* and *notifications*, 2) *publishers* and *subscribers*, 3) *subscriptions*, 4) *event notification service* [41]. We briefly introduce each of these essential constituents.

Events and Notifications. An *Event* is defined as: any happening of interest that can be observed from within a computer [41]. In the scope of our research into an event base system, we consider any detectable arbitrary change in state as an *event*. A notification is defined as: a datum that represents an event [41]. A notification is created by the observer of the event and may simply indicate the plain occurrence of the event, but often may carry additional information describing the circumstances of the event. For convenience of illustration, we sometimes use the term *publication* in the remainder of this thesis to refer to the notification. In some cases the "(publication) messages," is also used as the name of the notification. As these terminologies are easy to distinguish by the context in which they appear, we will make no further explanation at their single appearance.

Producers and Consumers. Producers and consumers are the abstraction of real software components, which provide or utilize notifications.

Producers (often referred to as publishers) publish event notifications. The Producer's implementation is said to be "self-focused" in the sense that it observes only its own state [41]. The decision to publish a change in state is entirely dependent on internal business logic running on the producer.

Consumers (often referred to as subscribers) are the software component that reacts to notifications delivered to them by the notification service. In generic pub-sub architecture, a consumer can only submit its interests to the system, and react to the information that it has received.

Subscriptions. A subscription (also referred to as a *filter*) describes a set of notifications that a consumer is interested in. Consumers register their interest in receiving certain kinds of notifications by submitting subscriptions to the notification service. The service evaluates the subscriptions on behalf of the consumer and delivers those notifications that match one of the consumer's subscriptions. [41]. In the remainder of this thesis, the term *filter*, *subscription filter*, or *subscription pattern* may also be used to refer to *subscription*.

Event Notification Service. An *event notification service*, or *notification service* for short, is the mediator in event-based system that decouples producers from consumers. It is responsible for conveying notifications, and it must deliver every published notification to all consumers who have previously installed matching subscriptions. [41].

An *Event notification service* commonly provides a *publish-subscribe interface*, offering operations such as: *publish()*, *subscribe()*, *unsubscribe()*. The notification service gets a *publication* from a *publisher* via the *publication()* operation and a *subscription* from a *subscriber* via the *subscribe()* operation. On each node, upon receiving a *publication*, a *matching evaluation* will be carried out between the *publication* that it has just received and the *subscriptions* that it has stored. If there is a match, it will deliver the *publication* to the corresponding *subscriber* utilizing the underlying *event-routing mechanism.* The event notification service executes this *matching evaluation* on behalf of *subscriber* and delivers matching *publications* on behalf of the *publisher.* In this way, it decouples the *publisher* and *subscriber* during the event-dissemination process.

The implementation of an *Event Notification Service* can be either *centralized* or *distributed*, based on the underlying network environment and application scenario. In this research we only focused on distributed implementation.

The event notification service is the core component of a pub-sub system. Generally speaking, it can be decomposed into two functional layers: *event matching* (also called *event filtering*) and *event routing* (also referred to as *event dispatching*). *Event matching* is the process of matching an event to the subscriptions based on an *event-matching scheme*. An *event-matching scheme* is comprised of an *event filter model*, *a data model*, and a *matching evaluation algorithm* that is used in the event matching process. *Event Routing* handles the dispatching of the matched messages to corresponding interested peers. We will briefly introduce *event matching* and *event routing* in the following sections.

2.2 Event Matching

The event-matching process is dictated by the *event-matching scheme*. The event-matching scheme defines a set of rules and algorithms that are used for matching each publication against subscriptions. The event-matching scheme consists of the *event filter model*, the *data model*, and the *matching evaluation algorithm* used in the event-matching process.

2.2.1 Conventional Event-Matching Schemes

In conventional event-based systems, there are four kinds of event filter models: channels, subjects, types, and content. Based on the similarity of the filter models, the conventional event-matching schemes can be classified into two categories: *topic-based* (channel, subjects, types) and *content-based*. Topic-based event matching is simple and straightforward:

all of the information is grouped into several groups, based on predefined topics. Hence, in general, the topic-based matching algorithm is simple to implement, and efficient in systems that only cover small domains of interests, because, in this case, the system designer can easily define the term of the interests with a few topics. However, the necessity of providing a predefinition of the topic can seriously limit the flexibility and adaptability of the system. To tackle this problem, the *content-based* approach is proposed. The content-based approach makes it possible to choose filtering criteria along multiple dimensions without requiring a predefinition of topics [2]. The content-based matching algorithm evaluates a message by analyzing the content of the message. In this way, it reduces the management overhead for predefining and managing the "type" of information that is crucial to the topic-based approach. Interestingly, when analyzing contents based on a certain metric such as *name*, or *kind*, the *content-based matching* process will degenerate to the model of the topic-based approach. Thus, the content-based approach can be treated as a more general form of event matching. In implementation, the content-based approach generally requires that the message be represented with a unified data model, such as: a *key-value pair* (also called an *attribute-value pair* or *tuple* in some studies). Based on the data model, a standard evaluation algorithm can be applied to the content of the message to decide whether the message matches the subscriptions. The content-based matching scheme introduced extra difficulties, as the matching problem is more complex and there is no known scalable solution [2].

There is a large amount of research in the area of *event matching*, from earlier work focusing on the problem of matching in topic-based systems

[5, 44, 48, 39] to later work focusing on content-based matching problems [2, 18, 69, 31, 68]. As topic-based matching is straightforward and less flexible than content-based matching, we would rather not discuss the subject further here. Regarding content-based matching, there are two types of approaches: 1), the iterative approach. (e.g., Gryphon [28], Siena [9].); 2), the decision tree-based approach. (e.g., the work in [69, 68, 52].) The iterative approach simply consists of running an iterative matching algorithm that counts the number of matches between each properties of an event and each of the subscription predicates. Once a match is found, the algorithm will increase the counter by one. Thus, after evaluating all of the properties in the event, the algorithm can compare the number in the counter and the number of predicates denoted in each of the subscriptions. If they are equal, then there is a match between the event and the corresponding subscription. Normally this kind of method is straightforward, and easy to implement. However, the disadvantage of using an iterative algorithm is that it is not efficient if the process involves a large number of evaluations for each subscription. In addition, this approach only considers the conjunction relation between the predicates inside one subscription, while being incapable of handling the disjunction relation between the predicates. In a *decision tree-based algorithm*, the logical relation between predicates inside certain subscription is represented by a tree-structure, e.g., a *binary decision tree*. In the tree structure, each non-terminal node is a boolean function corresponding to the predicate inside one subscription, while each edge defines the matching result of the attached ancestor node (the one closer to the root). The logical relation between these predicates, such as conjunction and disjunction, can be represented by such a tree structure with a specific topology. The terminal node represents the final matching decision, normally either true or false. By going through certain path from the root to the terminal node, a matching evaluation result will be reported. Besides the efficiency in handling a large number predicates within subscriptions, another advantage of the decision tree-based method is that it is capable of handling both the conjunction and disjunction of the attributes. There have been a few attempts to enhance the original decision-tree based approach: In [2], the authors proposed a tree-based data structure to contain all constraints defined in all subscriptions. Based on this data structure, an optimization method was used to decrease the redundancies produced by the search trees. By doing a preprocessing beforehand, it can reduce the time complexity to sub-linear. But, as the author has mentioned, this approach is only efficient when the subscription is not likely to change frequently, in order to keep the overhead introduced by the preprocessing of subscriptions below a certain level. In [69, 68], the author proposed an extended version of the *Binary Decision Tree* (BDD), which was named "hierarchy colored OBDD graphs." By introducing a hierarchical structure and colored features to the original BDD graph, the normal predicate-based matching can then be extended to two levels of matching processes: *semantic matching* and *composite matching*. In work [53], an ontology-based matching process is proposed to eliminate the semantic heterogeneity that occurs in event matching. A commonly shared ontology is established beforehand and a structured representation *x-tree* [4] is adopted. Based on the commonly shared ontology, the event and subscription of a standard content-based system are converted into ones that are based on *concepts*. A matching method is also proposed to test the constraints on concepts by going through the *x-tree* representation.

2.2.2 Structure-based Matching

Currently in the literature, the *key-value pair* is the most popular data model for content-based matching schemes. For this reason, many studies have defined content-based matching schemes as those that have strictly adopted the *key-value pair* data model. With the growing popularity of XML, the importance of structured data representation is gradually becoming apparent. Attention is starting to shift from the conventional content-based data model (i.e., the key-value pair) to the XML-based data model. Even though technically speaking this new XML-based data model and the corresponding event-matching process still fall under the scope of the content-based matching scheme, many studies tend to treat it as a new category in order to distinguish it from the conventional key-value pair-based system. Hence, in this thesis, we follow this convention and treat XML-based event matching as the third category, which we have named *structure-based matching*. Unlike topic or content-based matching, structure-based matching focuses on the structure that is embedded inside the data. Compared to its predecessor (the topic-based and content-based model), the structure-based model can provide better expressiveness and flexibility by using structured information representation (commonly XML), and corresponding filter representation languages such as XQuery [67] and XPath [66].

Structure-based matching is also referred to in some studies as XML filtering. The goal of XML filtering is to realize the selective dissemina-

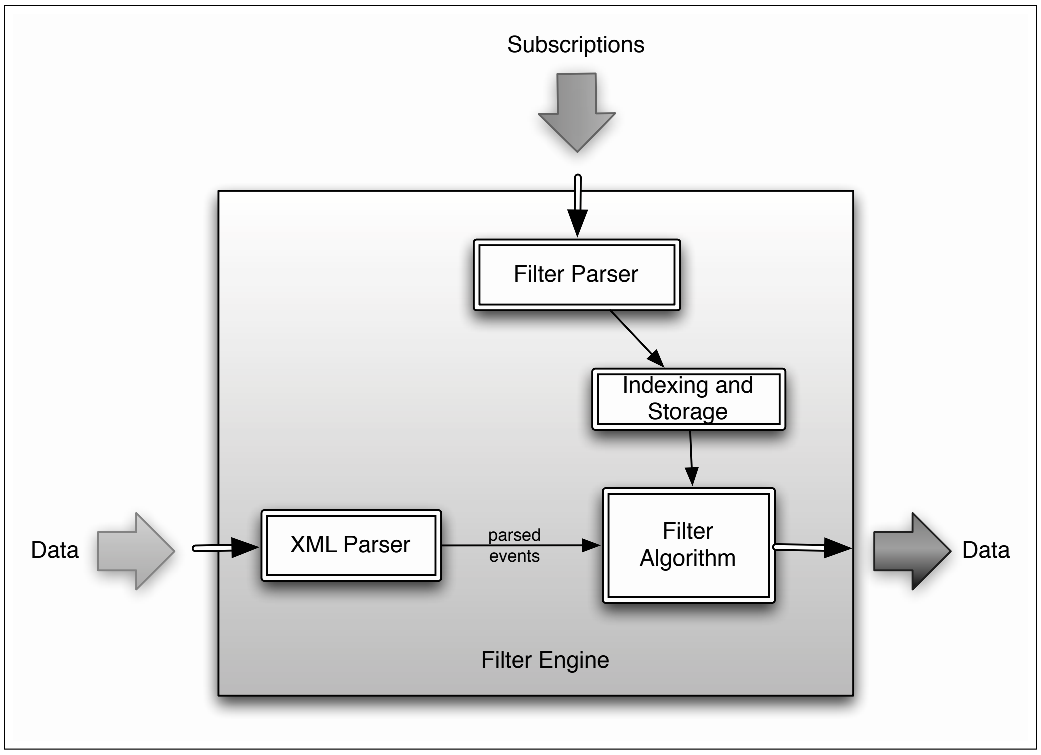


Figure 2.2: General Architecture of an XML Filtering System

tion of XML documents based on the XML-based filter languages, such as XQuery, XPath, and others. In spite of the difference in implementation, most of the current XML filtering systems adopted a similar architecture, as shown in Figure 2.2.

Refer to Figure 2.2, when incoming data (an XML document) arrives at the pub-sub system, it will first be parsed by an XML parser, which generates an abstracted structure that can be used by the corresponding matching algorithm (e.g., an event-based parser like SAX will generate a stream of events corresponding to element-tags, attributes, and values). The parsed events will be fed into the filter engine and matched against subscriptions. The subscriptions also need to be parsed first and then stored for the filter evaluation. If needed, an index will be created for subscription filters to accelerate the matching process. After the matching evaluation, a (logical) routing table, which includes a list of matched subscribers, will be created. In the end, the document will be forwarded according to this routing table.

2.3 Distributed Event Routing

The objective of *event routing* is to route the event notifications (publications) to the subscribers who are interested, based on the event matching results. To achieve this, we need to solve two problems:

1. How to ensure that the matching publication and the subscription congregate at least once on a common node?

2. How to deliver the publication to the corresponding subscriber once a match has been found?

In this section, we first provide a summary of a general approach to event-routing problems, namely *publication propagation* and *subscription propagation*, followed by an introduction to the event-distribution tree, which is the supporting infrastructure for subscription propagation-based approaches.

2.3.1 General Approach

There are mainly two ways to achieve event routing. First, the simplest is *publication propagation*, in which the event notification (publication) is simply forwarded to all existing consumers regardless of their subscription. This is essentially a flooding process. It can be used in small-scale network, but as the network scales up, it will become inefficient at some point, since the transmission cost of the flooding process will grow exponentially as the scale of the network grows. Thus, this method is not commonly adopted, so we will not discuss it further. The other is *subscription propagation*, which is commonly adopted in many studies on event-based systems. In this method, the subscription is first propagated (installed) to all or a portion of the brokers to establish a routing table at those brokers. A routing table consists of routing entries. Each routing entry is a *filter-destination pair*, where *filter* indicates the matching criterion, while *destination* indicates where the publication should be forwarded to once a match has been found. At each broker, the incoming matching publication will be forwarded to the destination based on the routing table.

To further improve the performance of the above-mentioned subscription propagation process, three optimization methods have been proposed, namely: identity-based optimization, covering-based optimization and merge-based optimization [41]. Identity-based optimization makes use of the similarities between subscriptions. Based on those similarities, subscriptions are grouped together to facilitate the event-dissemination process. Covering-based optimization takes the coverage relation among subscriptions into consideration and further prunes out the routing entries that have been already covered by the others. The merge-based approach finds the disjunction relation implicitly defined in the subscription filters and merges those filters into a new combined routing entry at each broker. These optimizations can, in varying degrees, reduce the amount of storage required for the routing table, and, at the same time, boost the routing look-up efficiency.

2.3.2 Event Dissemination Tree

Subscription propagation normally relies on an *event dissemination tree* (also, a pub-sub tree, PST). PST is a network overlay that connects all of the brokers in a pub-sub system into a tree structure. This is normally achieved in two ways: the *spanning-tree-based approach* and the *key-based routing approach*.

Spanning Tree-based Approach. Normally, the tree is constructed by identifying subscriber nodes that share common subscriptions and that are in close proximity to each other, while minimizing the messages transversal path to all other subscribed nodes. The computation requirement for forming the tree dictates that the subscription among the nodes does not change frequently and those nodes do not constantly enter or leave the subscription tree. As such, as the dynamism and the changes in the network topology increase, the system will suffer from a severe degradation in performance. Such an assumption is clearly invalidated in Internet applications where it is normal for potential nodes to join, leave, or fail and subscription updates happen frequently. This kind of approach utilizes the underlying overlay network to inherently acquire several good properties such as flexibility, robustness, and scalability.

Key-based Routing (KBR) Approach. A key-based routing structure is adopted to provide a routing and look-up facility for the routing of incoming publications. Both subscriptions and publications are mapped to a few *keys* in a key-space based on an *addressing* scheme. Based on their corresponding keys, the rendezvous point(s) can be found for potential matching publications and subscriptions. Compared to the spanning tree-based approach, the KBR approach has many advantages.

- Robust. Many overlay approaches have robust failsafe mechanisms to handle the failure of nodes.
- Incrementally deployable. The overlay approach scales well, even without a centralized consensus infrastructure. Nodes can join and leave willingly.
- Adaptable. The overlay approach can tolerate the dynamic changes that happen in the underlay.

In the next section, we will introduce this KBR approach in detail.

2.4 Key-based Routing Overlay

Key-based Routing (KBR) is a common abstraction of the routing mechanism widely adopted in structured P2P systems. A common implementation is the Distributed Hash Table (DHT). Thus, the terms KBR, DHT, and structured overlay are used interchangeably in many studies. In a standard KBR network, nodes are organized into a structured overlay, and each node is assigned a unique bit-array represented as a *key* from a large *key space*. Each node with its assigned key is responsible for handling a certain *key space range*. Likewise, the contents (also known as application-specific objects) are mapped to the same *key space* by means defined by the *addressing scheme*. For example, DHT uses the hashing results of the content objects as the *key*. Each node stores the contents (or the reference to the original contents such as the URL) that have keys which are within the *key space range* of the node. Hence, based on their hashed keys, the contents are distributed throughout the whole network. KBR provides a basic routing method so that, given an arbitrary key of an application-specific object, we are able to locate this object within a limited number of hops.

KBR has many applications. One popular application area is the peer-to-peer file sharing system; other fields such as content distribution and distributed caching also exploit KBR. In distributed file/content sharing, each file can be easily hashed into a key and assigned onto the node that is hosting it. When a user is required to retrieve a file, the process is reduced to a series of table look-ups of this key to locate the host node of the file and download the file accordingly. Normally in data-storage-oriented implementation, KBR provides a *distributed hash-table* function (DHT). A *key-value pair* is stored in the DHT. Given a *key*, a client can acquire the corresponding *value* by the simple look-up function offered by this mechanism. The distributed architecture of KBR makes it possible that a small part of the participants' change will only causes a minimal amount of disruption in the system. This guarantees good scalability and robustness.

A KBR system has four basic components: the *key-space*, *key-space partitioning*, *looking-up algorithm* and *underlying network*. These components are briefly introduced below.

Key-space This is a unified identifier space, which is used for the keys and identifiers of the nodes for a KBR system. It is the foundation of the whole system. A 160-bit key-space is adopted in common DHT implementations.

Key-space Partitioning The partitioning of the key-space is mainly for the purpose of storing the resources with their associated keys. Normally, the identifier of each node will act as the end point and separate the whole continuous space into several segments.

Looking-up algorithm This is normally the enhancement of the linear look-up algorithm for a basic ring structure. Techniques such as *fingering* and a multi-level routing table might be used to achieve better routing performances.

Underlying network This is the infrastructure network. Normally, nodes are allowed to join/leave/fail at will in the underlying network.

Research on KBR has been stimulated by the growing popularity of peer-to-peer file sharing systems. There are four prominent KBR implementations: Chord [57], CAN [50], Pastry [51], and Tapestry [77]. We will provide a brief overview of those implementations in the following sections.

2.4.1 Chord

Chord [57] is one widely used KBR solution. Like the most common KBR overlay, Chord's basic responsibility is to provide a key look-up mechanism. The key is represented with an *m-bit* array. The resource and the address (normally the IP address of the node) are mapped to this key-space. As a result, each node has an *m-bit* identifier. A node with the identifier *id* is responsible for the $key = id$ or the *key* that immediately precedes *id*. That is, the *node's id* is the *successor* of the *key*. The *id/key* space is wrapped by 2^m, i.e. the id after $2^{16} - 1$ is 0. The mapping from the address/source to the id/key is achieved by the function of consistent hashing. This consistent hashing guarantees that both *keys* and *ids* are uniformly distributed and in the same identifier space. The uniform distribution of *keys/ids* assured the robustness of the system in return.

Next, we briefly describe the looking-up algorithm in Chord: First, the nodes are arranged into a ring structure by their *ids*. Each node has a successor and a predecessor. There are normally "holes" in the ring, meaning the actual successor/ predecessor connected to the node i might not be the node $i + 1/i - 1$. Chord allows this case. To guarantee the robustness of the system, each node will not only keep one successor/predecessor, but also keep track of the k closest nodes, in case some of them leave/fail. This ring structure establishes the basic routing functions, i.e., the message will travel through the ring by the successor list stored at each node that is traversed. Based on this basic routing scheme, the fingering table is used to accelerate the routing/look-up process. The fingering table is a routing table that includes the IP addresses of several of the nodes in the ring other than the successor of the current node. The chord proposes the following rules to decide which nodes should be kept in node n's fingering table:

$$(n + 2^{k-1}) mod\ 2^m \ , 1 \leq k \leq m$$

With such a fingering table, the number of nodes that must be contacted to find a successor in an N-node network is *O(log N)* [57]

Figure 2.3 shows the basic look-up process when the request for resource 13 is generated from *node 1* and *node 2* respectively, the line shows the routing path from the request node to the destination nodes in the above process.

2.4.2 Pastry and Tapestry

Pastry is a ring-structured KBR overlay similar to Chord. The nodes are organized into a ring structure, and each node has a unique *id* in a *key-space*. The resources are also mapped to the same *key-space*, and thus can be correlated to each other as in Chord. However, there are some differences between Pastry and Chord. In Pastry, a multi-level routing table is proposed to accelerate the look-up procedure. The *key* is first separated into different groups (called digits), with each group having b bit, yielding a numerical system with base: 2^b. Later a multi-level routing table is established at every node, with level 0 representing a zero-digit common prefix, level 1 a one-digit common prefix, and so on. The routing table keeps the addresses of the closest known peers with the right digit at each level, except for the digits of its own at that particular level. Then,

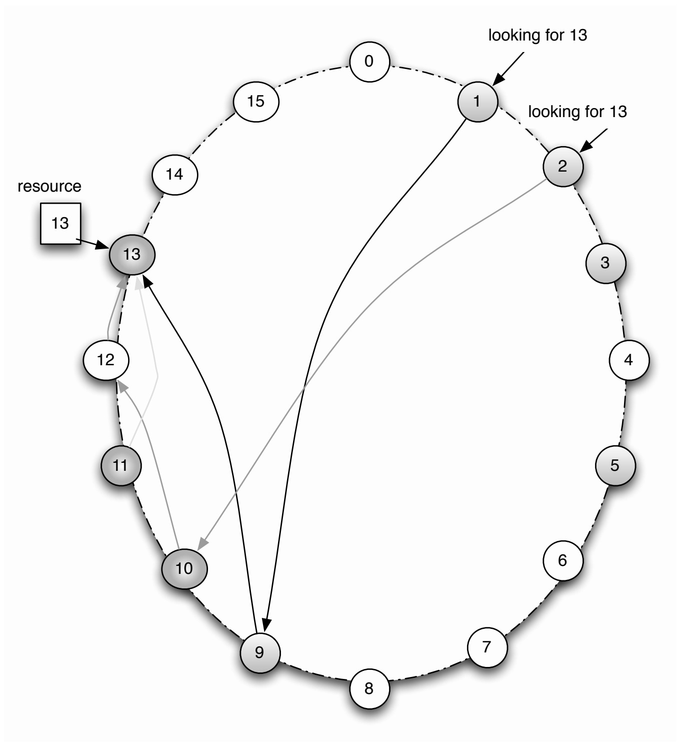


Figure 2.3: Chord Ring and Key Assignment

in each routing step a node will forward the message to a node whose *id* shares a prefix that is at least one digit (or b bits) longer than the prefix shared with the present node's *id*. If no such node is known, the message is forwarded to a node whose *id* shares a prefix with the *key* that is as long as that of the current node, and is numerically closer than the present node. During this procedure, the routing table is looked up one level deeper after each forwarding. Finally, the message will reach the leaf (the bottom level), which is precisely the node that is responsible for the requested *key*.

For a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $O(log_{2^b}N)$ steps under normal operations (b is a configuration parameter with a typical value 4).

As Tapestry uses identical strategy, we will not describe it any further.

2.4.3 CAN

CAN [50] is another KBR-based overlay network. It is formed on a multi-dimensional Cartesian coordinate space. In CAN, each node can have its own distinctive zone within this space. When a *key-value pair* needs to be stored, the *key* is deterministically mapped onto a point P in the total space using a uniform hash function. The pair will then be stored at the node that owns the zone that P lies in. When retrieving the pair, all we need to do is to find the point P. If P is in the zone owned by the retrieving nodes, then we can directly get the pair from the current node. If not, the request must be routed to other zones to find the node where P belongs. A coordinate-based routing mechanism is used to guarantee that the right node will eventually be found.

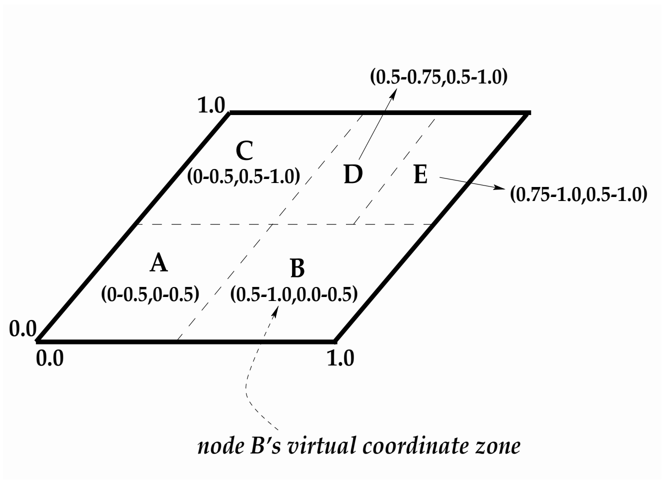


Figure 2.4: An Example of CAN Zone-division in a 2-d Space

The nodes can self-organize into an overlay network that represents the virtual coordinate space with the help of a *zone-splitting* and a *zone-merging* algorithm. For example, Figure 2.4 shows a demonstration of zone division in 2-d space. When node D joins the overlay network, it will be assigned a space by horizontally splitting E's zone. The routing is achieved by the coordinate-based routing table, utilizing the immediate neighbors of each node.

2.4.4 Other KBR networks

The above-mentioned KBR systems are the most fundamental ones. There have been many other studies in this area. Kademlia [35] is another famous KBR network that has several real applications. HIERAS [70] is a multi-layered KBR-based P2P routing algorithm, which is built on a multiring based structure. In [40] the overlay is built on a tree of rings. Cyclone [3] provides a way to cluster multiple different overlay networks together. The work in [20] proposed an architecture that can be used to hierarchically merge the existing overlay structures. It is implementation independent, and thus can be used on top of networks such as Chord and Kademlia. Since the design of KBR is not our major focus, we will not discuss them any further.

2.5 Summary

In this chapter, we introduced background knowledge on event-based systems.

The pub-sub is a communication paradigm for the selective dissemination of information based on the contents of the message. The strength of event-based architecture is that neither the subscriptions nor the published events are directed towards specific communication entities. The event-based style of communication carries the potential for autonomous and heterogeneous components to be easily integrated into complex systems that are easy to evolve and scale.

In general, a pub-sub system consists of the following components: 1) events and notifications, 2) publishers and subscribers, 3) subscriptions, 4) event notification service. Among those, the core is the event-notification service. An event-notification service can be decomposed into two functional layers: event matching and event routing. Event matching is the process of matching evaluation between publications and subscriptions. In current literature, there are topic-based, content-based, and structure-based event-matching schemes. Structure-based matching provides rich expressiveness and flexibility compare to its predecessors. Event routing mainly handles the routing involved in the dissemination of event notifications. The spanning tree-based approach and the KBR-based approach are two major ways of achieving event routing. The KBR-based approach has much potential, as it can bring scalability, robustness, and adaptability to the system.

Overall, this chapter lays the foundation and defines the terminologies and concepts upon which the later chapter is based.

Chapter 3

Efficient Event Matching

3.1 Introduction

The exploding volume of information on the Internet has made it more difficult to discover interesting information by human-initiated queries. With this as the trigger, a new generation of applications has been developed based on the philosophy of the selective dissemination of information. In these applications, heterogeneous data is selectively and automatically disseminated to a large number of interested users. This requires a new model of communication that deviates from the traditional request-reply model.

Publish-subscribe has emerged as one such promising paradigm. It provides loosely coupled and content-oriented communication through the selective dissemination of information based on user subscriptions. It has enabled a new concept of "pushing" the publication or alert messages to interested users who have shown their interest by subscribing. Nowadays, some famous applications that follow this concept have utterly changed our experience of using the Internet.

On another front, XML (Extensible Markup Language) has become the *de facto* standard for representing and exchanging data over the Internet. It has rich expressiveness, and provides standardized structures for data representation. As a result, a large amount of information is represented by XML, which calls for new XML-based data management and dissemination techniques.

Motivated by the popularity and expressiveness of XML, there have been attempts in recent studies [43, 34, 55, 54, 26, 16, 12] to encode the event message in a pub-sub system using XML-based messages or documents, in order to integrate and leverage the expressiveness of XML data with the effectiveness of the Pub-sub paradigm. Other than topics or attribute-value pairs in a conventional pub-sub system, the data can be published in a structurally rich and expressive format based on standardized XML, in which the subscriber's interests can be described in a more precise manner using semantically rich query languages such as XPath, among others.

Although many studies have been done on the XML-based pub-sub system, most of the studies suffer from excessive overheads incurred in matching time and the storage/memory space needed during the matching process. This makes them unsuitable in mobile computing environment, where computing resources are extremely scarce – In today's mobile computing platform, a large proportion of devices are compact hand-held devices which has quite limited storage capability and computational power due to the size and battery life constraint of those devices. Hence we address two issues:

1. The parsing and matching are tightly bound together, and cannot be separated. As the parsing takes a tremendous amount of time, the time efficiency is severely compromised.
2. The complex indexing and matching algorithms adopted by these works impose the risk of memory (or storage) space overuse.

In this chapter, we address the above issues, and propose a novel XML filtering method. Our method supports the separation of filtering and indexing, which frees the matching process from the time-consuming parsing job. The parsing can (but will not necessarily) happen at the client side once the new messages (or filters) are generated. In this way, the originally aggregated parsing job at the broker side will naturally be distributed to the client side. Along the way, we proposed a novel method to represent structure information using a bloom filter. This representation contains all of the essential information extracted from original XML documents by the parsing process, yet still keeps extremely compact. It has a fixed size and is independent of the complexity of the XML documents or user profiles, thus provides good scalability. Moreover, using this representation, the matching process can be simplified to a bit-vector comparison. The only trade-off is a false positive error; however, this can be easily controlled within an allowable rate. Our contribution is mainly as follows:

- *The concept of separating parsing from the matching process.* We provide a novel architecture to relieve the computation burden of parsing from the matching process. This will enable our framework to be adapted in a computation-constrained environment such as a mobile environment.

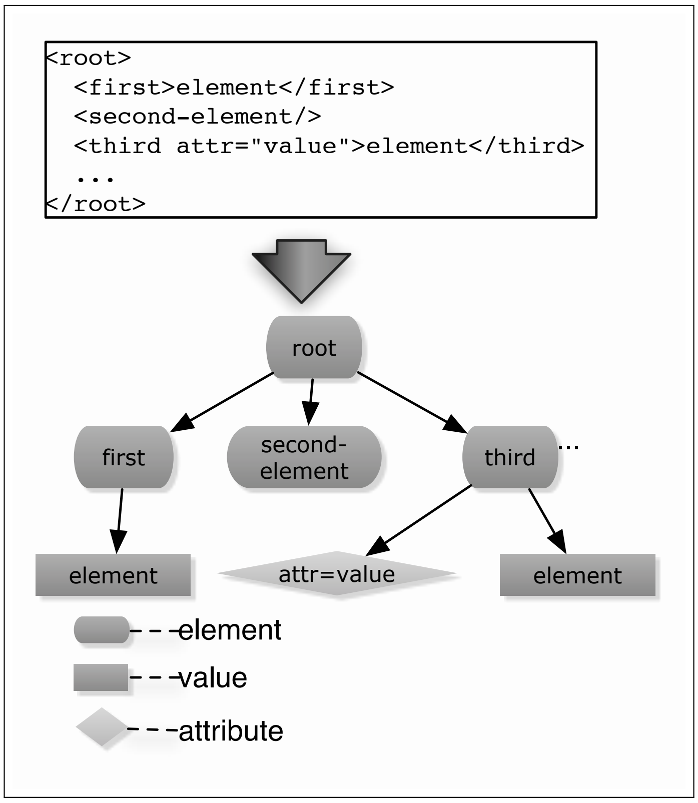
- *Efficient structure representation by a bloom filter.* We provide a way of representing information on the structure by using a bloom filter, which originally only supported the representation of non-structured data sets.
- *Efficient matching method based on the representation of a bloom filter.* Using our method, the evaluation can be as simple as a bit-vector comparison.

3.2 Bloom filter-based XML Filtering

In this section, we will present the main ideas of our bloom filter-based XML filtering system. First, we briefly describe the data and filter representation model in an XML filter system. After this, we provide an architectural overview of the proposed framework. In the rest of this section, we present proposed bloom filter based structure representation approach.

3.2.1 Data and Filter Model

Event notifications (publication) are represented in the form of XML documents or document streams. They will be fed directly into the pub-sub system. XML documents are normally modeled as ordered labeled trees (Figure 3.1) in which the nodes represent elements or values, and the edges



```
<root>
  <first>element</first>
  <second-element/>
  <third attr="value">element</third>
  ...
</root>
```

Figure 3.1: XML Document Tree

represent a relationship (the parent-child relationship) between two nodes. Values can only appear at the leaf nodes. Each element can have a list of (attribute, value) pairs associated with it. In this work, as predicate testing is not our concern, the attributes and values are treated in the same way as the elements.

Like most state-of-the-art XML filtering systems, we chose XPath as a filter representation language. The XPath language treats XML documents as a tree of nodes (which represents the Elements), and provides a way of retrieving parts of this tree. Normally, a query is an evaluation of an XPE,

```
Path           ::= AbsolutePath | RelativePath
AbsolutePath   ::= '/' RelativePath
RelativePath   ::= SimplePath | RelativePath '[' SimplePath ']'
SimplePath     ::= NodeTest | SimplePath '/' NodeTest
NodeTest       ::= TagName
```

Figure 3.2: Grammar of XPath Subset

which yields an object whose type can be a node-set, a Boolean, a number, or a string. However in the context of XML filtering, the returning results would be the whole documents if a match has been found. The XPE pattern can be naturally decomposed into a set of basic parent-child and ancestor-descendant relationships between pairs of nodes. As we stated on the issue of space/time efficiency, expressiveness is not our major concern. For ease of illustration, we limit our XPath expressions to a subset of the standard XPath. To be more specific, we only provide the un-nested twig pattern without a predicate evaluation. Neither the ancestor-descendant relation nor the wild card axis is supported. The grammar is defined in Figure 3.2.

3.2.2 Architecture Overview

The architecture of our proposed framework is shown in Figure 3.3. The filtering engine is depicted in the grey box. It is the core component of the system. The preprocessing module is comprised of a parser and a bloom filter creator. It will be used to process the raw XML documents and XPath filters before the filtering process. They can be embedded at the client side. In this way the parsing processes are distributed over the individual clients themselves, providing a naturally balanced distribution

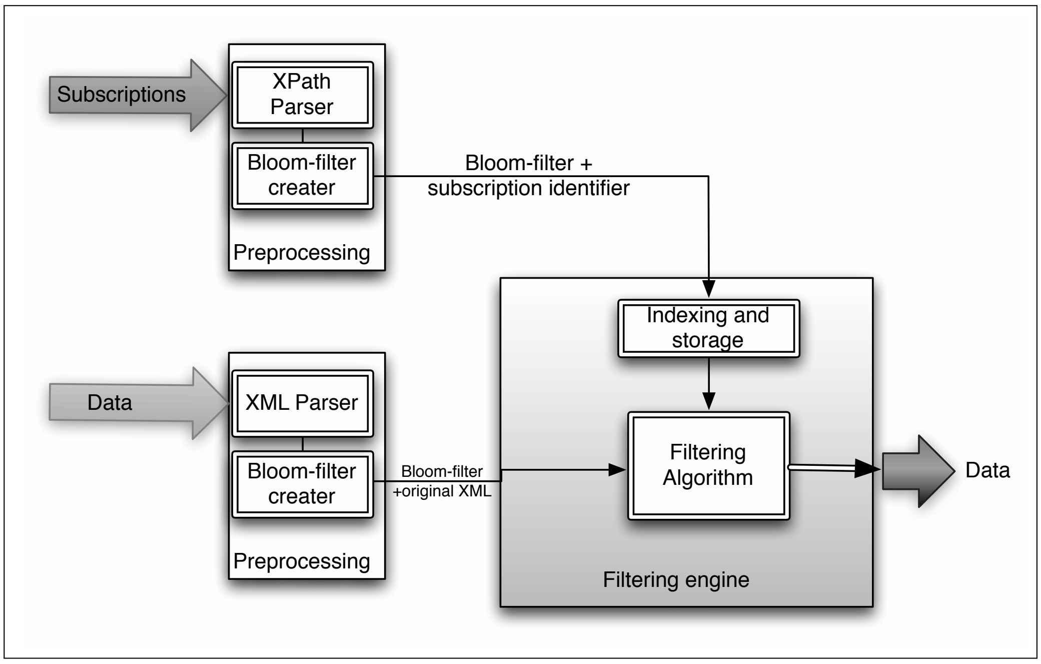


Figure 3.3: Architecture of Framework

of the computation throughout the system.

When a subscription/publication has been generated, it will be first processed by a preprocessing module, which is comprised of an XPath/XML parser and the bloom filter generator. A bloom filter is created after the processing for each subscription/XML document. The bloom filter combined with a subscriber identifier will be sent to the filtering engine, then indexed and stored for the filtering process. As to the publication, the XML documents will be sent together with the corresponding bloom filters. Once a document has arrived, its bloom filter will be used for the evaluation against the subscription bloom filters stored previously. The subscription process is a simple instruction to remove the record in the filter index, identical to that in a conventional pub-sub system.

The distinctive characteristic of our architecture is the separation of the parsing process from the filtering process. Existing approaches require that the parsed event stream be immediately used in the filter process. Thus, the parser is an essential part in the filtering engine. Consider that the time used by the parse will completely dominate the entire evaluation time [16]. This feature is especially important for enhancing the time efficiency of the filtering. Because we shift this part of the job to preprocessing, the preprocessing procedure takes care of the parsing and, in addition, handles a part of the processing job (i.e., bloom filter creation), thus moving a tremendous part of the computation out of the filtering engine. An intermediate representation of the original data (in the form of a bloom filter) will be generated, which can fully represent the parsing result. It is extremely compact in size and simple in evaluation. (It is essentially a bit-vector, so the evaluation process can be as simple as a bit-vector comparison, and thus can be carried out very quickly.)

3.2.3 Structure Model and Problem Formulation

Any structured data includes two parts: the data element and the relation between the data elements. Let E be the space of the element, and $\Re$ be the space of the total relationship. We define a data element as $e_i \in E$. The relation $R_{m,n} \in \Re$ denotes the relationship between e_m and e_n. The structure S of the data entity D with element $E = \{e_1, e_2, ...e_n\}$ can be defined as: $S = \{R_{i,k}, R_m | e_i, e_k, e_m \in E\}$. We call S the *structure* of this document, and R the *relationship pair*. For the sake of consistency, we do allow a singe element relation, like R_i and define it as the element $R_i = e_i$.

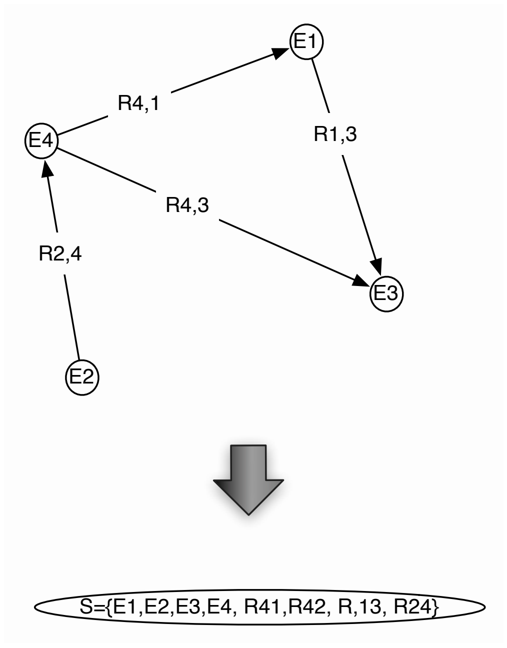
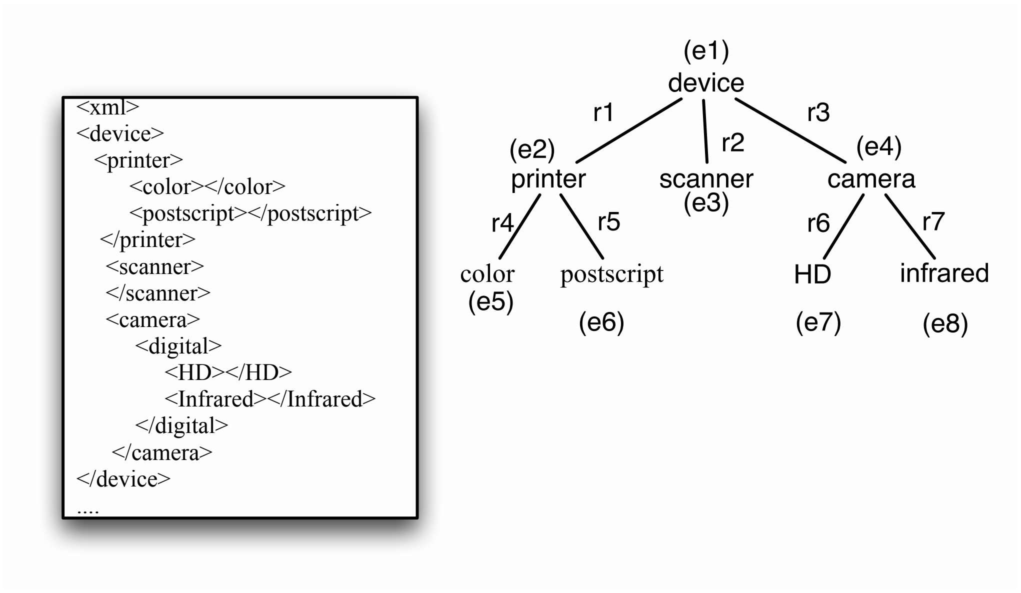


Figure 3.4: Graph Representation of Structure S

Any relationship involving more than two elements can be transferred into a set of relations on two elements. Thus, using relationship pairs would be sufficient to represent the whole structure. The data entity D can therefore be fully represented by its *structure* S.

The above-mentioned data representation can easily be depicted as a labeled directed graph $G(V, E)$, where the vertex V is the data element E, and the edge is the relationship pair $R_{i,k}$ (see Figure 3.4).

An XML document or an XPath expression can be formulated as a tree structure and easily fitted into the aforementioned structure representation. For simplicity, attributes are also treated in the same way as tag names, which we call nodes. All of these nodes will form the universe of elements. For XML documents, the relation represented by the edge between elements



```
<xml>
<device>
  <printer>
      <color></color>
      <postscript></postscript>
   </printer>
   <scanner>
   </scanner>
   <camera>
      <digital>
          <HD></HD>
          <Infrared></Infrared>
      </digital>
    </camera>
</device>
....
```

Figure 3.5: A Piece of XML File and the Representation of Its Structure

is the parent-child relation. This relation can be denoted by the order of two elements; thus, the relationship pair can actually be defined as an ordered pair, e.g. $R_{a,b} =< a \rightarrow b >$. Eventually, we can get the structure of the XML document: $S = \{R_{i,k}, R_m | e_i, e_j, e_k \in E\}$.

Figure 3.5 shows an example of an XML file being transformed into a structure: $S = \{e_1, ...e_8, r_1, ..., r_7\}$.

Based on our previous definition and model, we can now formulate an evaluation process in XML filtering as follows:

For a given target structure $ST = \{R_{i,j} | e_i, e_j \in E_{Target}\}$ and a matching pattern $SP = \{R_{i,j} | e_i, e_j \in E_{Pattern}\}$, we want to test whether $SP \in ST$.

We can find that using our model, the structure-matching problem can

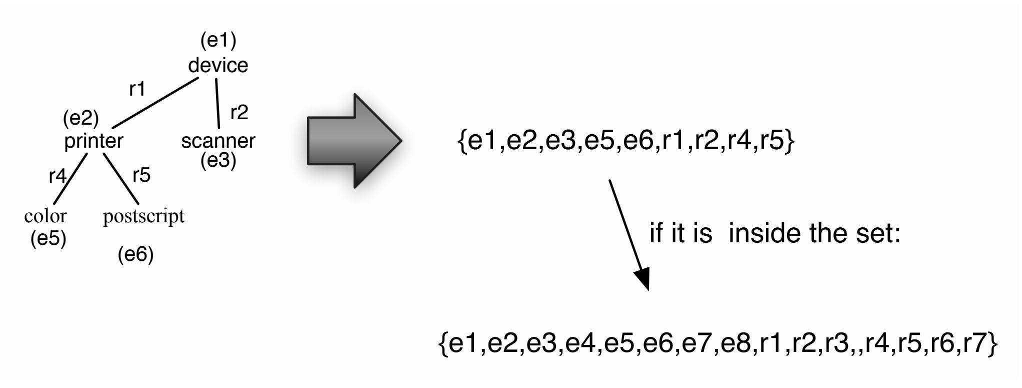


Figure 3.6: Complicated Pattern

be transformed into a partial matching problem (Figure 3.6 demonstrates this process with a given twig pattern).

3.2.4 Represent Structure Using a Bloom Filter

One important component in our framework is bloom filter-based structure representation. Based on the aforementioned novel structural model, the task at hand – the structure-based matching problem – can be naturally transformed into a much-studied partial matching problem. Hence, the novel data structure, such as a bloom filter, can be exploited. This section presents details about this process of representing information on the structure using a bloom filter.

The bloom filter was proposed by Burton H. Bloom in [7], and has been widely introduced in many studies. The aim of a bloom filter is to test a series of messages one-by-one for membership in a given set of messages. To achieve this, a hashing code is provided with a novel method. The new methods are intended to reduce the amount of space required to contain the hash-coded information from that associated with conventional methods [7]. The effect of a reduction in space is accomplished by a trade-off with an allowable error rate, more specifically, the false positive error rate. Normally, in a case where the space that is needed is more critical than the small risk of getting a false positive, a bloom filter is a good choice.

A bloom filter mainly provides two functions: One is that it provides us a way of representing a set, in other words, indexing. Another function is that it presents a way of partial matching.

Consider a set $A = \{a_1, a_2, ...a_i, ...a_n\}$ with n elements. First, we allocate an m bits array. Initially, in this array all of the bits are set to 0. We then select k independent hash functions $h_1(), h_2(), ..., h_k()$. Each of these hash functions is a consistent hash function that yields a result from 1 to m. Now for each element $a_i \in A$, the hash functions are applied individually, and we will get a series of hashing values $h_1(a_i), h_2(a_i)...h_k(a_i)$. Using these values as the address of the bit in the bit array, we can set the bit in question to 1 at the address $h_{1,...,k}(a_i)$ (see Figure 3.7).

We now start a query $B = \{b_1, b_2, ..., b_p\}$ that has p elements. We want to know if all of these elements in p are also in A. First, we apply the same set of hash functions $h_1(), ...h_k()$ onto each element $b_i \in B$. We then check the bit array at $h_{1,...,k}(b_i)$ to see if it is 1. If all of the values that are returned are 1, this means that there is a match between the query B and the target set A; otherwise, if any 0s are returned, the test will fail. That will mean that not every element in B is also in A.

We can see that the storage space needed for storing the bloom filter is only the space needed for the m-bit array. However, this characteristic is

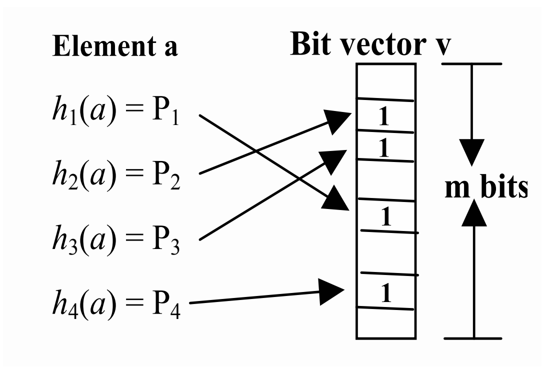


Figure 3.7: Bloom Filter

acquired with a trade-off of a possible false positive error. This means that, even if the test has passed, there is also a small chance that B may not be the subset of A. Even so, we can estimate the false positive rate, and adjust it under a certain allowable value. According to [42], this estimation can be calculated by: $P = (1 - e^{-kn/m})^k$. n denotes the number of elements [1]. A brief analysis of this estimation is provided in Appendix A. In addition, the hash function number k that minimizes P is found to be one of the two integer values closest to $(m/n)ln(2)$ [19].

Adding an element into the existing set is easy (when updating an existing bit position, the bitwise OR operation can be used). But removing an element from an existing set is impossible because the “1” bit may be set by multiple elements. However, deletion is not required in our system design.

A bloom filter has following properties:

[1]There exists some dispute on accuracy of this estimation, see [8]. More complex methods [8, 14], is proposed to acquire better results.

- The entire universe of the elements can be represented as a vector with all of the bits set to 1.
- The conjunction and disjunction operations between two sets can be easily implemented using a bitwise AND and OR operation
- The false positive error rate can be adjusted with a carefully selected k and m.

A bloom filter is designed to handle a set of elements that only have conjunction relations among them. Thus, structured information, which has a more complex relation among elements, cannot be directly represented by a bloom filter. Therefore, representing the structured information with a bloom filter is one of our major contributions. In the previous section, we have already shown a way of extracting the structure information from the structured data, and converting it into a set of relationship pairs with fine granularity. We can treat each relationship pair as an element in a bloom filter, so that the whole document, which is represented by a set of relationship pairs, can be represented by a bloom filter. Compared to existing research in [30], our method has finer granularity in terms of relation representation because a more detailed part of the structure has been extracted, carefully injected into the bloom filter, and is ready to be used for matching. Our method is simple in terms of implementation, yet it is precise enough to represent every detail of the relationship among elements of structured data.

A false positive could be a problem in this case because of the false positive results introduced by the bloom filter. However, we can easily control the rate of false positives by adjusting the length of the array as well as the number of hash functions, so that the overall performance will be optimized. After all, it is easy to deal with an affordable number of false positive errors at the subscriber's side.

3.3 Performance Study

In this part of the article, we first evaluate the time efficiency of our framework under various operational conditions. Then, we study the performance impact of the bloom filter length and the number of hash functions used in the creation of the bloom filter, denoted by k. We also evaluate the space efficiency by calculating the space conservation rate. Finally, we study the false positive rate introduced by the bloom filter under different parameters.

3.3.1 Setup of the Experiments

For our experiments, we acquire XML documents from the Niagara project [13]. The total number of XML documents in the sample is 3,680, conforming to nine distinctive DTDs (refer to Table 3.1 for more details). These XML documents were extracted from several real case applications across different domains. Therefore, the sample is representative enough for a common applications scenario. We generate XPath expressions using tools provided by the *yFilter* [16]. We implement our algorithm mainly in Matlab. For a comparison of performance with the *yFilter*, which was written in Java, we implement our matching procedure in Java and com-

Table 3.1: Characteristic of DTDs

DTDs	No. of Elements	No. of Attributes
SigmodRecord.dtd	11	1
actors.dtd	9	0
bib.dtd	13	1
club.dtd	13	0
department.dtd	18	0
movies.dtd	12	0
personal.dtd	15	1
profile.dtd	18	0
quote.dtd	12	0

pile the matching algorithm into Java class files. These Java class files are later imported into Matlab together with *yFilter*'s original Java implementations. In this case, Matlab mainly acts as the container for the testing environments, in order to control the system parameters and the collection and processing of the results.

The time cost of filtering was measured by averaging the filtering time of a given set of documents, while a document's filtering time was calculated by summing up the filtering cost of a given set of user filters. The average filtering time is the average amount of time that it takes to perform an evaluation between a single XML document and a single filter.

All of our experiments are performed on a 3.8GHz Core 2 Duo machine with 2G of memory, running a 32 bit-Linux OS. The Matlab version is R2009a in 32 bits. The Java version used to compile the Java code is 1.6.0_22. We used eclipse IDE to facilitate the construction and compilation of our Java code.

Three groups of experiments are conducted: time efficiency, space efficiency, and the false positive rate. In the following sections, we will discuss each of these experiments.

3.3.2 Time Efficiency

In this section, we analyze the time efficiency of our filtering method (we will call it the *bFilter* for ease of reference) in terms of matching time. We first calculate the matching time and compare it to *yFilter*. Then we evaluate the average filtering time of our algorithm exclusively.

Varying Numbers of Filters First, we compare the filtering time of our methods with the *yFilter* by varying the number of filters. The number of filters varied from 100 to 400. We randomly select 200 XML files from the total XML data set as the input documents. We choose 256 as the length of the bloom filter, and set the number of hash functions k to 1.[1] We calculate the matching time of both filtering methods. For a better understanding, we also record the parsing time used by the *yFilter*, and provide the time used for matching by subtracting the parsing time from the total time.

The results are shown in Figure 3.8. As can be seen, our method excels the *yFilter* by almost two orders of magnitude in terms of matching speed. However, after subtracting the parsing time used by the *yFilter*, the matching algorithm performs at almost the same level. This clearly indicates that the parsing time is the predominant factor in the *yFilter*'s total matching time. Also worthy of note is that the commonality between

[1] This parameter is decided by observing our test data, which generally include around 100 elements in total and an estimated false positive rate of under 0.2.

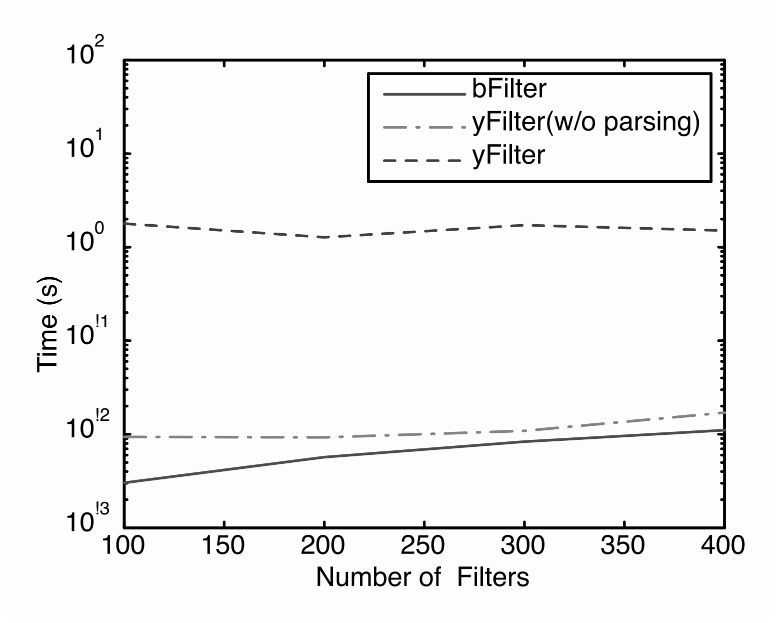


Figure 3.8: Comparison: Number of Filters

filters can be possibly used to further improve the performance of filtering system, which we will explore in a future work.

Varying the Filter Depth In this section, we compare the filtering time of our methods with that of the *yFilter* by varying the depth of the filter. First, we define the depth of an XPath expression as the maximum depth of the linear XPath that is decomposed from the XPath expression:

$$Depth(X) = \max_{i}^{N} depth(LP_i);$$

where LP is the linear path extracted from the XPath expression X by decomposing X, and N is the total number of LPs that can be extracted from X. We randomly choose 200 XML documents in the total XML

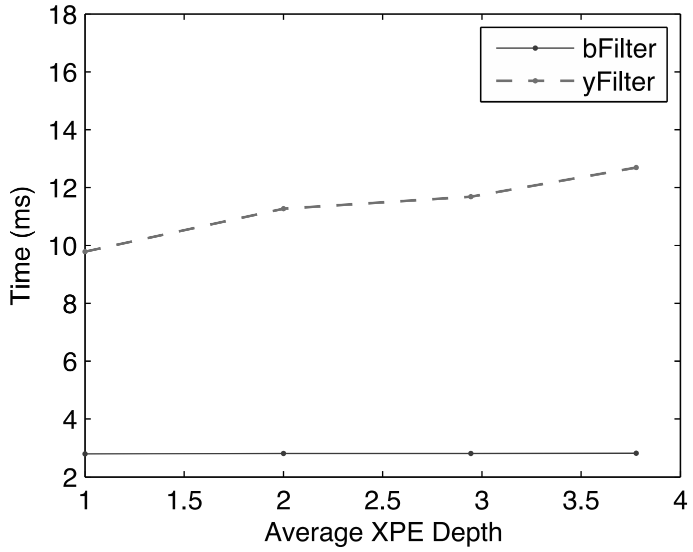


Figure 3.9: Comparison: Filter Depth

data set. The XPEs are generated into several sets, each with 1000 XPEs. For each set, we adjust the maximum length, yielding sets of XPEs with different average depths. We calculate the matching time for both the *yFilter* and *bFilter*, and evaluate the impact of the average XPE depths. The results are shown in Figure 3.9. As the result shows, *bFilter*'s filtering time stays almost constant as the filter depth grows, while the *yFilter*'s is affected by the depth of the filter and shows a linearly increasing trend. This is caused by increasing number of states in the NFA. Contrarily, the matching of *bFilter* is only a simple bit-vector comparison. Normally, it only depends on the vector length. Since we use a fixed length bit-vector, the matching time will not be affected by the complexity of the filter. This is a major advantage of our matching method.

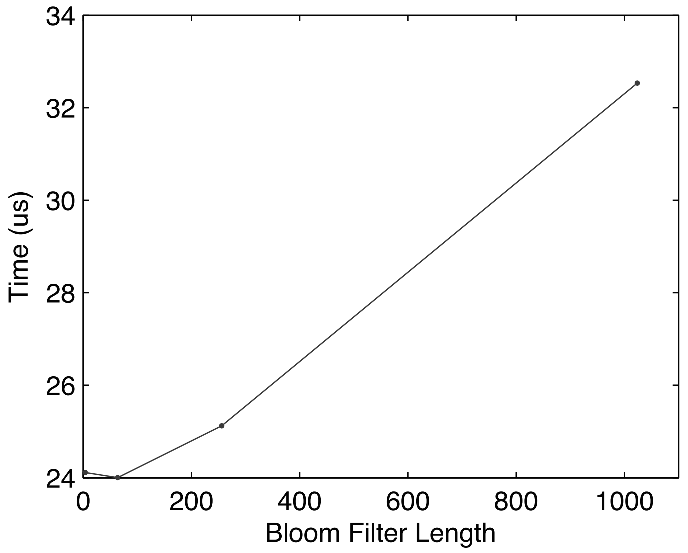


Figure 3.10: Average Matching Time: l

Average Matching Time In this section, we focus on the *bFilter* itself. We designed a set of experiments to study its behavior, especially the impact of the bloom filter length l and the hash number k.

- **Effect of bloom filter length** In this test, we randomly choose 100 XML documents from the sample set and randomly generate 10 XPaths from each DTD as the filter. This will provide us with a total of 9000 single matches. We then calculate the individual matching times, and acquire the average matching time for each matching process. By varying the length of the bloom filter from 4 to 1024, we repeat the above experiment, and collect the statistical results on the impact of the bloom-filter on the average matching time in Figure 3.10.

  We observed a trend of an increase in matching time as the length

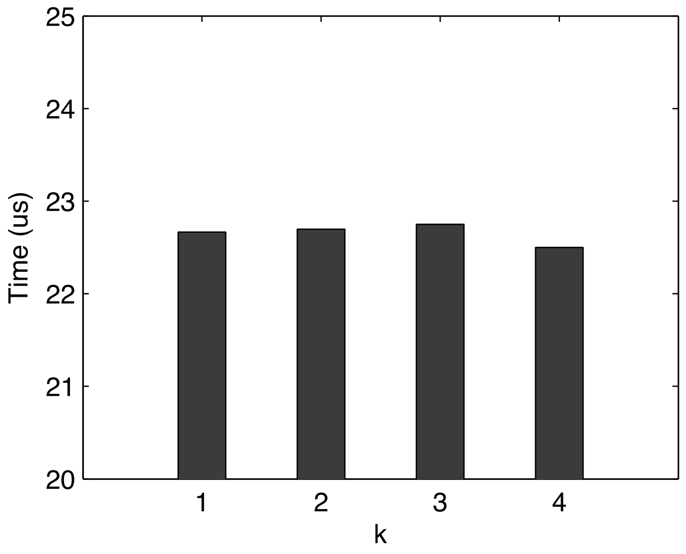


Figure 3.11: Average Matching Time: k

of the bloom filter increased. This is simply because the number of comparison processes between the bits increases as the total length grows. However, even at the length of 1024, the average matching speed is only 32μs, compared to the 24μs at the length of 4. This increase in time is reasonable and acceptable.

- **Effect of the hash number** Identical to the above experiment, we acquired the statistical result of the impact of the hash number on the average matching time (see Figure 3.11).

  The results show that the hash number had little impact on the matching time. This is because the evaluation process is a bit-vector comparison; thus, as long as the length of the bit-vector stays unchanged, the matching time will not be affected.

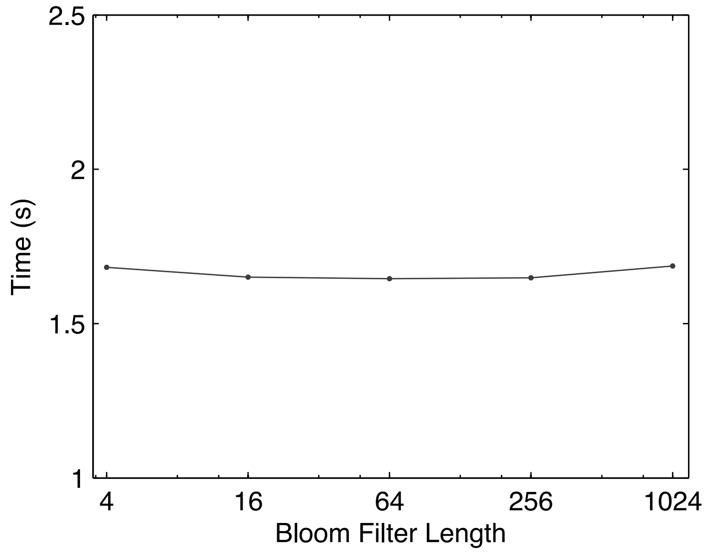


Figure 3.12: Preprocessing: l

Preprocessing Time In this section we vary different parameters to study the their impact on the preprocessing time. The preprocessing time is calculated by averaging the processing time on 1000 randomly selected XML files. We mainly evaluate the XML files in this section, as the XPath is relatively small and simple to parse.

- **Effect of bloom filter length** The length of the bloom filter is adjusted from 4 to 1024. The results are shown in Figure 3.12. We can see that the length is not a key contributor to the preprocessing time cost. This is because the hashing algorithm we adopted generates hash arrays of a constant length, which may be reduced to the needed length. Thus, no matter how long the bloom filter is, the computation needed is the same.

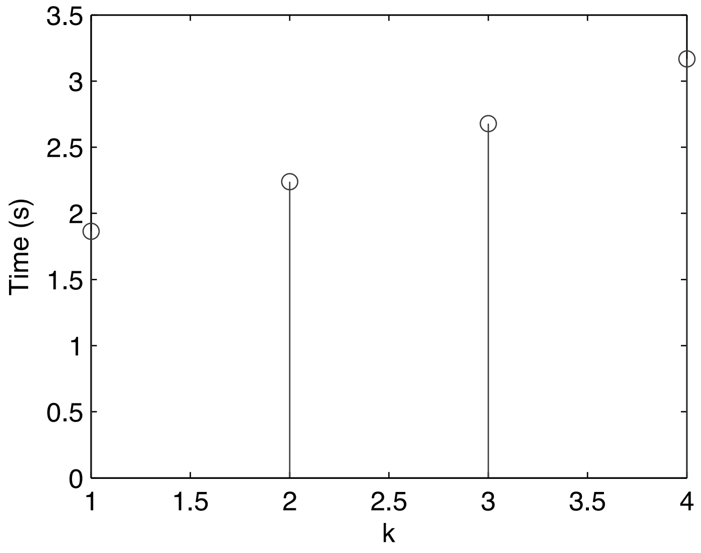


Figure 3.13: Preprocessing: k

- **Effect of hash number** In this experiment, we adjust the hash function number k. The result is shown in Figure 3.13. It can be observed that as the hash number increases, the time needed for preprocessing increases linearly.

3.3.3 Space Efficiency

For XML documents, we calculate the total space needed to represent all of our sample XML documents. As to user profile, a new metric – *space conservation rate* – is used for evaluating space conservation by using bloom filter representation instead of XPath expressions.

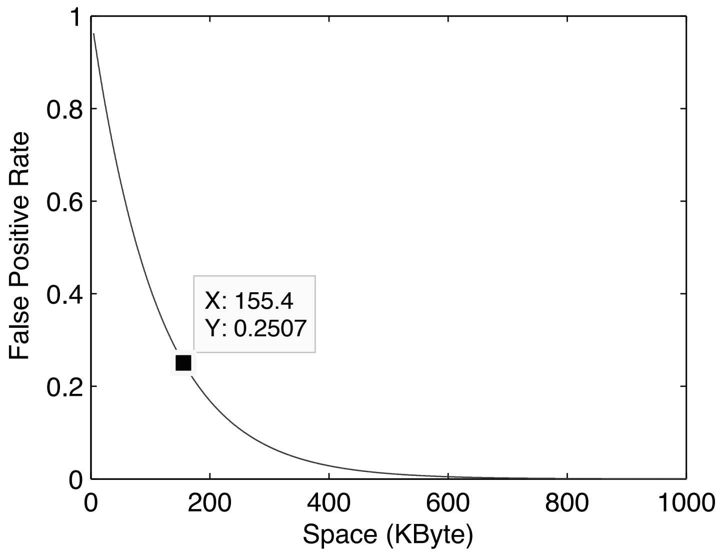


Figure 3.14: Storage Space

Space for Representing All XML Documents The total space needed to store our testing data sets is around 3GBytes. Figure 3.14 shows the space needed for representing all of the data sets under a different false positive rate estimation p. The false positive rate estimation is derived at the optimized status, where k is selected automatically to guarantee an optimized result. According to the results, when p is 0.25, the bloom filters only take up around 150KBytes of space. This is only about 5×10^{-5} times the amount of space needed for XML documents.

Space Conservation Rate We define the space conservation rate as follows:

$$R = (Space_{XPEs} - Space_{bloomfilter})/Space_{XPEs}$$

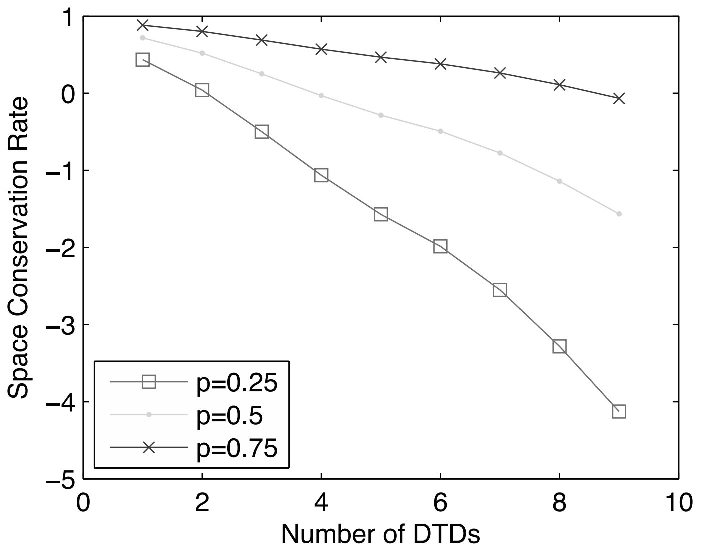


Figure 3.15: Space Conservation Rate

We randomly generate XPEs with the given DTDs. The filter length is not limited, thus resulting in a filter length of between 1 to the maximum length for each DTD supported. We vary the number of DTDs involved. The length of the bloom filter is then estimated by the total number of elements involved in DTDs, guaranteeing a false positive rate of 0.25, 0.5, and 0.75, respectively. We present the results in Figure 3.15.

We can observe from the figure that as the DTDs involved increase, the space conservation rate shows a decreasing trend under all of the false positive rates. Also, smaller false positive rates lead to a faster decline in the space conservation rate.

We can conclude that, when the scale of the system is moderate and the allowable false positive rate is not so stringent, using a bloom filter-based representation can largely conserve the space needed for storing user filters.

In addition, we should be aware that for the XPEs we generated by our sample DTDs, the average length for a single XPE is around 64Bytes with UTF-8 character encoding. In the case of a longer tag name and deeper XPE level, more space will be needed for XPEs, and the conservation rate will increase even further.

3.3.4 False Positive Rate

According to [19], given a fixed bloom filter length m and the number of elements n, a minimum false positive rate p can be achieved when the number of hash functions $k = (m/n)ln(2)$. Thus, the estimation of p can be calculated given m and n.

In the following sections, we conduct a series of experiments to study the impact of several important parameters of the system on its performance, including the bloom filter length and the hash function number k in terms of their impact on the false positive error rate. We also evaluate impact of the number of filters and the filter depth, which are important factors in real case situations, for they implicitly decide the number of elements n. We adjust each parameter under observation and repeat the test. For each test, we use the same 100 XML files that have been randomly selected from the sample set as the input documents, and the same set of filters that are generated from nine distinct DTDs. The number of the filters is 900. For each test, we calculate the average false positive rate from the filtering results. The total number of relationship pair is 93, which is obtained by analyzing the XML documents and filters. In the filter number and depth evaluation, the length of the bloom filter is 256 and k=1.

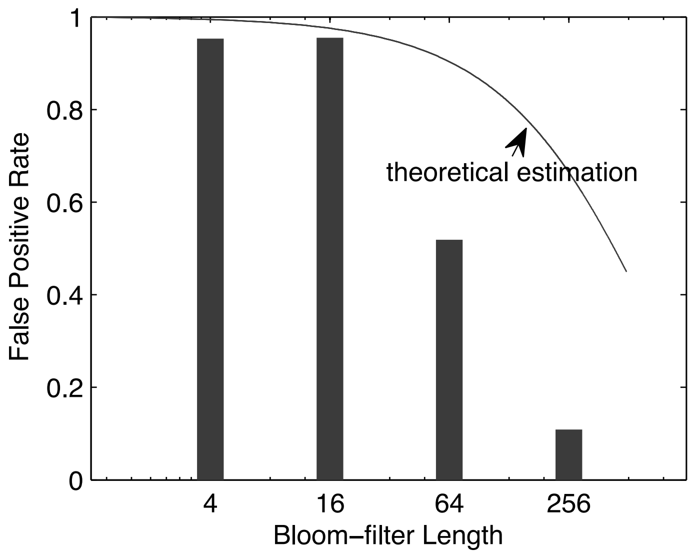


Figure 3.16: False Positive Rate: l

Varying the Bloom Filter Length The bloom filter length is a major parameter to control the false positive rate. Thus in this section we conduct the experiments to study the impact of bloom filter length. We vary the bloom filter length from 4 to 256. The hash function number k is set to 1. Refer to Figure 3.16 for the results.

We can see that the actual false positive rate is well bounded within the estimation.

Varying k In this section, we study the impact of the parameter of hash function number k. The experiment is identical to the evaluation of the length of the bloom filter, except that we adjust the hash function k from 1 to 4. The result is shown in Figure 3.17.

It is apparent that when k is no more than 2, we have the best results;

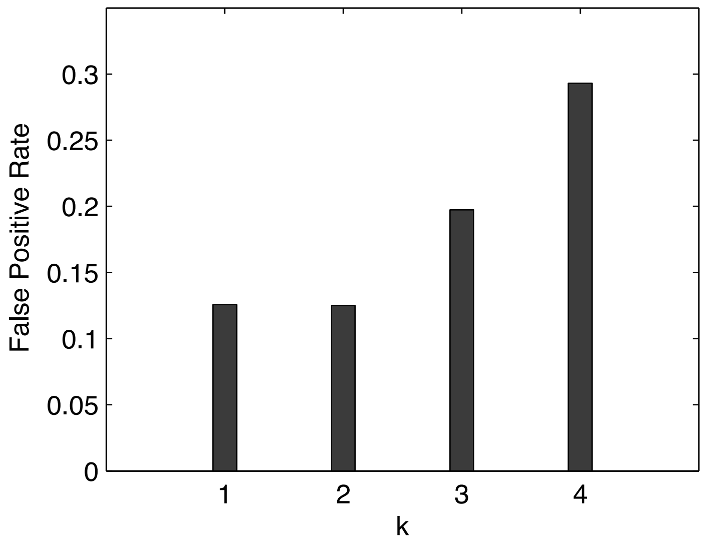


Figure 3.17: False Positive Rate: k

after that, the false positive error will increase. Indeed, this also coincides with the theoretical estimation.

Varying Numbers of Filters In this experiment, we will evaluate the impact of the filter number on the false positive error rate. We vary the number of filters from 100 to 900. The results are shown in Figure 3.18.

According to the result, the number of filters has little effect on the false positive error rate. This is mainly because the number of filter is almost independent of the total number of relationship pairs. This shows that our method exhibits good stability and scalability.

Varying Filter Depths Here, we evaluate the impact of the depth of the filter on the false positive error rate. The filter depth is adjusted by setting

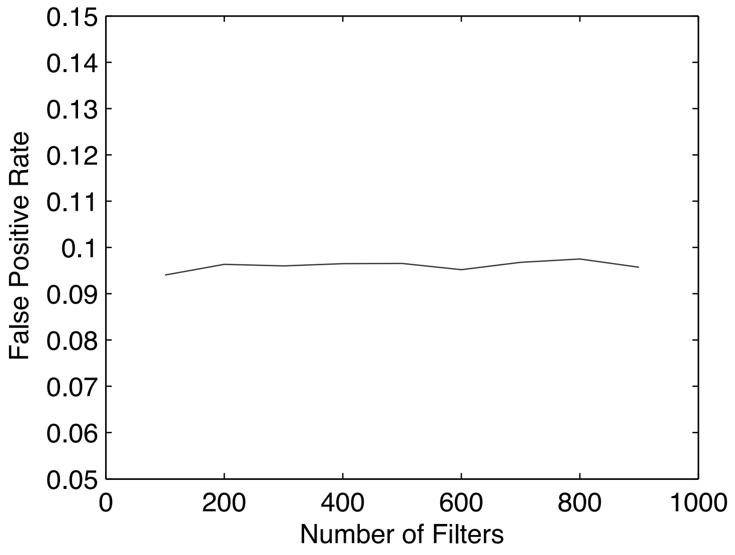


Figure 3.18: False Positive Rate: Number of Filter

the maximum filter depth parameter of the XPE generator. The average filter depth is calculated under each test to provide more meaningful results. Refer to Figure 3.19.

We can see that the false positive error rate increases as the depth of the filter grows. This is because the growth in depth caused the increase in the relationship pair number n, which will cause an increase in the false positive rate. However, the trend of increase becomes less obvious as it grows. We noticed that it never exceeds 0.1. This coincides with the theoretical estimation of the false positive rate.

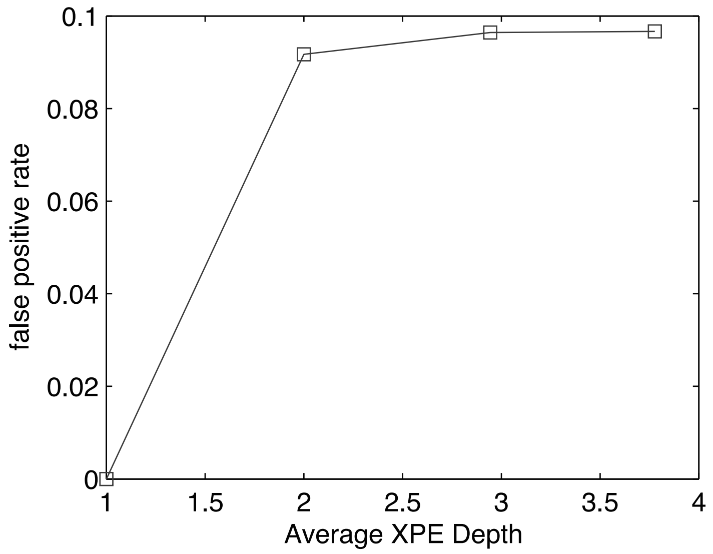


Figure 3.19: False Positive Rate: Filter Depth

3.4 Related Work

In the current literature, there are two main groups of research on the XML filtering algorithm: the *Automaton-based* method (also called the *Navigation-based* method) and the *Indexing-based* method.

Automaton-based method A large proportion of existing studies is based on the automaton, including: *Non-deterministic Finite-state Automata* (NFA) [16, 54, 25, 36], and *Deterministic Finite-state Automata* (DFA) [22, 23, 45]. With this method, an element in the twig pattern can be represented by a transition of states in Finite-state Automata (FSA). Hence, the parsing results of the incoming document can be used to drive the state transition. Once the final state (also known as the *accepting state*) is reached, a match will be reported. One major disadvantage of the automaton-based method is that it requires parsing to drive the change in state; thus, it is not easy to separate the parsing from the filtering engine. Moreover, the number of states is dependent on the total number of elements in XPath filters. As more distinctive filters become involved in the system, the space/memory needed for the increasing number of states will affect the space efficiency with which we are concerned.

Indexing-based method Many studies have focused on building an efficient indexing method for the incoming user profiles. In [12], an index structure called *XTrie* is proposed to support the efficient filtering of XML documents based on XPath expressions. By indexing sequences of elements organized in a Trie structure, the filtering engine is able to share the processing of the common part of the substring among filters to reduce the number of unnecessary indexes and redundant matches. However, the indexing structure needs to store all of the elements in every XPath filter in a string format without any compression. Space consumption will become an issue as the number of distinctive filters increases. In the PRIX system [49], the user profiles and incoming documents are transformed into a prüfer sequence, and the matching can be taken as a test of two sets of sequences. Similarly in [34, 33, 32] prüfer sequences are also used to express XPath filters. As a prüfer sequence requires a copy of names of the original elements, the same space-consumption issue exists as in XTrie. The work in [58] resorts to a relational database to efficiently evaluate a large number of subscriptions in a long-running system. Although using a database can shift the usage of memory to the external storage, in most mobile devices the external storage is as limited as the main memory. Therefore, this shift is no help at all. In fact, the establishment and maintenance overhead for a relational database will affect the performance of the filter to some extent. In sum, as the above approaches concentrate on providing a feature-rich indexing structure, they inevitably increase the space needed for the indexing structures to fully operate and maintain the system. Moreover, when the filters are updated frequently the maintenance overhead grows greatly.

To the best of our knowledge, there is no research work identical to ours in the literature. But we still list the two closest research works here for reference. In [27], each XPE is translated into an ordered set of predicates, where each predicate is an (attribute, operator, value)-triple, and the relation between two adjacent tags is encoded in the predicate. The predicate-based evaluation can be easily adopted. This is similar to our structure-extraction approach. But one major difference is that, in our approach, after we extract the structure out of the XML documents, we proceed with an extra procedure; i.e., to represent the structure in a more compact way (by a bloom filter), which is also optimized for evaluation. In [21], the bloom filter is also used as an efficient data structure representing path queries. But the way that they handle structure information is quite different. Briefly, each incoming document is parsed and turned into a candidate path list that contains all of the possible subpaths. Each path is treated as an element in a bloom filter. Thus, the bloom filter of the candidate path can later be compared to the user profile to test whether there is a match. Since retrieving all of the possible subpaths of an XML document tree is a time-consuming job when the tree structure is complex, it cannot be well scaled. Moreover, only linear paths are supported; the lack of support for the twig pattern would compromise the effectiveness of the filtering system as a whole.

3.5 Summary

To essentially enhance the space/time efficiency of the filtering system, the amount of time that is used for parsing needs to be reduced. Moreover, the matching algorithm should be as simple as possible to reduce the complexity of the computation and memory usage, which is critically important for mobile devices with resources constraints. This has driven us to propose a framework that separates parsing from the matching process, in which the former incurs significantly high overheads. In addition, we adopted a highly efficient method of representing structure, which is highly compact and optimized for evaluation using a bloom filter. Judging from the results of several evaluations carried out by extensive experiments, our method shows excellent matching speed and compact storage size. It also shows good stability and scalability because it is not affected by an increasing number of documents/filters once the system parameter is fixed. Even though it introduced a false positive error, the false positive error can be bounded and adjusted to match the application's requirements. Considering the significant speed-up and performance gain, our approach represents a viable solution when time/space efficiency is paramount.

Chapter 4

Scalable Event Routing

4.1 Introduction

XML-based structural data has been widely adopted in recent data-oriented applications to provide semantically friendly representations of data entities. The explosion in the volume of information and drastic increase in the number of users involved in these applications have brought radical challenges to data management and dissemination. Traditional request-reply style searching techniques no longer suffice. Instead, pub-sub-based selective dissemination has emerged as a better solution, providing loosely coupled and content-oriented communication among parties. This development has completely changed the user's experience of data-centric applications. Therefore, the natural move to integrate and synergize the expressiveness of XML data with the effectiveness of the pub-sub paradigm elevates studies on providing scalability and flexibility in content dissemination over a ubiquitous and heterogeneous environment.

To date, attempts to disseminate XML through pub-sub have merely focused on adapting an XML-based matching scheme to existing pub-sub infrastructures, without considering the actual process of event routing and delivering. However, event routing and delivering, which is an essential component in a pub-sub system, is tightly bound to the event-matching scheme. Thus, to develop a practical yet efficient pub-sub system we should take both processes into consideration. Furthermore, conventional event routing was not initially designed for a large set of distributed clients. What is required is an utterly new design for the event routing process in a large-scale distributed environment.

P2P-based overlay technology, specifically the key-based routing overlay (KBR), supports large-scale exchanges of data between distributed clients. It is considered a standard protocol for distributed file sharing and media streaming. Thus, we decided to adopt the techniques of KBR overlay to our distributed XML dissemination system. Using such technology, we were able to design a robust, adaptable, yet easy-to-maintain application layer network routing scheme. More importantly, we managed to converge the event-matching scheme with the addressing scheme of the overlay network and facilitate the event routing process, which exactly suits our design purpose.

In our research, we found two essential elements that need to be consider when designing a pub-sub system that utilizes the KBR overlay:

1. **Addressing Scheme**. We needed to find an appropriate addressing scheme where each node can be easily reached by a unique address

that is assigned. This addressing scheme should be one that can be implemented in a distributed manner and also easily maintained. More importantly, the process of addressing should enable certain structural properties of the structured data to be embedded. In other words, an address should be able to reflect the structural information of certain XML data. We call this property *structure-awareness*.

2. **Event Dissemination Topology**. Based on the addressing scheme, an efficient overlay topology should be provided to facilitate the process of event dissemination. To be more specific, using the structural information embedded in the addressing scheme, we need to find a way to hierarchically organize nodes based on the structural similarity of their subscription.

Baring these points in mind, we took a holistic design concept and developed a bloom filter-based matching scheme that coalesced into the addressing scheme. On the basis of this, we exploited the existing KBR overlay design and came up with a hypercube-based overlay architecture, which is optimized for efficient event dissemination under our addressing scheme. Our contribution can be summarized as follow:

A scalable and flexible distributed XML dissemination system. We took both event matching and event routing into consideration, and carefully developed a distributed pub-sub system suitable for the large-scale dissemination of XML data.

An addressing scheme that is optimized for the event-matching scheme. We utilized the natural convergence between the event-matching scheme and the addressing scheme, thereby optimizing the event-routing process.

The address can reveal certain structural information that can later facilitate the event-routing process.

A robust, adaptable, yet easy-to-maintain application layer network routing scheme. We developed a hypercube overlay by exploiting the structure-awareness of our addressing scheme. With this overlay, a multicast style of communication can be established for efficient message propagation.

An extensible multilayer architecture. With a multilayer design, it is easy to make modifications to support different levels of optimization and adaptation corresponding to specific layers.

A holistic approach towards an efficient pub-sub architecture design. As far as we know, this is the first work to take a holistic approach in considering filtering and routing as two processes that may synergistically complement one another to support efficient message dissemination. In our approach, filtering is seamlessly coalesced into the routing scheme and can be accomplished seamlessly and efficiently as publications are propagated across the nodes.

4.2 System Model

In a distributed XML-based pub-sub system, nodes will selectively participate as publisher or subscriber or both. Since there is no dedicated broker to forward messages, nodes may also participate in message forwarding. We use P_{all} to denote the total set of publishers, and S_{all} to indicate the total set of subscribers. Publishers generate publication messages in the form of *XML documents*, denoted by $p_i | i \in P_{all}$ Subscribers submit their interests in the form of *XPath Expressions* (XPE) for short, denoted by $s_j | j \in S_{all}$ A matching evaluation $eval(s_j, p_i)$ is a boolean evaluation function that yields true if p_i matches s_j, and false otherwise. If a publication matches a certain set of subscriptions, the published message will be delivered to the subscribers of those matching subscriptions. For a given publication p_i, all of the matching subscribers form a set, named a *forwarding set*, denoted by $FS(p_i)$. Based on the above definition, we have:

$$FS(p_i) = \{j | eval(s_j, p_i) = True; j \in S_{all}\}$$

Hence, given a publication p_i, the major task of the pub-sub system is to acquire the right forwarding set $FS(p_i)$ and deliver information to each member of the forwarding set. To acquire $FS(p_i)$, we need to first have a proper evaluation function: $eval(s_j, p_i)$, then apply $eval(s_j, p_i)$ to every subscription candidate. In a distributed environment, this also implies the following: In order to identify all of the members in the forwarding set, we need to make sure that the matching publication and subscription congregate at least once on a common node, which we term *a rendezvous point.* This is necessary so that the evaluation can be carried out to perform the match between the publications and subscriptions. In this connection, we identified two basic problems that need to be addressed:

1. How can an efficient and effective matching evaluation be carried out for XML-based structure matching (which we term *a matching problem* in the remaining text)?

2. How can we effectively and (possibly) optimally align publications and subscriptions to congregate at the right rendezvous point(s) while op-

erating in a distributed environment (which we term *a routing problem* in the remaining text)?

In addition, the issue of dissemination efficiency should be addressed. This is particularly true for handling one-to-many message propagations in a densely populated network of nodes. The first problem can be efficiently solved by our event-matching scheme discussed in Chapter 3.2. In the following sections, we address the routing problem, and present our solutions.

4.3 Key-based Routing for Pub-sub

In this section, we address the routing problem, specifically, the challenge of ensuring that the publication and the subscription converge to at least one rendezvous node. In conventional pub-sub systems, this is done in two ways based on whether the publication or subscription is being propagated. The former, which is essentially a flooding of publications, is named *publication propagation*. The latter is named *subscription propagation*. Here, subscriptions, in the form of filtering tables, are propagated and placed across the entire network of participating nodes. Publication and subscription propagations are both inefficient when applied to large-scale distributed systems due to their flooding nature. Generally, publications are generated more frequently than subscriptions and publication messages are significantly larger in size than subscriptions. For this reason, most established systems use subscription propagation, which incurs less traffic and lower processing overheads on the nodes. To further facilitate efficient propagation, a spanning tree overlay on the network infrastructure is often established to provide the controlled multicasting of subscriptions to avoid flooding. Although multicasting works well in terms of requiring the source to send only a single copy of the subscription to multiple destinations, it does not scale well to a large network of nodes that are sparsely distributed, as in an Internet environment. Furthermore, in a pub-sub network that is characterized by the frequent joining and leaving of nodes, frequent topological changes will significantly increase the overhead involved in maintaining the spanning tree. As such, it is necessary and important for us to derive an architecture that works efficiently in a distributed Internet-scale network. Importantly, such an architecture establishes a platform to effectively assemble publications and subscriptions in a distributed environment with acceptable communication and maintenance overhead. Additionally, it serves as the underlying delivery platform for the efficient dissemination of publication messages in a dynamically changing network environment.

4.3.1 Structure-aware Addressing in KBR

KBR is widely used in large-scale P2P systems for file sharing and media streaming. This routing mechanism supports not only good scalability and flexibility, but also poses intrinsic ability to tolerate topological changes caused by the dynamic nature of the underlying network. However, as mentioned, the KBR approach is essentially built around the concept of distributed file sharing. More specifically, it provides a distributed mechanism to rapidly locate a node that is hosting a certain piece of data. As such, it only support direct one-on-one mapping between the key and the content. However, the challenge in pub-sub is that information is generated dynamically and it is necessary to selectively disseminate the same information to multiple matching nodes. This is made more challenging when we consider that subscriptions may change over time as nodes join and leave the system. Hence, the "one-on-one" matching between query and content used in file sharing no longer suffices. To be more specific, the publication needs to be delivered to a group of users based on a coverage relation, i.e., the publication is delivered to subscribers whose subscription is covered by the publication. Unfortunately, with the direct hashing used in the traditional KBR addressing scheme, the coverage relation between contents cannot be reproduced using corresponding *keys*. Thus, the coverage-based publication delivery cannot be achieved using this addressing scheme. To address this problem, we introduce a novel *structure-aware addressing* method. With our method, the structural information of XML documents is integrated into the process of generating keys. This provides a necessary mechanism to facilitate coverage-based publication delivery. Details of the operation are given below.

We use an example to demonstrate the basic concept of structure-aware addressing. In Figure 4.1, the upper part shows the abstracted twig patterns in a tree-structure of an XML document: f_1, f_2, and f_3 which have some common branches, something that is termed as *structural resemblance*. The corresponding bloom filter representation for each branch is shown in the lower part of the figure. The doted line shows the individual mapping of each element (*a, b, ..., f*) and the associated paths to the position in the bloom filter. As we can see, the structural resemblance is revealed by the pattern of "1" bits (f_2 shares the same bit pattern of 3rd and 5th bits with f_1, while f_3 shares the bit pattern of 7th and 8th with f_1). We call this

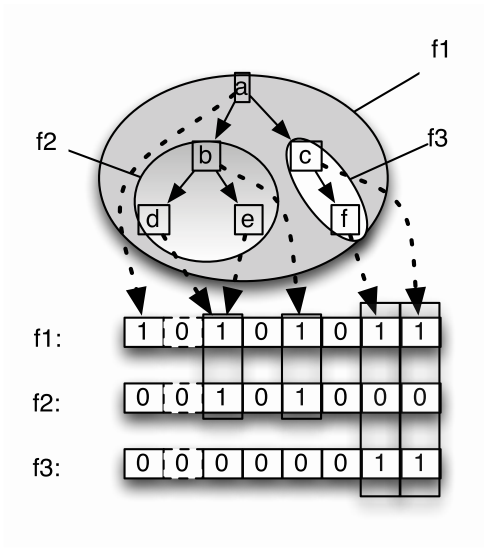


Figure 4.1: Structure-aware Addressing

characteristic structure-awareness. This characteristic indicates that the coverage relation between two tree-structures can be essentially identified by the coverage relation between their corresponding bloom filters. Hence, if a publication matches a subscription, the publication's bloom filter should also cover the subscription's bloom filter.

Since a bloom filter is a bit array, we can establish a new addressing scheme by directly using the bloom filter of the publication and subscriptions as their key in KBR. We name this addressing method *structure-aware addressing*. Using structure-aware addressing, given a publication key, all potential matching subscription keys can be easily generated based on the coverage relation between the keys. For example, given a publication with key 1101, we can generate all potential subscription keys (covered by the publication key) by simply setting each 1-bit in a publication key to wildcard (either "1" or "0"), as shown in Figure 4.2. In this way, we can achieve a "one-to-many" mapping from the publication key to the subscription keys. Since the key is equivalent to an address in KBR, the above one-to-many mapping correlates the coverage relation between publication and subscription to the coverage relation between their corresponding addresses (of their hosts). Identically, given a subscription key, all potential publication keys can be generated by treating the 0-bit as the wild card.

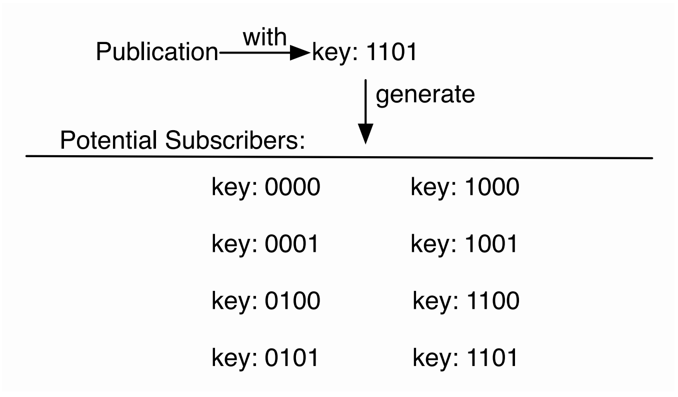


Figure 4.2: Potential Subscription Keys Generated by Covering Relation

4.3.2 KBR-based Propagation Mechanism

With structure-aware addressing we are able to establish a coverage-based relation between publication and subscription. The next step is to exploit this characteristic in the pub-sub context to establish a routing method between publisher and subscriber, so that publications and subscriptions can congregate and the matching publication can be efficiently delivered to the corresponding subscribers. Similar to the approaches used in publication and subscription propagation in typical spanning tree-based pub-sub systems, two propagation mechanisms can be applied to the underlying communication overlay, as described below.

KBR-based Publication Propagation (KBPP) In this technique, an arbitrary *subscription* s_j generated by node j is forwarded to the node that is in charge of s_j's key $key(s_j)$, We call the receiving node the *Rendezvous Node* (*RN* for short) of s_j, denoted as $R(s_j)$. A routing entry: $< s_j, j >$ is created accordingly at $R(s_j)$. Given a publication p_i, we send p_i to a set of potential matching subscription keys, denoted by S, generated by the method described in Section 4.3.1. Assuming that s_j was covered, we have $key(s_j) \in S$. Hence, p_i will arrive at $R(s_j)$. In this way we can guarantee that the publication and subscription will meet at the subscription's *RN*. When p_i arrives at $R(s_j)$, the evaluation process $eval(s_j, p_i)$ will be carried out on p_i and s_j. If this evaluation passes, $R(s_j)$ will send p_i to subscriber j based on the routing table entry $< s_j, j >$ created earlier. This process is repeated for all of the subscriptions of *RN*, from which the publication is delivered to all matching subscribers. As we can see, the KBPP is intuitive and simple to implement; however, it only works well in a small-scale application. This is because the propagation of publications is normally much more costly than the propagation of subscriptions (forwarding a whole XML document vs. forwarding an XPath expression in our case). With larger scale applications, the propagation would consume an excessive amount of bandwidth.

KBR-based Subscription Propagation (KBSP) In this method, once generated by node i, the publication p_i will be forwarded to the node that is in charge of publication's key $key(p_i)$. We call the receiving node the *Rendezvous Node* (*RN*) of p_i, denoted by $R(p_i)$. Given an arbitrary incoming subscription s_j, generated by j, we can propagate s_j to each potential publisher's *RN* by utilizing structure-awareness features of the corresponding keys. We denote this set of nodes (the set of the potential publisher's RN) as P. Assuming that s_j was covered by p_i, then we have $key(p_i) \in P$. When publication p_i arrives at $R(p_i)$, the evaluation $eval(s_j, p_i)$ will be carried out. If passed, p_i will be forwarded back to subscriber j using the reverse path that has been recorded during subscription s_j's propagation. Technically, KBSP works better when a subscription is stable (meaning that, once sent, the subscription will change less frequently; thus the system is able to reach a stable state). The publication could be highly dynamic, since there is no requirement regarding publication behavior. Therefore, KBSP can suit most of the typical pub-sub scenarios. In these scenarios, filters are less frequently changed and updated compared to the incoming publications. Compared to KBPP, KBSP has better scalability, as the propagation of subscriptions is less costly.

4.3.3 Matching Evaluations in the KBR-based Approach

In the previous section, we presented two methods, namely KBPP and KBSP, as two common routing methods for congregating of publications and subscriptions at the corresponding *RNs*. In this section, we address the design of matching evaluation $eval(s_j, p_i)$, involved in both methods.

As mentioned previously, the proposed addressing scheme (i.e. structure-aware addressing) provides the function of coarse-grained filtering by using the bloom filter as the address (key) in KBR and propagating the subscriptions or publications based on the coverage relation indicated by corresponding keys. This filtering capability functions mainly to reduce the number of nodes that need to be propagated. It essentially provides a mechanism to balance the loads across the entire network by dividing the nodes into smaller groups that are categorized by the structural resemblance of the pub-sub events that the individual RN is responsible for. Within each group, we can execute a matching evaluation $eval(s_j, p_i)$ at each RN, to provide a finer-grained filtering. In our design, the bloom filter-based matching proposed in Chapter 3 is adopted to provide such a finer-grained matching evaluation, a process that is discussed below.

For a given publication p or subscription s, we associate a bloom-filter $bl(p)$ or $bl(s)$ to the original p or s. At RN, we can use $eval(bl(s), bl(p))$ to do the matching evaluation instead of $eval(s, p)$. Since $eval(bl(s), bl(p))$ can be implemented using a simple bit-wise operation, this evaluation function can be carried out very efficiently, which in return, can drastically reduce the overhead caused by the matching evaluation.

It is important to note that the efficiency of the bloom filter is acquired by trading off with the probability of false positive matching. However, one can keep the false positive rate below a manageable threshold, by adjusting the bloom filter parameters (primarily the length l of the bloom filter). The adjustment of parameters and estimation of the false positive rate has been well studied elsewhere [7, 14, 8]. Recall that in the proposed addressing scheme, a bloom filter is created for each publication or subscription. Thus, in the pub-sub process two bloom filters are generated for each publication or subscription. Specifically, we used a shorter version of the bloom filter as the address (key) and then a relatively longer version as the subscription filter. Intuitively, we can optimize the bloom-filter generation process, so that it can generate two bloom-filters of different sizes in a single operation using the same set of hashing functions. This is accomplished by deriving a longer hash value and subsequently shortening it to a pre-defined size. With this method, we can further improve the system performance by reducing the computation overhead caused by the generation of bloom filters.

4.4 Hypercube Overlay

In this section, the issue of propagation efficiency is discussed. Compared to the traditional KBR-based approach, both KBPP and KBSP have a major advantage with regard to the propagation process: the propagation of publications (or subscriptions) only requires traversing through a subset of the nodes, rather than flooding the whole network. However, this propagation process may still become a performance bottleneck when handled improperly, especially in a large-scale application. For example, the straightforward solution of using unicast (sending to each candidate one by one) could easily result in network congestion around busy nodes. Thus, instead of using a unicast style propagation model, a multicast-like hierarchical dissemination structure may help alleviate heavy traffic conditions. In our work, we propose a hypercube overlay to achieve a hierarchical organization of the nodes based on their coverage relation as indicated by their keys. In our proposed architecture, nodes are organized into an ordered-hypercube based on their keys. The covering relation between keys of different nodes is maintained as a parent-child relation in an ordered-hypercube. With the hypercube structure, given any node with key K, every node, whose key is covered by K, can be reached by traversing through a subhypercube rooted at K. This traversal process can later be optimized into a tree traversal process. In this way, the propagation can be achieved in a multicast manner. Next, we discuss this process in detail.

4.4.1 Overview of a Hypercube

An n-dimensional hypercube $hypercube_n(V, E)$ has 2^n nodes, where V stands for the total set of nodes involved and E for the total set of edges. Each node $u \in V$ is represented by a unique n-bits bit array. We use $u[i], 0 \leq i \leq n-1$ to denote the *ith* bit of u (counting from the right). For every two nodes u, v in V, there exists an (undirected) edge (u,v) in E *if and only if* the hamming distance $d_{hamming}(u, v) = 1$, i.e., u differs from v by only one bit. A four-dimensional hypercube is shown in Figure 4.3, as an example.

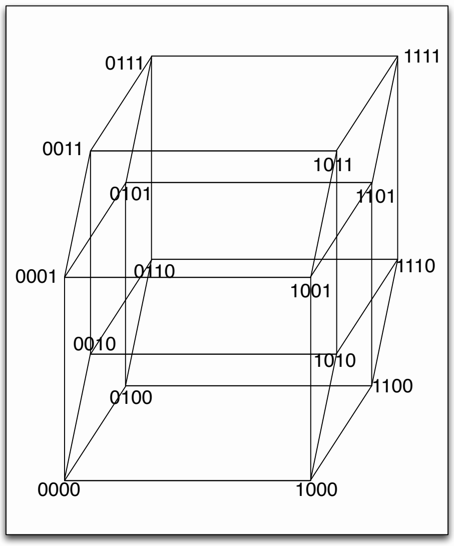


Figure 4.3: Four-dimensional Hypercube

We can obtain an *ordered-hypercube* by setting each edge $e \in E$, with a direction. Let us set the *weight* (essentially *hamming weight*) for node v as (n stands for the length of v):

$$W(v) = \sum_{i=0}^{n-1} v[i] \tag{4.1}$$

Then, for any connected node u, v, the direction of their edge (u,v) is $u \rightarrow v$ *if and only if* $W(u) > W(v)$ (refer to Figure 4.4). We call u the parent of v, and v a child of u. Given a node u, by selecting its children and its children's children, and so on (we call them u's descendant), we can get a sub-graph of $hypercube_n(V, E)$. It is observable that this sub-graph is also an ordered-hypercube. Since u is the root of this sub-graph, we call it the *subhypercube* rooted at u.

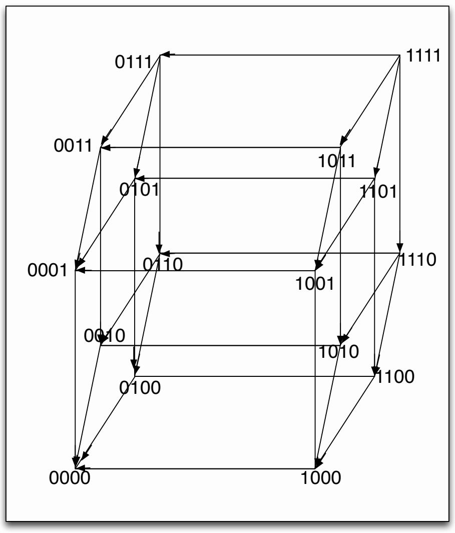


Figure 4.4: Four-dimensional Ordered-hypercube

4.4.2 Message Propagation via Hypercube

Recall that in KBR, each node is mapped to a bit-array as a key. Hence, using the keys we can organize nodes into an ordered-hypercube. Since the key is essentially a bloom filter, we inherit the terminology defined for bloom filters (such as: *cover, covered*), on keys. We may notice that in a hypercube, the key of a node will always cover the keys of its children node and be covered by keys of its parent node. This characteristic can facilitate the propagation process. For instance (refer to Figure 4.5), given a key of a publication, e.g. 1101, at its *RN:1101*, (assume that the publication has been already forwarded to its *RN*), we can just forward the publication to this node's children {0101, 1001, 1100}. Upon receiving the message, its children will repeat this process and pass the message to their children (e.g. 0101 will pass the message to 0100 and 0001). In this way, the publication can be delivered to all of the covering keys incrementally.

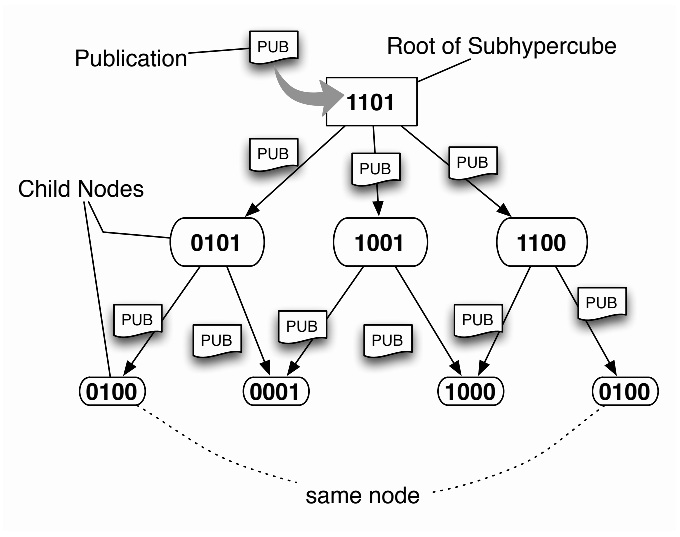


Figure 4.5: Publication Dissemination over Hypercube

Theorem 1. *Given an arbitrary key of weight* $\boldsymbol{n}$ *as the root of a subhypercube, we can reach any of its descendants within* $\boldsymbol{n}$ *step.*

Lemma 1. *Given a node* $\boldsymbol{a}$*, and one of its descendants* $\boldsymbol{b}$ *in a hypercube, if the hamming distance is:* $d_{hamming}(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{h}$ *we can always reach* $\boldsymbol{b}$ *in* $\boldsymbol{h}$ *steps.*

Proof of Lemma 1. *From node* $\boldsymbol{a}$*, we can always find a node* $\boldsymbol{c}$ *in its immediate children list, so that* $d_{hamming}(\boldsymbol{c},\boldsymbol{b}) = \boldsymbol{h\text{-}1}$*. Identically from* $\boldsymbol{c}$ *we can find* $\boldsymbol{d}$ *in* $\boldsymbol{c}$*'s immediate children so that* $d_{hamming}(\boldsymbol{d},\boldsymbol{b}) = \boldsymbol{h\text{-}2}$*. Repeat this until we find node* $\boldsymbol{b}$ *where* $d_{hamming}(\boldsymbol{c},\boldsymbol{b}) = \boldsymbol{h\text{-}h{=}0}$*. This takes* $\boldsymbol{h}$ *steps. Thus, Lemma 1 holds.*

Proof of Theorem 1. *Since the weight of this node is* $\boldsymbol{n}$*, the furthest descendant is an all-zero node (weight 0) in* $\boldsymbol{n}$*'s subhypercube. The hamming distance between these two nodes is* $\boldsymbol{n}$*. Thus, according to lemma 1, from the given node we only need to take* $\boldsymbol{n}$ *steps to reach the furthest node. Therefor Theorem 1 holds.*

Theorem 2. *Given a node* $\boldsymbol{a}$ *of weight* $\boldsymbol{n}$ *along a certain propagation path, node* $\boldsymbol{a}$ *only needs to send out at most* $\boldsymbol{n}$ *messages upon receiving a forwarding request.*

Proof of Theorem 2. *Because messages are forwarded along the same direction as edges in the ordered-hypercube, since* $\boldsymbol{a}$*'s out-degree is* $\boldsymbol{n}$*, (*$\boldsymbol{a}$ *could only have* $\boldsymbol{n}$ *immediate children),* $\boldsymbol{a}$ *could at most send out* $\boldsymbol{n}$ *messages. Thus, Theorem 2 holds.*

Consider a network with N nodes. Assume each hop in propagation takes the same amount of time. Theorem 1 indicates that the time used for propagation is affected only by the weight of the publication/subscription key, bounded by *d*log(N)*, with d denoting the delay per hop. Theorem 2 indicates that the forwarding count for each node during one propagation event is only directly affected by the weight of the key, and bounded by *log(N)*.

4.4.3 Multicasting via Hypercube

The hypercube is not strictly a tree structure itself, thus multicasting through a hypercube (from the root to each level of the descendant nodes) is not optimal. This is because, unlike a tree structure, there are multiple paths between two nodes. To achieve a better performance, we adopted the method introduced in [56], which provides a viable way to induce a tree structure from a subhypercube rooted at a given node. This process is briefly explained below.

Given an n-dimensional ordered-hypercube $Hypercube_n(V, E)$, (using the same definition as in Section 4.4.1) a spanning tree rooted at u, denoted by $SPT(u)$, can be established by iteratively selecting appropriate children for each descendant v of u, $v \in V$, starting from u, based on the method presented below.

Let p be the bit position (also called the dimension) satisfying $v[p] \oplus u[p] = 1$ and $v[i] \oplus u[i] = 0, \forall i < p$. We set $p = n$, if $u = v$. Let $J_v = \{j | v[j] \neq 0, j < p\}$, ($J_v = \emptyset$, if $p \leq 0$). Then the children of v can be defined as:

$$\begin{cases} v[n-1]...v[p]...\overline{v[j]}...v[0], \forall j \in J_v & \text{if } p \neq n \\ v[n-1]...\overline{v[j]}...v[0], \forall j \in J_v & \text{if } p = n \end{cases} \tag{4.2}$$

It is also possible to acquire the spanning tree in question by choosing the appropriate parent node for each v. The parent for node v is defined as:

$$\begin{cases} v[n-1]...v[p+1]\overline{v[p]}v[p-1]...v[0] & \text{if } p \neq n \\ \emptyset & \text{if } p = n \end{cases} \tag{4.3}$$

Figure 4.6 demonstrates the above algorithm in a four-dimensional subhypercube, rooted at *node:1011* (shown on the left side). At the root, we have $J_v = \{0, 1, 3\}$, so according to Figure 4.2, the 1st, 2nd and 4th bits (counting from right) are reversed to 0 accordingly, yielding three children for *1011*: *1010, 1001* and *0011*. At node *0011*, we have $J_v = \{0, 1\}$, so we reverse the 1st and 2nd bits accordingly, yielding two children for *0011*: *0001* and *0010*. Applying this algorithm to the rest of the nodes, we can acquire a tree-structure, as shown on the right side of Figure 4.6.

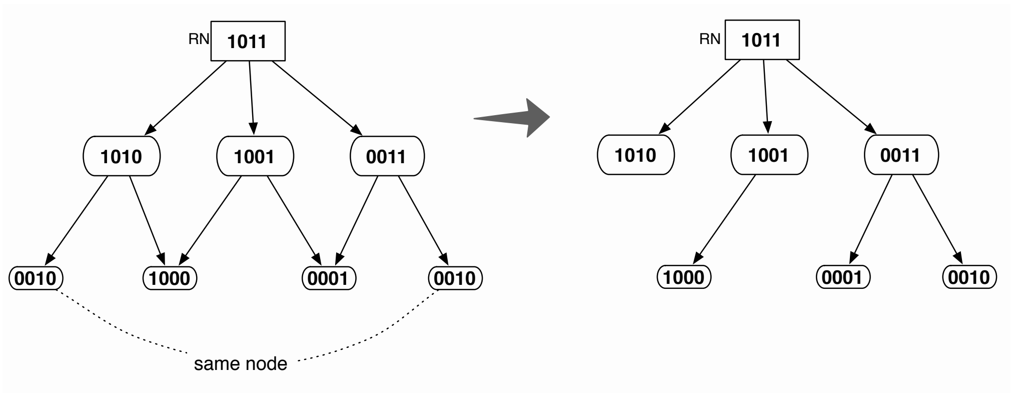


Figure 4.6: Multicasting-Tree Induced from a Hypercube

This approach can be applied to the proposed pub-sub system to improve on the propagation efficiency. Assume that a *publication* arrived at node *1011* as its rendezvous node. Rather than performing redundant forwarding through a subhypercube, the publication can be efficiently propagated throughout the tree-structure induced from the subhypercube. Notice that the above example uses publication propagation as an example to demonstrate the symmetric property of the hypercube (by reversing the roles of 1-bit and 0-bit, we can get a reversed ordered-hypercube). A similar procedure can be adopted for subscription propagation. The pseudo-code is shown in Algorithm 1.

Algorithm 1: Hypercube-induced Tree for Subscription Propagation

```
Data: mykey, fromkey, Msg
Result: send Msg to apropriate parent node, so that the forwarding
        follows a hypercube induced tree structure
startGenerating = false;
foreach bit myKey.getBit(i) in myKey do
    if fromKey.isUnspecified() then
        // this means I am the root;
        startGenerating= true;
    end
    else if myKey.getBit(i)!=fromKey.getBit(i) then
        startGenerating= true;
    end
    if startGenerating then
        if !mykey.getBit(i) then
            tmpkey= mykey;
            tmpkey.setBit(i,true);
            send(Msg, tmpkey);//deliver Msg to the generated key;
        end
    end
end
```

4.5 Exploiting Common-Paths

The basic KBSP model mentioned previously requires a publication and its potential matching subscriptions to congregate at the publication's *RN*. After undergoing the matching evaluation, the publication will be forwarded to the matched subscribers using the reverse path from which the corresponding subscriptions came. This implies that each subscription, if matched, will produce a corresponding publication delivery event. Therefore, if several matched subscribers share the same path (or part of a paths) to the publisher, multiple instances of publication deliveries will be triggered. In other words, the publication will be delivered multiple times along the shared path for each matching subscription. This is obviously redundant and suboptimal. To address this problem, in the implementation, we made some modifications to the basic approach in KBSP to exploit the common paths of subscriptions during publication delivery. The details of these modifications are discussed below.

During subscription propagation, we maintain a routing table at each intermediate node that records all of the subscriptions that traversed the node. This entry can be represented in the abstracted form: <*filters*, *Node*>. The *filters* field stores the subscription filters, while the *Node* field stores the address of the previous node through which this subscription traversed. When node k receives a subscription s_j from its directly connected neighbor n, it will create a routing entry $< s_j, n >$. When any incoming publication p_i arrives at k, it will be forwarded to n if it passes the evaluation $eval(s_j, p_i)$. In this way, the publication can be routed to the matching subscriber hop-by-hop based on the evaluation result at each intermediate node along the dissemination path. Fundamentally, the routing table has two major functions:

1. By maintaining the address of the previous node through which each subscription traversed, a reverse path to that of the forwarding of the subscription can be established. This can avoid the overhead imposed by source routing, in which routing information are recorded and carried along by the messages.

2. With hop-by-hop filtering, a multicasting-style of disseminating publications is achieved by exploiting the common paths among matched subscriptions. Using the new publication dissemination scheme, the shared path for different subscriptions is exploited, so that publication will only traverse once on the shared path for those subscriptions.

One drawback of this enhanced dissemination method is the overhead imposed by storing such a routing table. However, since only directly connected neighbors would produce a routing table entry, based on Theorem 2 the entry number in this routing table is bounded by W_k, where W_k denotes the *weight* of k's key. Considering that W_k will never exceed $log(N)$, where N is the number of nodes involved, the routing table is actually quite small. Hence, the overhead of maintaining the routing table is acceptable. Another drawback is the computation overhead caused by the matching evaluation process at each intermediate node through which each publication traverses. However, since bloom filter-based matching is essentially a bitwise operation between two bit-arrays, this overhead is indeed quite low.

It is important to note that the different subscription filters received by certain nodes can be aggregated. This would allow each node to maintain only one aggregated filter for each of its children. Doing so can greatly relieve the matching evaluation process, because a publication only needs to be evaluated on one filter for each routing entry, instead of many. It can also reduce the traffic generated during the propagation process, because succeeding filters that already have been covered by the aggregated one will no longer be forwarded. This aggregation process can be done very efficiently, as aggregating bloom filters equals to a bitwise *OR* operation among corresponding bloom filters (essentially a bit-array). This aggregation operation among bloom filters would impose extra false positive matching. However, we can reduce this false positive rate by increasing the length of the bloom filter, keeping it under a manageable threshold.

Under this modification, the end results of filtering in KBSP are essentially achieved in two stages: In the first stage, the filtering is coalesced into the addressing scheme. It filters out large numbers of irrelevant publications in a coarse-grained fashion. In the second stage, a finer-grained filtering is carried out during publication propagation to further prune out false deliveries at a low cost. Compared to subscription propagation in a conventional spanning tree-based infrastructure, our proposed method does not require flooding subscription filters throughout the whole network. Instead, we only need to install the filter in a small subset of nodes (members of the subhypercube for a given publication). More importantly, the hypercube structure organizes nodes into a hierarchical structure so that the propagation of subscription filters can be efficiently achieved in a multicast manner. Our experiments, which are described in the next section, demonstrate the effectiveness of the approach.

4.6 Performance Study

In this section, we evaluate the performance of the proposed design through extensive simulations. We first present some background information on our experiments in section 4.6.1. Then, we present the results of the network performance evaluations of the overlay architecture based on metrics such as *link stress*, *link stretch*, and *percentage gain* (described in section 4.6.2). Following that, we evaluate the efficiency of the two propagation methods, namely KBPP (described in section 4.6.3) and KBSP (described in section 4.6.4).

4.6.1 Setup of the Experiment

We used OMNeT++ [62] as our main simulation platform. OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators [62]. In our experiment setup, we make use of several extension frameworks available to ease the development of the simulation as well as to extend the capabilities of OMNeT++. The INET Framework [60] is an open-source communication network simulation package for the OMNeT++ simulation environment. The INET Framework contains models for several wired and wireless networking protocols, including UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, and many others [60]. OverSim [64] is an open-source overlay and peer-to-peer network simulation framework for the OMNeT++ simulation environment. The simulator contains several models for structured (e.g., Chord, Kademlia, Pastry) and unstructured (e.g., GIA) P2P systems and overlay protocols [64]. These two frameworks were both used in our simulation setup. We adopt Chord as our KBR in our experiments.

The sample dataset (XML documents) was acquired from the Niagara project [13]. The total number of XML documents in the sample is 3,680, conforming to nine distinctive DTDs (refer to Table 3.1 for more details). These XML documents have been extracted from several real case applications across different domains. Therefore, the sample is representative enough for a common applications scenario. We generate XPath expressions using tools provided by the yFilter [16].

All of our experiments were performed on a 3.8GHz Core 2 Duo machine with 2G of memory, running on 32 bit-Linux OS. The version of OMNeT++ that we used was 4.0. The OverSim version was *release–20101103*. The INET that we used was a patched version provided by OverSim.

4.6.2 Evaluation of Overlay Performance

In this section, we evaluate the performance of the overlay architecture. The results from the experiments pertaining to the overlay multicasting performance, such as *link stress*, *link stretch*, and *percentage gain* are analyzed and presented in this section.

Link stress *Link stress* is defined as the number of identical packets carried by a physical link. It is one of the standard network metrics to evaluate an overlay performance. In this experiment, we continually initiate the random-sized propagations of publication messages (with the expectation of the key saturation settings of 20%, 50%, and 75%) from randomly selected peers for duration of 7200s. Subsequently, we recorded the *link stress* on all existing physical links during the simulation. The saturation of the key is defined as the proportion of 1-bits in a given key: W/L, where W denotes the hamming weight of the given key and L denotes the length of the given key. We group the value based on the saturation of publication key. The results are shown in Figure 4.7. As is evident in Figure 4.7, there is an increasing trend in the *link stress* as the number of nodes increases. The increase in the number of average saturations of publications can also cause an increase in *link stress*. This can be briefly explained as follows: A larger hypercube implies more overlay paths. Each of those paths has a certain number of overlapping physical links. Hence, more overlay paths means that there is a higher chance of overlapping in physical links. Since the increase in the number of nodes as well as in the saturation of publications could both lead to larger subhypercube sizes, they could both cause an increase in *link stress*.

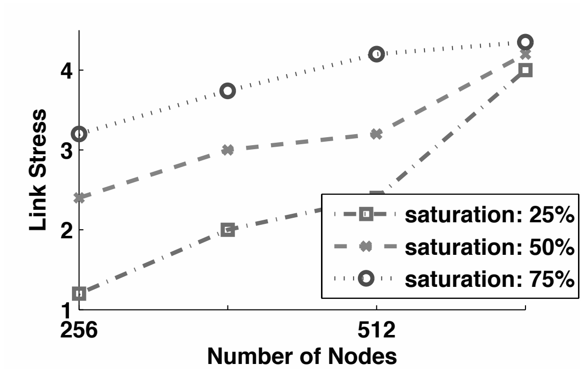


Figure 4.7: Link Stress

Link Stretch This metric is defined by the ratio of the delay between two nodes along the overlay distribution topology to the delay of the direct physical path. It is also called the relative delay penalty. *Link stretch* is another standard overlay performance metric used to evaluate the overlay performance. In this experiment (refer to Figure 4.8), we start a packet transmission between a pair of randomly selected nodes, and measure the delay in two cases: 1) where the transmission is carried out via the overlay layer; 2) where the transmission is carried out via the UDP layer. The link stretch can be calculated using the measurement results from these two cases. We repeat this process 1000 times. The average value is recorded and presented in Figure 4.8. For the purpose of comparison, we also present the *link stretch* measurement of a generic P2P application that uses Chord as its overlay. We can see from the figure that the *link stretch* is identical in both cases. This is because the *link stretch* is mainly dictated by the overlay implementation. Since we used Chord as the common overlay for both, the link stretch should be more or less the same. The result also shows an almost-flat trend. This indicates that the *link stretch* will not deteriorate as the number of nodes grows.

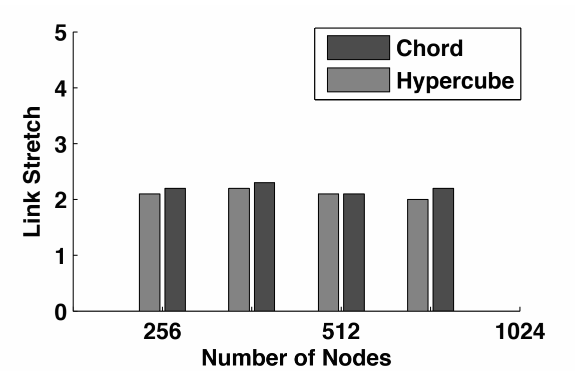


Figure 4.8: Link Stretch

Percentage Gain using Hypercube Overlay This metric indicates how efficient our multicast dissemination structure is compared to the unicast transmission. The metric is defined as:

$$\delta = 1 - L_m/L_u \tag{4.4}$$

where L_m is the total number of multicast links in the distribution tree and L_u is the sum of all unicast hops. The variable δ represents the percentage gain in multicast efficiency over unicast. As δ approaches zero, multicast and unicast performances are identical with little to no savings in bandwidth. As δ approaches one, all receivers share a single multicast path resulting in the maximum possible bandwidth efficiency.

We compare our method with an application layer multicasting system (which can be treated as a topic-based pub-sub) named *Scribe* [10]. In our experiment, we initiate the random-sized propagations of subscription messages from randomly selected peers for a duration of 7200s. The expectation of key saturation is set to 50%. The average number of multicasting nodes in the *Scribe* is set to be equal to the average number of nodes involved in each propagation event in the proposed system. In Figure 4.9, we can observe that as the number of nodes increases, the percentage gain drops. This is because as more nodes were involved, the traffic in the underlying physical paths became dense (with more hops per dissemination path), leading to more overlapping physical links along a dissemination path. The increasing number of overlapping physical links would reduce the *percentage gain*, since L_m is increased. It should also be noted that both systems produced almost identical results. This means that at least the same degree of efficiency can be attained using our approach in comparison to the simple topic-based *Scribe*. Considering the fact that Scribe is merely a multicasting protocol, which has neither XML filtering capacity nor coverage-based propagation functionality, the performance of our proposed system is very promising.

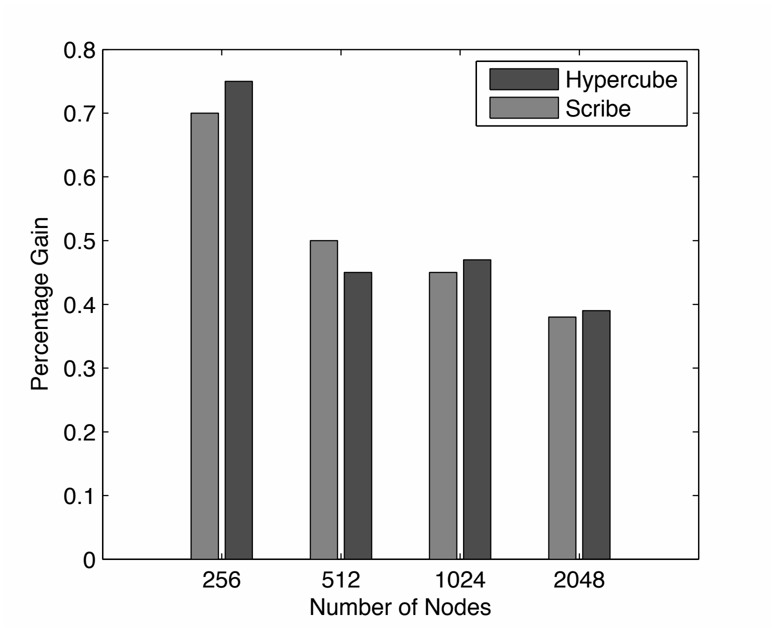


Figure 4.9: Percentage Gain Using Hypercube Overlay

4.6.3 KBPP

In this section, we present two traffic-based evaluations by varying the saturation of publication keys for KBPP. First, we evaluate the scalability of KBPP, in terms of the overall traffic produced as the number of nodes involved in each propagation event is increased (refer to Figure 4.10). Then, we validate the benefit of using hypercube overlay in the propagation process, by comparing the traffic on maximum-link traffic with and without the hypercube as shown in Figure 4.11. In these experiments, we used a fixed-size network with a total of 2048 nodes. The average size of a publication message is 4kB.

In Figure 4.10, the traffic is measured as the total number of bytes traversed across each overlay hop. For comparison, we also included the measurement of the flooding-based approach that has been widely used in conventional spanning tree-based pub-sub systems. As shown in the figure, the traffic increases exponentially as saturation increases in hypercube-based propagation, indicating poor scalability. This is primarily because the increase in the saturation of publication key would directly cause the size (depth) of the subhypercube for the publication to grow, leading to more traffic being generated. However, compared to flooding, the performance is still better even in the worst case in our test (at 60% saturation).

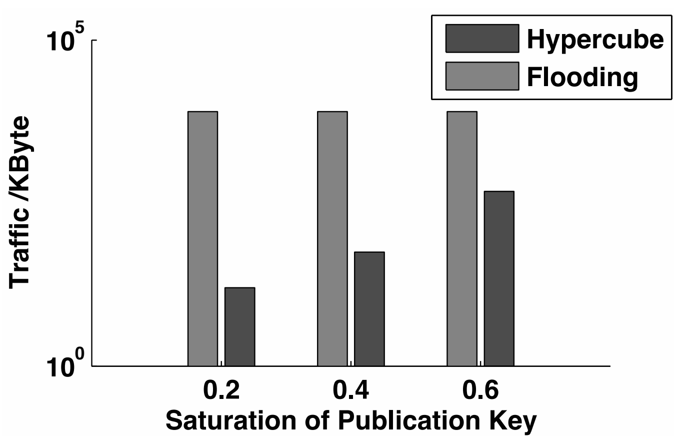


Figure 4.10: Total Traffic

In Figure 4.11, the traffic is measured as Bytes traversed over the maximum-traffic link (MTL for short). MTL denotes the link that bears most of the traffic for each propagation process. MTL is normally where potential congestion occurs. We tested message dissemination through KBR in two cases: one with the hypercube, and one without. The results show that in the hypercube, the peak bandwidth on MTL increase exponentially when the system scales. Without a hypercube, the traffic on MTL is several orders of magnitude higher than in the case where a hypercube is adopted, indicating a higher risk of congestion.

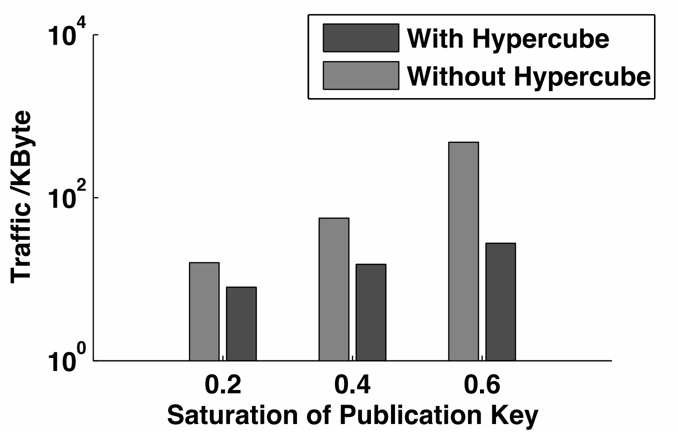


Figure 4.11: MTL Traffic

4.6.4 KBSP

In this section, we present the performance evaluation result for KBSP. By varying the length of the subscription filter, denoted by l in the following paragraph, the system is evaluated in three cases: l=0 (no filtering), *l=128*, and *l=1024*. The hash function number is set to 1 in both the *l=128* and *l=1024* setups. The testing configurations of all of the experiments mentioned in this section conforms to the parameters listed in Table 4.1.

Table 4.1: Simulation Parameters

Average XPE Length	3
Simulation Time	8200(s)
Publication Rate	0.1
Subscription Rate	0.1
Initial Phase Wait	1000(s)
Measurement Time	7200(s)

Publication Delivery To study publication delivery behavior, we conducted two groups of experiments. In the first group, we measure the total number of publications that have been sent and received. The result is the accumulative value counted on all of the nodes involved during the simulation measurement period.

In Figure 4.12 and 4.13, the results are collected from three experiment scenarios, with subscription filters of different lengths, indicated in the legend by *l=0*, *l=128*, and *l=1024*. For each setup, we vary the number of participating nodes, and record the total number of both sent and received messages. We observe a linearly increasing trend for both sent (refer to Figure 4.12) and received (refer to Figure 4.13) messages as the number of nodes increases for each scenario. This trend can be briefly explained as follows: As the number of nodes increases, more nodes will begin to publish; Hence, the number of publication messages will increase accordingly. At the same time, as more nodes join, the number of subscribers will increase, leading to an increase in the number of matching subscribers. Therefore, the number of the messages received will also increase. It is also observable that the number of the total messages sent is identical in all three cases (*l=0, l=128*, and *l=1024*); refer to Figure 4.12. This is in accordance with the fact that the number of publication messages is directly affected by the number of nodes. It is important to note that as the subscription filter length l increases, the number of publication messages received by peers will decrease (refer to Figure 4.13). This means that a longer subscription filter can filter out more false publication messages.

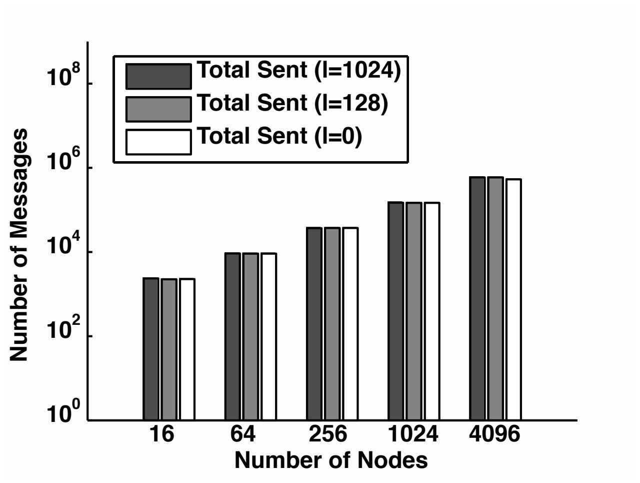


Figure 4.12: Total Sent

Next, we evaluated the publication coverage, which is defined as the number of subscribers for each publication that have successfully received the publication message. In other words, the metric indicates the number of successful deliveries of each publication. For each publication we first recorded the number of subscribers that had received this publication. Then, we calculated the average and maximum value of this metric. The results are as shown in Figure 4.14 and Figure 4.15. The results are collected under three scenarios, with different lengths of subscription filters, indicated in the legend by *l=0*, *l=128*, and *l=1024*. For each scenario, the mean (refer to Figure 4.14) and max (refer to Figure 4.15) values were recorded, while varying the number of participating nodes. As we can see

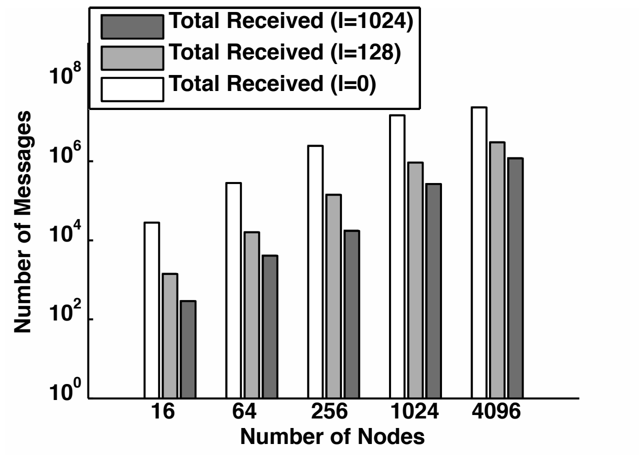


Figure 4.13: Total Received

from both Figure 4.14 and Figure 4.15, with the increase in the length of the bloom filter of the subscription filter, the number of deliveries can be drastically reduced. This means that false publishing can be reduced by increasing the length of the subscription filter. This is because large portions of the dissemination paths for unmatched publications (caused by false positive errors) are pruned out by the subscription filter during the dissemination of the publications. In the worst case (with *l=0,* and *4096* nodes participating) shown on Figure 4.15, a publication is received by at most 200 nodes (around 4% of the total of 4096 nodes), which is still quite reasonable.

Hypercube Layer Stress and Overhead At the hypercube layer, each node is responsible for forwarding the incoming subscriptions and publications based on the matching evaluation. As mentioned earlier, the cost of a matching operation is comparatively small due to the bitwise evaluation. Hence, the stress is mainly dictated by the frequency of the forwarding

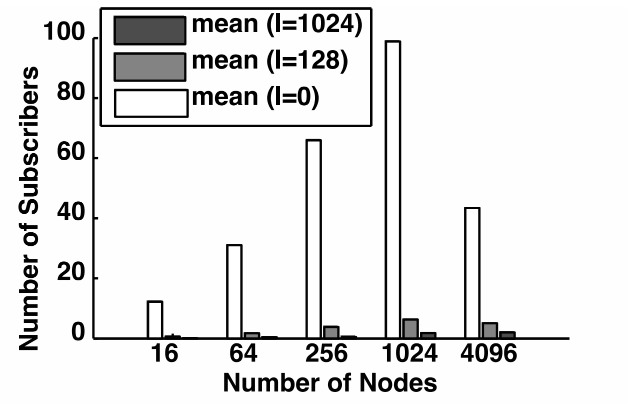


Figure 4.14: Publication Coverage: Mean

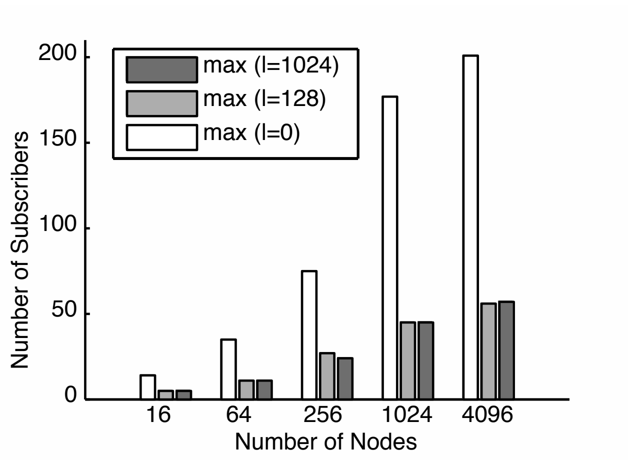


Figure 4.15: Publication Coverage: Max

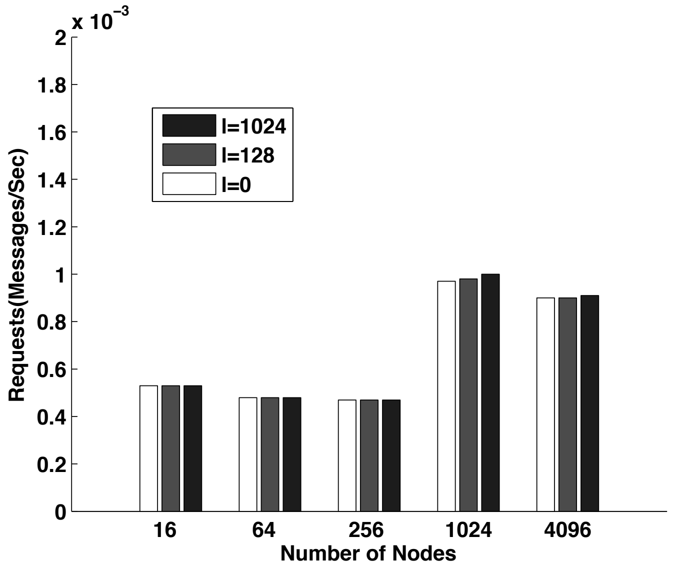


Figure 4.16: Requests per Second: Subscription

requests for incoming publications and subscriptions. Thus, it can be evaluated by measuring the frequency of incoming requests.

First, we evaluate the hypercube layer stress in terms of the frequency of incoming subscription requests, varying the number of nodes in the system. The average number of subscription requests received by each node is shown in Figure 4.16. The results are identical in all three of the experiment scenarios (with *l=0*, *l*=128, and *l*=1024, respectively). This outcome can be briefly explained as follows: In subscription propagation, subscription requests are propagated along the paths in a hypercube-induced tree structure. Since same hypercube structure is shared among all three scenarios, the results in the three scenarios should follow the same pattern.

Next, we evaluate the hypercube layer stress in terms of the frequency of incoming publication requests in the three scenarios (*l*=0, *l*=128, and

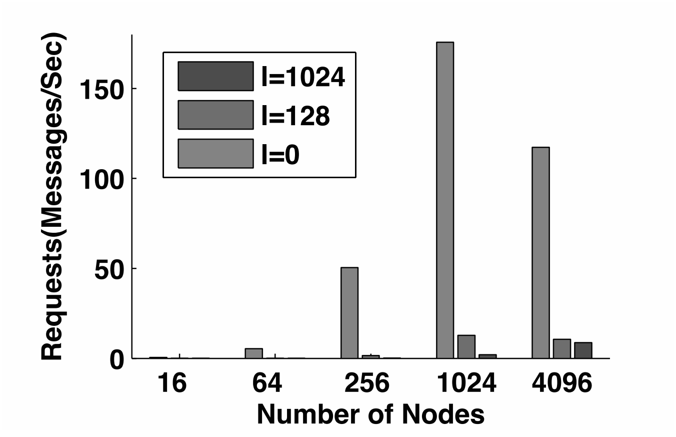


Figure 4.17: Requests per Second: Publication

l=1024), while varying the number of nodes in the system. The average number of publication requests received by each node per second is shown in Figure 4.17. According to the figure, as the number of nodes increases, the frequency of publication requests increases in the initial phase (starting from 16 nodes to the 1024 nodes setup). However, an observable but not significant reduction was recorded at 4096-node setup. This convergence property indicates good scalability of the system with respect to the hypercube overlay stress. It should be noted that, with the subscription filter length l=1024, the number of publications handled by a single peer is reduced by as much as 98% (at the 1024 nodes setup) compared to the case where no subscription filter is applied (l=0). This means that the hypercube overlay stress can be reduced significantly by increasing the length of the subscription filter.

Finally, we evaluate the overhead of the hypercube overlay. The storage overhead is mainly incurred by the storage of the routing table structure, which is comparatively small (6Kbits on average). The computation overhead is caused by the matching evaluation, which is essentially a bitwise operation, and hence is negligible. Thus, the overall overhead is mainly contributed by the communication overlay, which is defined as traffic dedicated for the routing table exchange. We vary the number of nodes, and the results are shown in Figure 4.18. Notice that this overhead is below the level of 2Bytes/Sec. This is only 13% of the KBR maintenance traffic and 0.01% of the total traffic. This validates the claim that the proposed hypercube-based approach incurs an insignificant amount of traffic overhead.

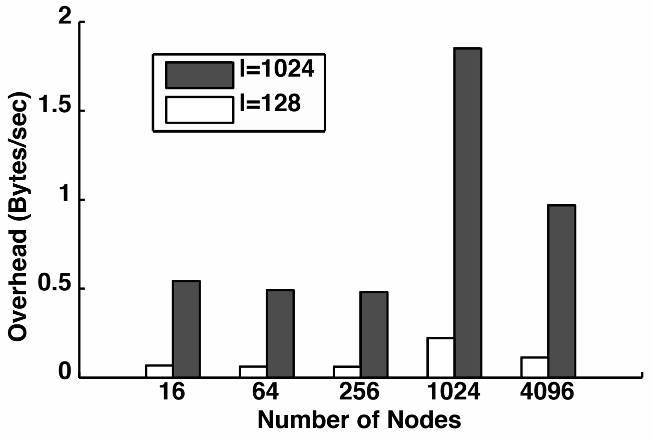


Figure 4.18: Communication Overhead of the Hypercube Overlay

Traffic In this section, we focus on the traffic generated on the KBR layer. The overall traffic consists of application traffic and maintenance traffic. We group the experiments into two categories, one for application-generated traffic, and the other for maintenance traffic.

First, we evaluate the application-generated traffic. The experiments are conducted based on three scenarios: l=0, l=128 and l=1024. The incoming and outgoing traffic are shown in Figure 4.19 and in Figure 4.20, respectively. It is observable that for the l=1024 scenario, as the number of participating nodes increases, both incoming and outgoing traffic show an increasing trend until the number of nodes reaches 1024, then they start to converge to a certain value (8KBytes/Sec for incoming traffic and 10KBytes/Sec for outgoing traffic). A similar trend can be observed for the l=128 scenario. This property indicates the proposed system has good scalability, which is largely attributed to the underlying load-balancing mechanism provided by hypercube overlay. We may also notice that the increasing length of the subscription filter can largely reduce the application-generated traffic, especially for setups of 16, 64, and 256 nodes. This is because increasing the length of the filter improves the precision of the filter, reducing the number of false matchings.

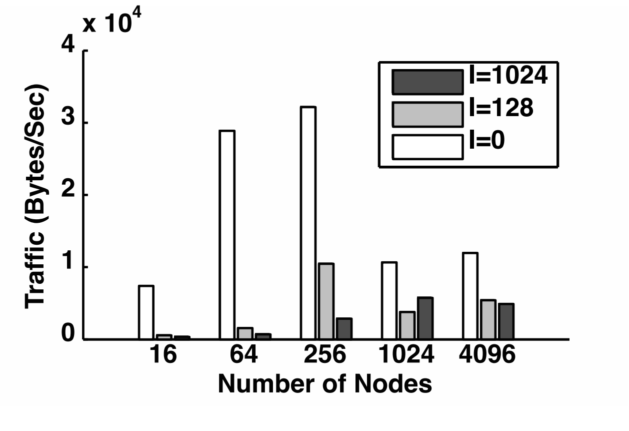


Figure 4.19: Application-Generated Traffic: Incoming

Next, we evaluate the maintenance traffic. Similar to the previous experiment, we conduct the experiment based on three scenarios, using

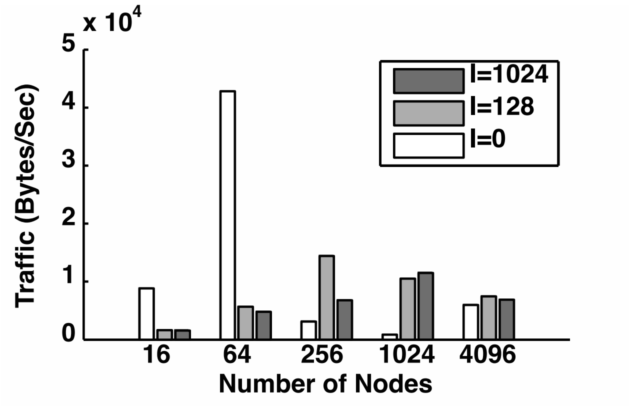


Figure 4.20: Application-Generated Traffic: Outgoing

subscription filters of different lengths. The incoming and outgoing maintenance traffic for KBR is presented in Figure 4.21 and Figure 4.22, respectively. It is observable that in both figures, a similar amount of maintenance traffic is generated in all three scenarios. This is because KBR maintenance is independent of the upper layer implementation. Hence, varying the length of the subscription filters will have no impact on KBR maintenance. We may also notice that both incoming and outgoing traffic show a mildly increasing trend (considering the x-axis as being in log scale) as the number of nodes increases; however, the volume of maintenance traffic is quite reasonable (below 20Bytes/Sec for both incoming and outgoing traffic in our experiment).

Publication Multicasting Size The publication multicasting size is defined as the number of overlay links in the multicasting tree produced by a given publication. To calculate the publication multicasting size, we monitored and calculated the total number of hops that publication messages

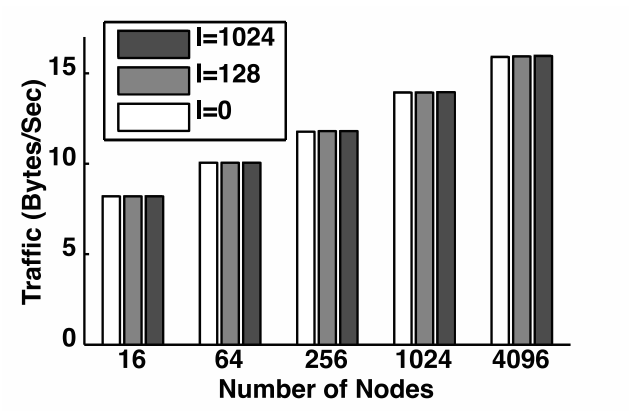


Figure 4.21: Maintenance Traffic: Incoming

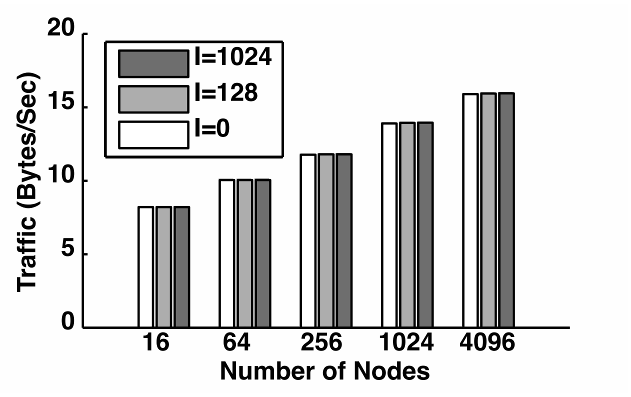


Figure 4.22: Maintenance Traffic: Outgoing

traversed during propagation for each publication event. As delivery of the matching content is always guaranteed by the matching algorithm (only false positives may occur, not false negatives), the smaller the publication multicasting size for each publication, the better the efficiency that can be obtained. In Figure 4.23, we evaluate the publication multicasting size in three scenarios, in which we use subscription filters of different sizes, (labeled by l=0, l=128, and l=1024 in the figure). Notice that the result captures the average size of all of the publication multicasting trees; thus, it is possible for a value of below “1” to appear, which means that there are many publications that are “0” in size. This is because a publication can be filtered out at its initial hop, yielding a measurement value of “0”.

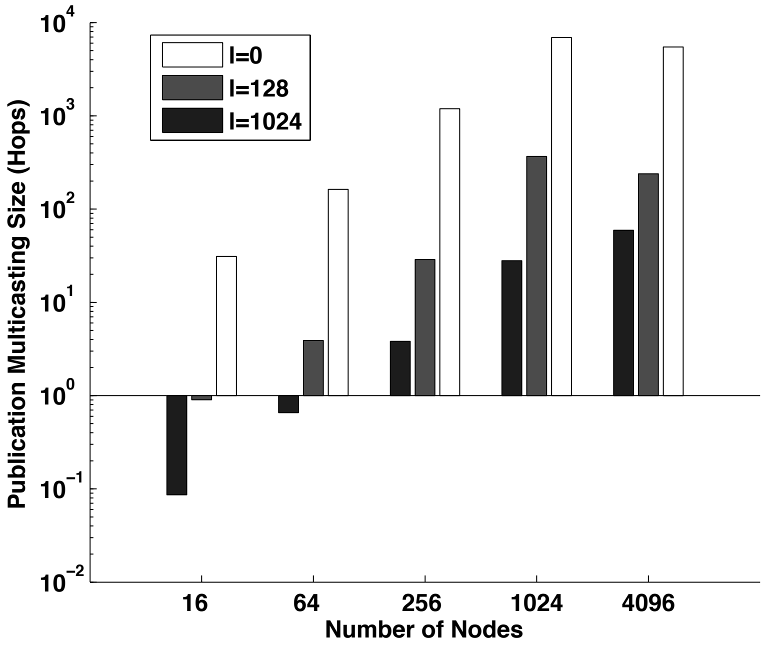


Figure 4.23: Publication Multicasting Size

In Figure 4.23, we can see that as the number of nodes increases, the multicasting size increases. This is mainly caused by the increase in the size of the hypercube for each publication as the number of nodes increases. It is important to note that all of the measurement values at l=1024, are very small (below 100), indicating that most of the publications can be filtered out at early steps along their propagation paths.

To further study the impact of the length of the subscription filter to the publication multicasting size, we calculated the ratio between multicasting size in the l=128 scenario and that in the l=0 scenario (labeled as l=128), and also the ratio between the l=1024 and l=0 scenarios (labeled as l=1024). The results are shown in Figure 4.24. It is easy to observe that using a *subscription filter* can significantly reduce multicasting size by several orders of magnitude.

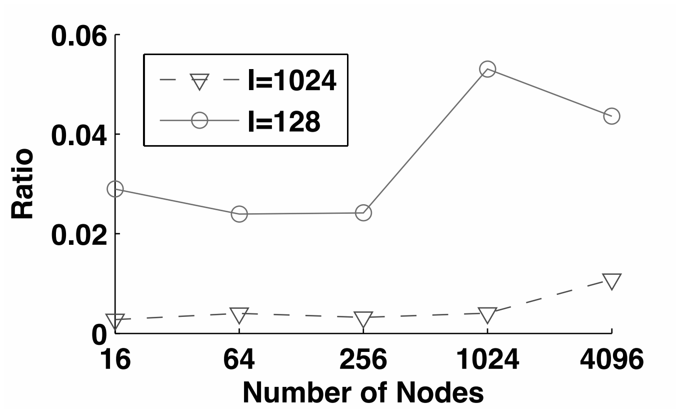


Figure 4.24: Publication Multicasting Ratio

False Positive Rate To test the false positive rate, we define two new metrics: the *absolute false positive rate* and the *relative false positive rate*. The *absolute false positive rate* is defined as:

$fp_{abs} = FalsePositiveCount/MessageReceived$

FalsePositiveCount is the number of false positives occurring for a given peer, while *MessageReceived* captures the number of messages received by the peer in the course of the experiment. The above definition captures the absolute false positive rate experienced by a given node. However, since we will rule out more false matching subscribers as we increase the length of the subscription filter, the total number of messages received by a peer is expected to drop, leading to an increase in the absolute false positive rate measurement. Hence, we defined another metric- the *relative false positive rate*- as:

$$fp_{rel} = FalsePositiveCount/MessageReceived_{baseline}$$

where $MessageReceived_{baseline}$ is the number of messages received by the peer in question when *l=0* (i.e. as without subscription filter).

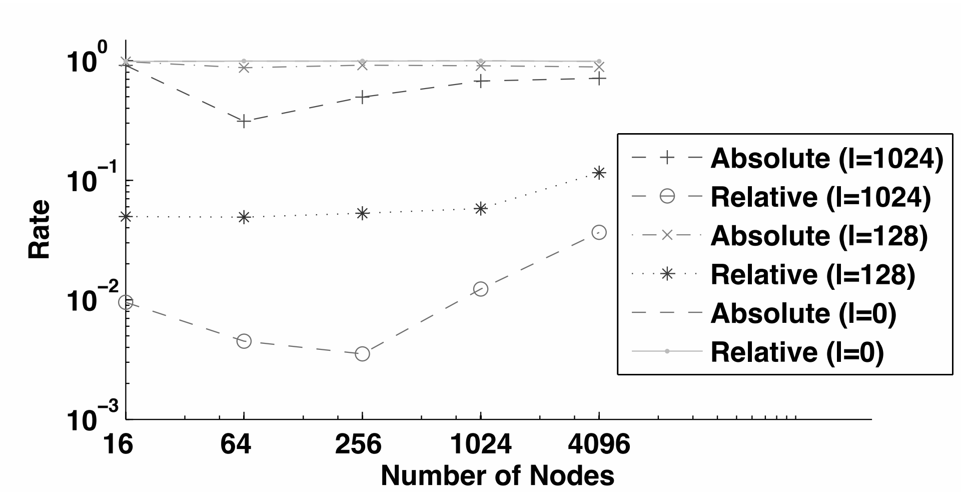


Figure 4.25: False Positive

In Figure 4.25, both absolute and relative false positive are recorded for three application scenarios with subscription filters of different lengths (indicated by l=0, l=128, and l=1024), while varying the number of participating nodes. As we can see in Figure 4.25, by using the subscription filter the rate of relative false positives can be significantly reduced (both the *l=128* and *l=1024* experiments yield a relative false positive rate of below 0.1). Increasing the size of the subscription filter could further reduce the false positive rate to as low as 0.004.

4.7 Related Work

In summary, routing methods found in general pub-sub systems can broadly be classified into two categories: the spanning tree-based approach and the key-based approach.

Spanning Tree-based Approach. Many established pub-sub system designs such as SIENA [9], Gryphon [2], and Elvin [52] have adopted the spanning tree approach. Briefly, the aim of this approach is to build a universally accessible spanning tree that connects all of the nodes (brokers). A subscription is sent, aggregated, and stored at each intermediate node along the upstream direction, while a publication is forwarded downstream and filtered by the previously stored subscriptions at each intermediate node. As a result, a multicast infrastructure is built. The spanning tree-based approach achieves a decent performance on a small scale and a less dynamic setup. However, the performance would degrade rapidly in a non-dedicated network environment where nodes can join and leave freely. In addition, the overhead imposed by maintaining the routing table (filtering table) could severely limit the scalability of the system.

KBR-based Approach. Some recent studies [46, 11, 10, 71, 59, 78, 1, 29, 24] have introduced the KBR approach that is widely adopted in P2P file sharing systems. KBR-based overlay technology supports large-scale exchanges of data between distributed clients. It is considered a standard solution for distributed file sharing and media streaming. Compared to the spanning tree-based method, KBR-based event delivery has many advantages such as robustness, scalability, and adaptability. However, these studies mostly focused on either a simple group-based pub-sub system [46, 11, 10] or a simple content-based pub-sub system [71, 59, 78, 1, 29, 24]. In the former case, all messages (publications) are grouped into several categories defined *a priori* and subscription is reduced to a matter of simply joining certain established group(s), while in the latter, a fixed filter-dimension is required. In our case, however, the structure of arbitrarily generated XML document is extremely volatile and cannot be defined at design time. Thus, earlier works on fixed group-based categorization approaches and their corresponding addressing and routing schemes can no longer be used.

Most of the work in XML-based pub-sub research has not addressed the aspect of the routing and dissemination of messages to the destination based on the end nodes' subscriptions. Only a few studies [17, 38] have targeted this issue. The work in [17] is an extension of the author's previous research work on yFilter [16]. It uses a conventional spanning tree-based distributed broker network as the dissemination infrastructure. A major drawback of this approach is that the spanning tree-based pub-sub requires each broker along the dissemination path to run a matching evaluation algorithm upon the arrival of an event. In yFilter, this evaluation process could be expensive in terms of CPU and memory usage, which will inadvertently lead to inefficiency as the number of broker nodes increases. The research work in [38] introduced a distributed NFA, which provides functionality equivalent to that of an NFA in a centralized structure. Specifically, in this approach, an NFA is decomposed into several segments, and each segment is randomly distributed to a peer in the broker network. This approach suffers from inefficiency, as it requires an XML stream to be transferred between different peers for possibly multiple times as they navigate through the different NFA states until the final state is reached. This will impose a large communication overhead. Besides, how the matched event is disseminated to the interested parties has not been fully discussed, although, in our opinion, this is a crucial aspect to consider when designing a practical pub-sub system.

In our research work [74], we adapted the hypercube overlay approach into the tag-based system, and acquired satisfied performance with respect to the scalability and robustness. The rationale behind this adaptation is that the tag-based system can be considered as a special case of a structure-based system, hence the method proposed here can also be applied.

4.8 Summary

In this chapter, we proposed a series of novel approaches towards an Internet-scale distributed XML-based event routing architecture. By using bloom filter-based structure-aware addressing, XML documents can be organized by their structural resemblance and efficiently disseminated to subscribers through a hypercube overlay. Our method does not require flooding-alike mechanism used in conventional pub-sub systems for subscription propagation. Instead, only a small subset of nodes (members of the subhypercube for a given publication) is needed to install the filter. More importantly, the hypercube overlay organizes all of the nodes involved into a hierarchical structure so that the propagation of subscription filters can be efficiently achieved in a multicast manner. Extensive simulation results, based on realistic sample data and a typical network topology model, indicate that the proposed system scales well while achieving acceptable overhead, traffic, and stress. The results also show that the proposed architecture can balance the load over the entire network of nodes by deploying each pub-sub event to a small subset of nodes grouped by structure-resemblance based on their keys. The proposed system architecture can achieve significant speed-up and performance gains with a trade-off in a manageable false positive rate.

Chapter 5

Fault-Tolerance in Event Routing

5.1 Introduction

XML-based publish-subscribe is a promising data-centric communication paradigm for the active dissemination of structurally formatted information on an as-available basis through a "push-based" approach, leveraging the expressiveness of XML data with the effectiveness of the pub-sub paradigm. It enables loose coupling between the data source and the sink. In a dynamic environment where clients continuously join and leave, while servers may fail at times, pub-sub is able to effectively handle exchanges of data among a very large number of entities without requiring all of the information sources and sinks to be present in the network at the same time. This in turn enables the decoupling of application logic and communication, which can considerably ease the application design process. In such a system, publishers publish contents as publications in the form of XML documents. Subscribers register their interests in future publications through expressive subscriptions represented by powerful filtering languages such as XPath or XQuery, which specify complex filtering criteria by using a tree-structured model and twig pattern matching to evaluate the publications. Upon receiving a publication, the system evaluates the matching of the event to the subscriptions, which can be essentially treated as filters, and delivers the publication to the matched subscribers.

Studies on distributed XML filtering systems normally assume a spanning tree-based routing approach, which is widely used in conventional topic or content-based pub-sub systems. Normally, the tree is constructed by identifying subscriber nodes that share common subscriptions and are close in proximity to each other, while minimizing the messages traversal path to all other subscribed nodes. Using explicitly constructed spanning trees for event delivery introduces nontrivial costs (e.g., bandwidth consumption) in tree construction and maintenance, especially in dynamic systems where nodes join or leave at will. It also introduces computation overhead caused by iterative matching evaluations, which have severely limited the scalability of the system.

Peer-to-Peer (P2P) solutions based on Key-based Routing (KBR) offer efficient functionalities, such as: event routing flexibility, scalability, load balancing, and fault tolerance. Many system architectures have been proposed to exploit the advantages of KBR overlay, such as Scribe [10], Ferry [78], and Meghdoot [24]. However those systems are essentially topic-based or content-based pub-sub systems; they do not directly support XML-based pub-sub services. This motivated us to develop an XML-based pub-sub framework that can fully exploit KBR, which has been discussed in previous chapters. In this chapter we address the robustness of the proposed framework. Issues such as hypercube-to-KBR mapping and fault-tolerance [75] [76] are discussed and solutions are proposed accordingly to enrich the capability of the proposed framework. Our aim is to guarantee that the proposed system will be able to survive not only in a large-scale application, but also in a dynamic environment where nodes may join, leave and fail at times. The main contribution of this part of the research can be summarized as follows:

- We developed a robust yet scalable routing scheme for XML-based pub-sub systems. Specifically, the scheme is able to efficiently handle changes in topology caused by the joining/leaving or failure of nodes, and is also equipped with the intrinsic capability to scale up to operate in an Internet-scale environment.

- We proposed a hypercube-to-KBR mapping algorithm that can loosen the restrictions on mapping that requires the key-space being fully occupied by the nodes, hence strengthening the adaptability and robustness of the system.

- We proposed a redundancy backup strategy for subscription filters to handle the random joining, leaving, and failing of the nodes, which gives the proposed system an effective fault-tolerant capability.

5.2 Architecture Overview

We follow the same system and data model described in Chapter 4. We generalize our architecture into three layers, as shown in Figure 5.1. The major difference from the base model illustrated in Chapter 4 is that the hypercube layer now becomes a logically independent layer, separated from the generic overlay layer. This modification in architectural abstraction guarantees that the functions and properties can be fully exploited without any compromise, such as automatic formation, self-healing topology maintenance provided by KBR . It lays a foundation for the proposed extension of fault-tolerance to the original KBR-based pub-sub design.

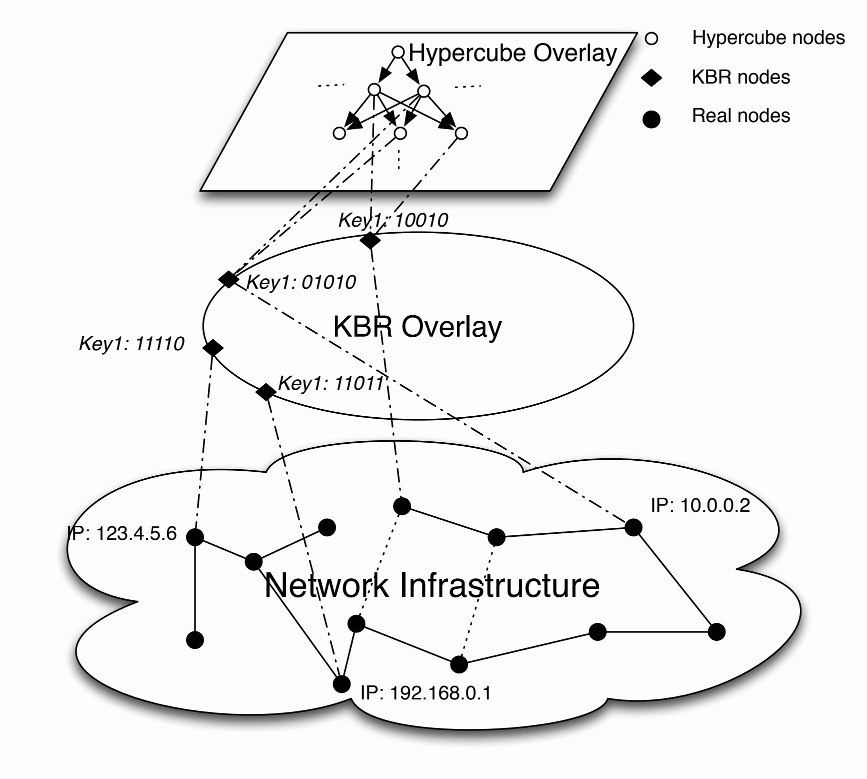


Figure 5.1: Three-layer Abstraction of the Proposed Architecture Design

In Figure 5.1, the top-most tier is the hypercube overlay layer, which provides a high-level message dissemination function. It provides an efficient multicasting infrastructure for the message propagation process required by distributed event routing. The middle layer is a generic KBR overlay layer. The implementation of this layer could vary as long as the common API mentioned in [15] is provided. This means that the proposed architecture could be used in different KBR implementations. Hence, adaptability and flexibility can be provided. The bottom layer is the abstraction of the actual network infrastructure. It could be a heterogeneous local area network or a mesh network like the Internet. The adaptability of the system is largely attributed to the overlay routing approach offered by the KBR abstraction layer.

To the best of our knowledge, the proposed architecture is the first solution that extensively exploits the KBR overlay to manage subscriptions and disseminate events for XML-based pub-sub systems. It is also the first to propose a loose mapping (from the hypercube to the structured KBR overlay) algorithm that exploits the *closest-node* mapping characteristic of KBR, leading to the development of a redundancy-based strategy for fault-tolerance. By the deep exploiting of KBR in its design, the proposed architecture has numerous advantages:

(i) The fault-tolerance and self-organizing nature of the KBR overlay, together with the proposed Hypercube-to-KBR mapping, makes the proposed architecture resilient in the case of node failures.

(ii) It does not require the flooding-alike mechanism used in spanning tree-based pub-sub systems for subscription propagation.

(iii) The proposed matching evaluation scheme exhibits high efficiency in terms of storage space and processing time.

(iv) The hypercube overlay organizes all of the nodes involved into a hierarchical structure so that the propagation of the subscription filter can be efficiently achieved in a multi-cast manner.

We implement this architecture design using OMNeT++, an extensible, modular, component-based C++ simulation library and framework. Via detailed simulations, we have evaluated the performance of the system extensively in terms of traffic, overhead, and fault-tolerance capability. The results of the extensive simulation shows that the proposed architecture can deliver events to various numbers of subscribers under different network sizes efficiently and in a timely manner with a moderate probability of the random joining, leaving, and failing of the nodes. It works successfully in 4096 nodes system that has a node's failure rate of 25% , with a replication factor of 8 (which causes link traffic overhead of only 788Bytes/Sec).

The rest of this chapter is organized as follows: First, we briefly review our hypercube-based event-dissemination approach. After that, we discuss the hypercube-to-KBR mapping that enables the logical separation of the hypercube and the KBR overlay. Following that, we proposed a redundancy-based strategy for fault-tolerance. Finally, we provide an experimental evaluation.

5.3 Hypercube-based Event Dissemination

Our previously proposed hypercube-based event dissemination approach utilizes a hypercube overlay to achieve efficient subscription installation. A publication can be delivered based on the reverse path embedded in the routing tables derived from the subscriptions installed on each intermediate node. The core process in the proposed architecture is the subscription installation process. The principle of subscription installation is to guarantee the congregation between potential matching publications and subscriptions. In section 4.3.2, a subscription propagation method KBSP is proposed for this subscription installation process, by exploiting key-based routing mechanism. The KBR is modified to support a coverage-based propagation utilizing a novel structure-aware addressing scheme to provide an event-dissemination infrastructure for the pub-sub systems. A hypercube overlay is then built on top of KBR to further improve the efficiency of the subscription propagation process. A brief summary of those techniques is presented in the following.

The normal KBR technique is built around the concept of distributed file sharing. This approach provides direct one-on-one mapping between the key and the content. However, it does not provide the capability for coverage-based propagation that requires one-to-many message dissemination, which is essential for subscription installation. We modify the original KBR technique by adopting a novel structure-aware addressing scheme, as the addressing scheme for KBR. Using structure-aware addressing, the content (XML) and filter (XPE) are mapped to the key that is essentially their corresponding bloom filter. Coverage-based propagation can be then carried out, exploiting the coverage relation implicitly represented by the bloom filter.

In structure-aware addressing, the bloom-filter of the publication and subscription is used as the key to map the content/filter in information space to the keys in key space, as opposed to the consistent hashing function used in conventional KBR. Recall that the coverage relation between two tree-structures can be essentially identified by the coverage relation between their corresponding bloom filters. Using this addressing method, the structural coverage relation between two objects in information space can be preserved in their keys in key space. For example, if a publication matches a subscription, the publication's bloom filter should also cover the subscription's bloom filter. This property can be further exploited to provide coverage-based propagation for subscription installation, which is briefly discussed in below.

Each key k is mapped to a real node N in the network in KBR. We call the node N the host of key k. Using structure-aware addressing, given an arbitrary subscription S whose key is s, we are able to find each key p_i in key space so that p_i covers s based on the coverage relation defined by the bloom filter. Therefore, we can treat p_i's host as a rendezvous node (RN for short). In this way, we are able to guarantee that all of the potential publications for S will eventually meet S, since the publication forwarded to RN will always be a potential publication for S due to the coverage relation between their corresponding bloom filters (keys).

The aforementioned subscription installation requires the propagation of subscriptions to the corresponding rendezvous nodes (p_i's host). This process requires a large number of messages to be forwarded to multiple destinations; hence, the multicasting style of propagation is preferred to unicasting message delivery to minimize the utilization of bandwidth. According to section 4.4, nodes are organized into an ordered-hypercube based on their keys. The covering relation between keys of different nodes is maintained as a parent-child relation in an ordered-hypercube. With the hypercube structure, given any node with key k, every node whose key is covered by k can be reached by traversing through a subhypercube rooted in k. This traversal process can later be optimized into a tree traversal process (section 4.4.3). In this way, the propagation can be achieved in a multicast manner.

5.4 Hypercube-to-KBR Mapping

As we have mentioned in Chapter 2.4, conventional KBR use consistent hashing as the addressing scheme to achieve the mapping between node/data and key. A flexible content mapping method is adopted to guarantee that the joining or leaving of certain nodes will cause only minimum disruption. Specifically, the content is mapped to the node whose key is closest to that of the content. By doing this, when a node n joins the network, certain content that previously mapped to n's successor now become mapped to n. When the node leaves the networks, all of its mapped contents are reassigned to n's successor. No other changes in assignment of keys to nodes need occur. Based on this feature, a replication strategy can be deployed to enable fault-tolerance. In our basic system model mentioned in Chapter 4, the hypercube topology implies that a key in hypercube has to be occupied by certain nodes to guarantee the completeness of the hypercube topology. Hence, a direct map between a key in hypercube and an actual node, like the one adopted in conventional KBR, may render useless all of the above-mentioned advantages of flexible content mapping, as the content has to be mapped to the exact key in order to make use of the hypercube overlay. However, we are able to adjust the mapping mechanism between the hypercube key and the actual node to preserve the advantages provided by consistent hashing, as explained briefly below.

First we introduce the concept of *virtual node*, which is a logical abstraction of a node entity. Each virtual node has a unique key identifier chosen from the *key space*. The term *physical node* represents the real node in the network. The key for the physical node is chosen by an algorithm defined by KBR. The virtual node resides within the physical node, and must do so to be able to function. A physical node is able to host multiple virtual nodes if needed. Next, we map the hypercube to the virtual node space using strict one-on-one mapping based on keys. After this mapping, a complete hypercube will be established with each vertex representing a virtual node. Lastly, we map the virtual nodes to a physical node, based on the identical flexible mapping rule to that for content mapping in conventional KBR - Each virtual node is mapped to the physical node whose key is closest to the key of this virtual node. Hence, after these three steps, the hypercube topology overlay is mapped to the KBR overlay without any compromise in functionality.

In the implementation, we use a data-structure called a *subscription list* to realize the above concept. For each node, we maintain one or a few subscription lists, with each list represents a *subscription filter table* for a virtual node. In the message, all we need to do is to append a *virtual key field* to tell the receiver from which virtual node the message has been sent. When the message needs to be sent to a certain key k, it is guaranteed to be sent to the virtual node that is responsible for k, even when the physical node that has key k is missing (the virtual node in question will reside in the physical node whose key is closest to k). Upon a change in topology (nodes joining or leaving), all we need to do is to hand over the corresponding subscription list to the node that is going to be responsible for it. It is possible that the virtual nodes within one physical node will communicate with each other. This can be treated as self-messaging. Self-messaging can be further tuned to avoid the consumption of real bandwidth by handling the messages internally on the object level in memory.

5.5 Redundancy-based Fault-tolerance Strategy

In modern decentralized data-centric architecture, fault-tolerance is a very important aspect of handling the dynamicity of the environment. Fault-tolerance is normally achieved by backing up the data storage for each node in KBR using appropriate replication strategies.

In the common abstraction of the KBR scheme [15], nodes are allowed to join and leave the system at will, causing churning in the set of nodes in the system. A structured topological overlay (such as a ring in Chord) regularly runs maintenance algorithms that detect failures and re-pair routing tables, allowing requests for a key to be routed correctly to their owner despite node churn. However, KBR can only provide fault-tolerance in the routing level, while content-storage is not fault-tolerant. When a node fails, the information (subscription list) that it carries becomes inaccessible, and must be recovered from elsewhere. This means that to provide reliable subscription installation, a replication algorithm must store and maintain backup copies of the subscription list. This must be achieved in a cost-effective way without compromising the scalability of the system and also must be implemented in a decentralized manner without causing unbalanced loads. Based on these requirements we propose the following replication algorithm, which exploits the aforementioned hypercube-to-KBR mapping scheme.

Replicas of a virtual node (referred to as an item for the sake of clarity) are placed only on the r closest physical nodes of the node responsible for that item's key. Those nodes are named siblings, and r is known as the replication factors. This placement is mainly due to two reasons: 1) When a node becomes unavailable, according to KBR, its successor will automatically become responsible for all the virtual nodes it hosts. Hence, it is quite natural to send the replica to its successor. 2) When the node and its successor both fail, it is necessary to have a replica placed on more than one of the nodes closest to the hosting node, thus we use r replicas at r closest nodes instead of only one. This will significantly reduce the probability of losing virtual node information.

To maintain this placement policy in case of node churn, a replica-maintenance protocol will be triggered under certain circumstances (discussed later). It also prevents the number of replicas of any object from either dropping too low or rising too high. This maintenance protocol can be summarized as follows:

1. Upon being notified by the joining of a new predecessor, a node checks the keys of the items (virtual nodes) that it is hosting to see if they are storing any item for which they are no longer responsible, and sending the found items to new host (the predecessor). Goto step 3.

2. Upon being notified by the leaving of the predecessor (also including the failure of the predecessor), a node searches in its replica for the items that its predecessor used to be responsible for and starts to host them. Goto step 3.

3. The node send a replica of items it is currently hosting to r siblings of the node.

4. Upon receiving a replica, a node synchronizes the replica that it kept with the replica that it received from other nodes.

In our implementation, the replica is essentially a copy of the subscription list. We added a flag *active* in the subscription data structure to determine whether it is currently a working subscription list or just a replica. Besides the above replica-maintenance, we utilized the periodic maintenance event provided by the KBR. In this periodic maintenance procedure, a node checks if it is still the sibling (*rth* closest nodes) of each key for replica, if not, it will delete the entry of the replica accordingly.

Replica-maintenance will be triggered in three circumstances:

- **Upon an update event notified by underlying KBR**. The update event, provided as a common API function for the KBR, will be triggered if any changes have happened to the predecessor of the node (joining or missing of the predecessor).

- **Upon an incoming subscription installation update.** Once the subscription list is updated on a node, it needs to be forwarded to all of the siblings of this node in order for them to keep an updated version of its replica.

- **Upon the receiving of a replica.** The synchronization required in the maintenance protocol needs to be conducted, once the node has received a replica from another node.

Assume any node can fail with a probability of p. In a network of n nodes, and a replication factor of r, with this replication policy, the subscription will only be lost when all r sibling fail. Hence, the probability of subscription loss can be estimated by:

$$p_{Loss}(p, r, n) = 1 - (1 - p^r)^{n/r} \tag{5.1}$$

5.6 Performance Study

In this section, we present the performance evaluation result of the proposed system architecture. We used OMNeT++ [62] as our main simulation platform. The INET Framework [60] and OverSim [64] were adopted to facilitate the simulation design. We mainly used Chord as the KBR in our experiments, but the system can be easily ported to other KBR implementations, as long as common KBR APIs [15] are provided. A real-world XML data set from [13] was used in our simulation, with 3,680 being the total number of XML documents, conforming to nine distinctive DTDs (refer to Table 3.1 for more details). The parameter of simulation setup can be found in Table 4.1.

5.6.1 Link Traffic

In this section, we present the results of our evaluation on per-link bandwidth consumption. In Figure 5.2 by varying the length of the subscription filter, denoted by l in the following, the system was evaluated in two cases: l=128, and l=1024. The hash function number for the bloom filter was set to 1. It is observable that for both the l=128 and l=1024 scenarios, as the number of participating nodes increased, the link traffic showed a tapering trend, indicating that the system has good scalability. This is largely attributed to the underlying load balancing mechanism provided by the hypercube overlay. It can also be noticed that increasing the length of the subscription filter can largely reduce the link traffic, especially for setups of 16, 64, and, 256 nodes. This is because increasing the length of the filter improves the precision of the filter so that more false matches are eliminated. In Figure 5.3, we focused on the traffic caused by the replication mechanism. By setting the filter length l to 1024 and varying the replication factor, we additionally record the total link traffic in three scenarios, where r= 1, r= 2 and r=8. The figure shows that the increase in the number of replicas will only cause a mild increase in total link traffic -

788Bytes/Sec in the worst case when r=*8* at the 4096 node setup (compare to r=1 case). This is crucial in a large-scale system, as when the system scales the replication should not impose an excessive overhead, which could potentially overload the system.

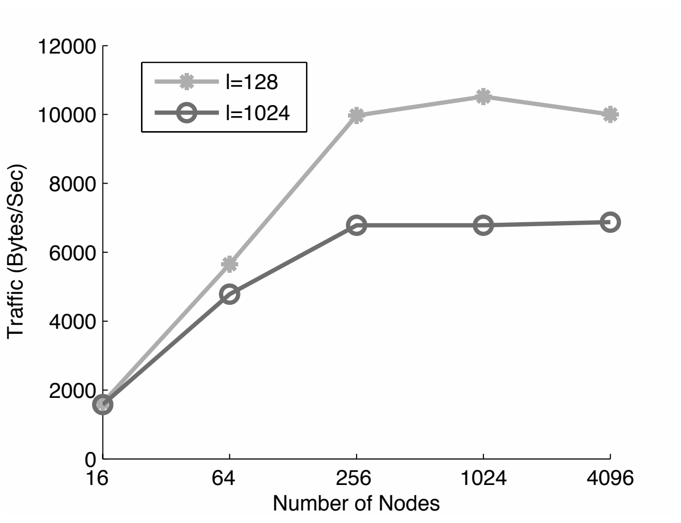


Figure 5.2: Link Traffic: Filter Length

5.6.2 Publication Delivery

To study the behavior of publication delivery, we designed two experiments: one focused the impact of filter length as the number of nodes increases, the other focused on the effectiveness of the replication mechanism when node failures occur.

First, we evaluated the impact of filter length. By varying the filter length l, we recorded the total number of publications received by every node involved in the system. The results are shown in the Figure 5.4. It is important to note that as the subscription filter length l increases,

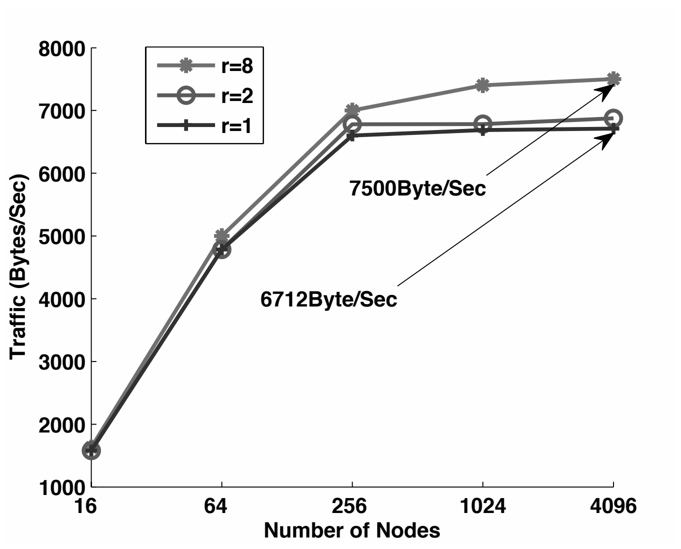


Figure 5.3: Link Traffic: Replication Factor

the number of publication messages received by peers will decrease. This means that a longer subscription filter can filter out more false publication messages.

In Figure 5.5, to study the effectiveness of the proposed replication method in handling node failures, we intentionally increased the probability of failure for each node p (p=0, p=0.25, and p=0.5), and then recorded the publication delivery rate in a 4096-node setup, which was calculated by:

$$DeliveryRatio = PubReceived_{with-node-failure}/PubReceived_{no-node-failure} \tag{5.2}$$

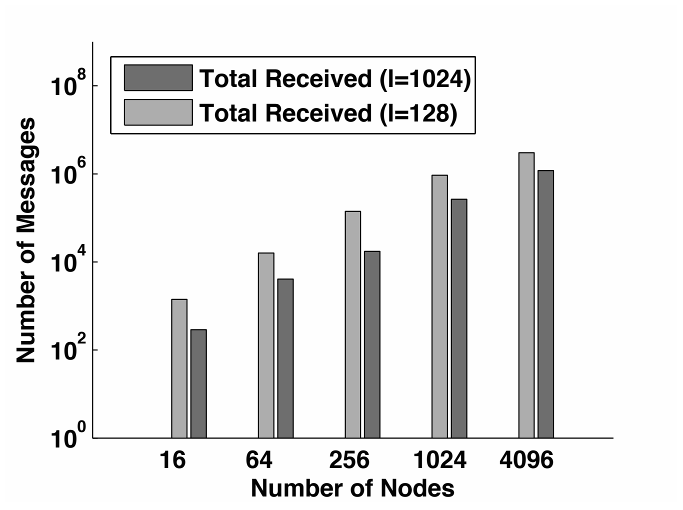


Figure 5.4: Total Number of Publications Received

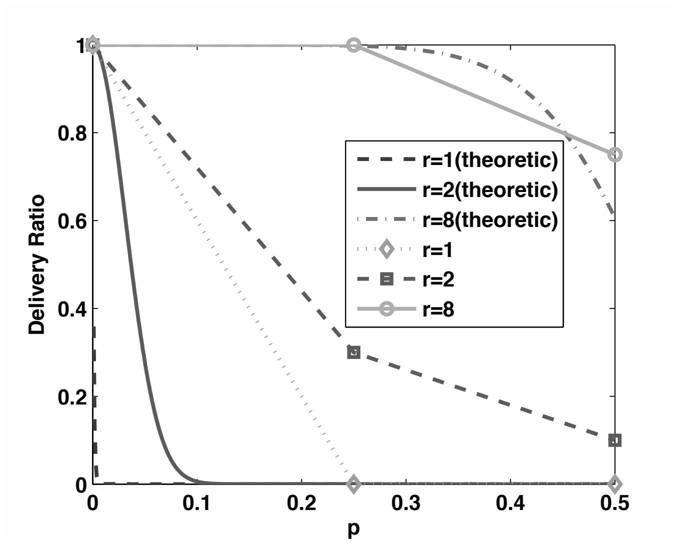


Figure 5.5: Delivery Ratio

The drop in the delivery rate indicates that there will be some subscription loss in the system. As a comparison, we provide a theoretic estimation based on equation 5.1. We can see in Figure 5.5 that, in line with the theoretic estimation, the delivery ratio drops as the probability of node failure increases. However, with a slight increase in *r,* this reduction can be significantly reduced. It is important to note that by increasing the replication factor to 8, we can achieve a 100% delivery rate at a node failure probability of 25%. Even when the nodes failure probability reaches 50%, only 20% of publications will be lost. Considering that there will be only a 788Bytes/Sec overhead in terms of link traffic produced by replication with r=8, this result is quite promising.

5.7 Related Work

In our previous research work [72] and [73], we introduced a series of novel approaches that contribute toward enabling an efficient, distributed XML-based pub-sub system design that scales up to operate in a wide-area and large-scale environment. In these works, a bloom filter-based filtering approach was coalesced into the addressing scheme for the Key-based Routing to provide a scalable, flexible, and robust pub-sub infrastructure. A hypercube overlay was later adopted as multicasting infrastructure for the efficient dissemination of publications. With that design, the hypercube overlay organizes all of the nodes involved into a hierarchical structure so that the propagation of the subscription filter can be efficiently achieved in a multi-cast manner. In this chapter of the thesis, we present our work in extending the hypercube-based XML dissemination architecture to address the impact of dynamicity of the system, by introducing a novel hypercube-to-KBR mapping algorithm, and a redundancy-based fault-tolerance strategy towards a fault-tolerant system design. The result is a full-fledged XML-based pub-sub system, which works well not only in a large-scale application, but also in a dynamic environment where nodes may join, leave, and fail randomly.

5.8 Summary

In this chapter, we specifically addressed the impact of the dynamicity of distributed pub-sub systems. By introducing a novel hypercube-to-KBR mapping algorithm, and a redundancy-based fault-tolerance strategy, we are able to extend our previously proposed event routing architecture to a robust pub/sub platform which aquires excellent fault-tolerance capability. Extensive experiments revealed that the proposed architecture works well not only in a large-scale application, but also in a dynamic environment where nodes may join, leave and fail at times.

Chapter 6

Conclusion

Current XML-based pub-sub systems are either centralized or distributed. Centralized solutions, while simple, have an inherent scalability problem as the number of events and subscriptions in the system increases. It is also prone to the single point of failure. Hence, many studies have proposed the use of distributed architecture, which implement publication matching and delivery in a distributed manner.

Studies on distributed XML filtering systems normally assume a spanning tree-based routing approach, which is widely used in conventional topic or content-based pub-sub systems. Normally, the tree is constructed by identifying subscriber nodes that share common subscriptions and are close in proximity to each other, while minimizing the messages traversal path to all other subscribed nodes. Using explicitly constructed spanning trees for event delivery introduces nontrivial costs (e.g., bandwidth consumption) in tree construction and maintenance, especially in dynamic systems where nodes join or leave at will. It also introduces computation overhead caused by iterative matching evaluations, which severely limits the scalability of the system. Peer-to-Peer (P2P) solutions based on Key-based Routing (KBR) offer efficient functionalities such as event routing flexibility, scalability, load balancing, and fault tolerance. Many system architectures have been proposed to exploit the advantages of KBR overlay, such as Scribe [10], Ferry [78], and Meghdoot [24]. However, those systems are essentially topic-based or content-based pub-sub systems, which do not directly support XML-based pub-sub services. This has motivated us to develop an XML-based pub-sub framework that can fully exploit KBR.

6.1 Research Results

Based on our study, an event-based architecture in general has two major function layers: event matching and event routing. Hence, in our study, we incrementally propose various approaches to tackling different issues with respect to event matching and event routing. This research covers many aspects: from structure-based event representation and event matching evaluations to structure-aware addressing and distributed event routing. With our systematic research, we managed to develop an architectural framework that can efficiently handle XML-based structural documents in a fully distributed large-scale environment. We did so by adopting a holistic approach to considering filtering and routing as two processes that may synergistically complement one another to support efficient message dissemination. The framework is essentially comprised of two core contributions: the bloom filter-based XML filtering scheme and the KBR-based distributed event-routing scheme. Both have been thoroughly evaluated through extensive simulation experiments.

Efficient Event Matching In the area of event matching, we adopted a highly efficient method of representing structure – one that is highly compact and optimized for evaluation using a bloom filter. Judging from the results of several evaluations carried out by extensive experiments, our method shows excellent matching speed and compact storage size. It also shows good stability and scalability because it is not affected by an increasing number of documents/filters once the system parameter is fixed. Even though it introduces a false positive error, the false positive error can be bounded and adjusted to match the requirements of applications. Considering the significant speed-up and performance gains, our approach represents a viable solution when time/space efficiency is paramount.

Scalable Event Routing In the area of event routing, we first established an Internet-scale distributed XML-based pub-sub architecture. By using bloom filter-based structure-aware addressing, XML documents can be organized by their structural resemblance and efficiently disseminated to subscribers through hypercube overlay. The proposed method does not require flooding-alike mechanism used in conventional pub-sub systems for subscription propagation. Instead, only a small subset of nodes (members of the subhypercube for a given publication) is needed to install the filter. More importantly, the hypercube overlay organizes all of the nodes involved into a hierarchical structure so that the propagation of the subscription filter can be efficiently achieved in a multicast manner. Extensive simulation results, based on realistic sample data and a typical network topology model, indicate that the proposed system scales well while achieving acceptable overhead, traffic, and stress. The results also show that the proposed architecture can balance the load over the entire network of nodes by deploying each pub-sub event to a small subset of nodes grouped by structure-resemblance based on their keys. The proposed system architecture can achieve significant speed-up and performance gains with a trade-off of manageable false positive rate.

Fault-tolerance in Event Routing To further enhance proposed event-dissemination architecture, we carried out a series of studies addressing the robustness of the architecture considering the dynamism of the underlying network environment. By introducing a novel hypercube-to-KBR mapping algorithm, and a redundancy-based backup strategy, we are able to provide a robust publication dissemination architecture with fault-tolerance capability. Extensive experiments have revealed that the proposed architecture works well not only in a large-scale application, but also in a dynamic environment where nodes may join, leave, and fail at times.

6.2 Future Works

Obviously, a number of issues have not been discussed in detail in this thesis and should be the subject of further research.

- The general topic of caching and event histories in a distributed publish-subscribe notification service opens up a wide area of opportunities for research. Event caching can make use of the historical messages at each intermediate broker, so that the new subscriptions will be able to retrieve those historical messages. At the same time, event caching enables the asynchronized communication to take place between brokers and clients. For instance, when a subscriber is offline, all new messages will be cached for it, and when it goes online, those messages can be automatically delivered to it, as if the subscriber had never gone offline. This caching mechanism further decouples the communication parties, making the system more flexible and robust. However, in this research we have not yet covered this topic.

- Another important issue is predicate-based matching. XPath supports predicate matching based on the element value in a XML document. Hence, the ranged query and linear path query can both be adopted as the filter expression for an XML-based pub-sub system. In this paper, for the sake of clarity, we focus on the structural aspect of an XML document, while ignoring the value-based predicate. The focus on structural matching ignoring value-based predicates may lead to inefficiency, since XPath expressions with only structural component (ignoring values) will be less selective and more “unwanted” XML documents may be selected.

- Real-world performance evaluations are another important area to focus on in our future work. The simulation-oriented evaluation approach adopted in this research has provided us many significant results. The controlled environment that the simulation platform provides is crucial for the initial stage of research, as it can not only speed up the implementation of the prototype system, but also shortening the evaluation cycle of experimentation tremendously. However, the real-world test of proposed system design is irreplaceable when evaluating the effectiveness and efficiency of a system design in reality. Since the proposed experiment evaluation is largely based on a simulation platform, the immediate next step is to implement the proposed architecture in a real-world test platform, such as PlanetLab [63].

6.3 Summary

In summary, this thesis presents several significant and novel approaches for empowering publish-subscribe with XML-based filtering capability, leading to effective, efficient, and convenient ways of communicating and interacting in large-scale pervasive computing environments. We analyzed the shortcomings of existing approaches, and the implications and requirements for large-scale distributed pub-sub applications. Subsequently, we showed the importance of efficiency, scalability, robustness, autonomy, and level of integration in a distributed XML-based pub-sub system design. The distributed publish-subscribe architecture presented in this thesis constitutes a noteworthy platform for building pervasive and ubiquitous computing systems. More importantly, it opens many new directions in research towards integrating and synergizing the expressiveness of XML data with the effectiveness of the pub-sub paradigm.

Chapter 7

Appendices

7.1 Appendix A

Estimation of False Positive Errors in Bloom Filter

Assume we have a target set A and a test set B.

The ideal hash function will produce a value that lies within the range of the bit-vector length with uniform distribution. Hence, each bit position will have the equal probability to be selected by a hash fucntion. Let p_0 be the probability that a bit is not set to 1 by a given hash function, we have:

$$p_0 = 1 - 1/m \tag{7.1}$$

Then for k hash function, if it is still not set to 1, the probability will be $(p_0)^k$, and for all n elements, the probability that it is still not set will be $(p_0)^{nk}$. The probability that it is set to 1 will be $1 - (p_0)^{nk}$.

Now assume an element $b \in B$, but $b \notin A$. Each bit in k bit position yielded by applying k hash functions on b are set to 1 with the probability $1 - (p_0)^{nk}$, given by previous analysis. Then for all k bit to be set to 1, which will produce a false positive, we have probability of false positive P:

$$P = (1 - (p_0)^{nk})^k = (1 - (1 - 1/m)^{nk})^k \approx (1 - e^{-kn/m})^k \tag{7.2}$$

Bibliography

[1] Ioannis Aekaterinidis and Peter Triantafillou. PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ICDCS '06, page 23. IEEE Computer Society, 2006.

[2] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, PODC '99, pages 53–61. ACM, 1999.

[3] Marc Sanchez Artigas, Pedro Garcia Lopez, Jordi Pujol Ahullo, and Antonio F. Gomez Skarmeta. Cyclone: A Novel Design Schema for Hierarchical DHTs. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, P2P '05, pages 49–56. IEEE Computer Society, 2005.

[4] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings*

of the 22th International Conference on Very Large Data Bases, VLDB '96, pages 28–39. Morgan Kaufmann Publishers Inc., 1996.

[5] Kenneth P. Birman. The process group approach to reliable distributed computing. *Commun. ACM*, 36(12):37–53, December 1993.

[6] Twitter Blog. What's Happening with Twitter? Technical report.

[7] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[8] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of Bloom filters. *Inf. Process. Lett.*, 108(4):210–213, October 2008.

[9] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, August 2001.

[10] M. Castro, P. Druschel, A. M. Kermarrec, and A. I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE J.Sel. A. Commun.*, 20(8):1489–1499, September 2006.

[11] Amina Chaabane, Wassef Louati, Mohamed Jmaiel, Jorge Gomez-Montalvo, Code Diop, and Ernesto Exposito. Towards an ontology and DHT-based publish/subscribe scalable system. In *IEEE International Conference on Communications*, ICC '12, pages 6499–6503. IEEE Computer Society, 2012.

[12] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *The VLDB Journal*, 11(4):354–379, December 2002.

[13] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 379–390. ACM, 2000.

[14] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a Bloom filter. *Inf. Process. Lett.*, 110(21):944–949, October 2010.

[15] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common API for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, pages 33–44. Springer, 2003.

[16] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, and Peter Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 28(4):467–516, December 2003.

[17] Yanlei Diao, Shariq Rizvi, and Michael J. Franklin. Towards an Internet-scale XML dissemination service. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 612–623. VLDB Endowment, 2004.

[18] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of*

the 2001 ACM SIGMOD international conference on Management of data, SIGMOD '01, pages 115–126. ACM, 2001.

[19] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.

[20] Prasanna Ganesan, Krishna Gummadi, and Hector Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ICDCS '04, pages 263–272. IEEE Computer Society, 2004.

[21] Xueqing Gong, Weining Qian, Ying Yan, and Aoying Zhou. Bloom Filter-Based XML Packets Filtering for Millions of Path Queries. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 890–901. IEEE Computer Society, 2005.

[22] Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.*, 29(4):752–788, December 2004.

[23] Todd J. Green, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML Streams with Deterministic Automata. In *Proceedings of the 9th International Conference on Database Theory*, ICDT '03, pages 173–189. Springer-Verlag, 2002.

[24] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: content-based publish/subscribe over P2P networks.

In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, pages 254–273. Springer-Verlag New York, Inc., 2004.

[25] Ashish Kumar Gupta, Dan Suciu, and Alon Y. Halevy. The view selection problem for XML content based routing. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, pages 68–77. ACM, 2003.

[26] Annika Hinze, Yann Michel, and Torsten Schlieder. Approximative filtering of XML documents in a publish/subscribe system. In *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ACSC '06, pages 177–185. Australian Computer Society, Inc., 2006.

[27] Shuang Hou and H.-Arno Jacobsen. Predicate-based Filtering of XPath Expressions. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, page 53. IEEE Computer Society, 2006.

[28] IBM. Gryphon: Publish/Subscribe over public networks. Technical report, IBM T. J. Watson Research Center, 2001.

[29] Hojjat Jafarpour, Bijit Hore, Sharad Mehrotra, and Nalini Venkatasubramanian. CCD: A Distributed Publish/Subscribe Framework for Rich Content Formats. *IEEE Trans. Parallel Distrib. Syst.*, 23(5):844–852, May 2012.

[30] Clement Jamard, Georges Gardarin, and Laurent Yeh. Indexing textual XML in P2P networks using distributed bloom filters. In *Pro-*

ceedings of the 12th international conference on Database systems for advanced applications, DASFAA'07, pages 1007–1012. Springer-Verlag, 2007.

[31] Satyen Kale, Elad Hazan, Fengyun Cao, and Jaswinder Pal Singh. Analysis and Algorithms for Content-Based Event Matching. In *Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05) - Volume 04*, ICDCSW '05, pages 363–369. IEEE Computer Society, 2005.

[32] Joonho Kwon, Praveen Rao, Bongki Moon, and Sukho Lee. FiST: scalable XML document filtering by sequencing twig patterns. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 217–228. VLDB Endowment, 2005.

[33] Joonho Kwon, Praveen Rao, Bongki Moon, and Sukho Lee. Value-based predicate filtering of XML documents. *Data Knowl. Eng.*, 67(1):51–73, October 2008.

[34] Joonho Kwon, Praveen Rao, Bongki Moon, and Sukho Lee. Fast XML document filtering by sequencing twig patterns. *ACM Trans. Internet Technol.*, 9(4):13:1–13:51, October 2009.

[35] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65. Springer-Verlag, 2002.

[36] Iris Miliaraki, Zoi Kaoudi, and Manolis Koubarakis. Xml data dissemination using automata on top of structured overlay networks. In

Proceedings of the 17th international conference on World Wide Web, WWW '08, pages 865–874. ACM, 2008.

[37] Iris Miliaraki and Manolis Koubarakis. Distributed structural and value XML filtering. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 2–13. ACM, 2010.

[38] Iris Miliaraki and Manolis Koubarakis. FoXtrot: Distributed structural and value XML filtering. *ACM Trans. Web*, 6(3):12:1–12:34, October 2012.

[39] Tova Milo, Tal Zur, and Elad Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 749–760. ACM, 2007.

[40] Alan Mislove and Peter Druschel. Providing administrative control and autonomy in structured peer-to-peer overlays. In *Proceedings of the Third international conference on Peer-to-Peer Systems*, IPTPS'04, pages 162–172. Springer-Verlag, 2004.

[41] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer, 1st edition, July 2006.

[42] James K. Mullin. A second look at bloom filters. *Commun. ACM*, 26(8):570–571, August 1983.

[43] Yuan Ni and Chee Yong Chan. Dissemination of heterogeneous XML data in publish/subscibe systems. In *Proceedings of the 18th ACM con-*

ference on Information and knowledge management, CIKM '09, pages 127–136. ACM, 2009.

[44] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus: an architecture for extensible distributed systems. In *Proceedings of the fourteenth ACM symposium on Operating systems principles*, SOSP '93, pages 58–68. ACM, 1993.

[45] Makoto Onizuka. Light-weight xPath processing of XML stream with deterministic automata. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 342–349. ACM, 2003.

[46] Flutra Osmani, Victor Grishchenko, Raul Jimenez, and Björn Knutsson. Swift: the missing link between peer-to-peer and information-centric networks. In *Proceedings of the First Workshop on P2P and Dependability*, P2P-Dep '12, pages 4:1–4:6. ACM, 2012.

[47] Pingdom. Report: Twitter Growing Pains Cause Lots of Downtime in 2007. Technical report, 2007.

[48] David Powell. Group communication. *Commun. ACM*, 39(4):50–53, April 1996.

[49] Praveen Rao and Bongki Moon. PRIX: Indexing And Querying XML Using Prüfer Sequences. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, page 288. IEEE Computer Society, 2004.

[50] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings*

of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '01, pages 161–172. ACM, 2001.

[51] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350. Springer-Verlag, 2001.

[52] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, September 1997.

[53] Wenting Si, Xiaoping Xue, and Xiaoping Wang. An ontology-based event matching dealing with semantic heterogeneity in Pub/Sub systems. In *Proceedings of the 4th International Conference on Computer Science & Education*, ICCSE '09, pages 1225–1230, July 2009.

[54] Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. XML-document-filtering automaton. *Proc. VLDB Endow.*, 1(2):1666–1671, August 2008.

[55] Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. Schema-conscious filtering of XML documents. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 970–981. ACM, 2009.

[56] G. D. Stamoulis and J. N. Tsitsiklis. Efficient Routing Schemes for Multiple Broadcasts in Hypercubes. *IEEE Trans. Parallel Distrib. Syst.*, 4(7):725–739, July 1993.

[57] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, February 2003.

[58] Feng Tian, Berthold Reinwald, Hamid Pirahesh, Tobias Mayr, and Jussi Myllymaki. Implementing a scalable XML publish/subscribe system using relational database systems. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 479–490. ACM, 2004.

[59] D Tran. Publish/Subscribe Service in CAN-based P2P Networks: Dimension Mismatch and The Random Projection Approach. In *Proceedings of 17th International Conference on Computer Communications and Networks*, ICCCN '08, pages 1 – 8, August 2008.

[60] Webpage. INET Framework. `http://inet.omnetpp.org[Acccessed:2013-07-12]`.

[61] Webpage. MATLAB - The Language of Technical Computing. `http://www.mathworks.com/products/matlab/[Acccessed:2013-07-12]`.

[62] Webpage. OMNeT++ Network Simulation Framework. `http://www.omnetpp.org[Acccessed:2013-07-12]`.

[63] Webpage. PlanetLab. `http://planet-lab.org[Acccessed:2013-07-12]`.

[64] Webpage. The OverSim P2P Simulator. `http://www.oversim.org[Acccessed:2013-07-12]`.

[65] Webpage. Twitter. `http://www.twitter.com[Acccessed:2013-07-12]`.

[66] Webpage. XML Path Language (XPath). `http://www.w3.org/TR/xpath/[Acccessed:2013-07-12]`.

[67] Webpage. XQuery 1.0: An XML Query Language (Second Edition). `http://www.w3.org/TR/xquery/[Acccessed:2013-07-12]`.

[68] Gang Xu, Wei Xu, and Tao Huang. An Extended Event Matching Approach in Content-based Pub/Sub Systems for EAI. In *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, EDOC '05, pages 287–296. IEEE Computer Society, 2005.

[69] Gang Xu, Wei Xu, and Tao Huang. Extending OBDD Graphs for Composite Event Matching in Content-Based Pub/Sub Systems. In *Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science*, ICIS '05, pages 519–525. IEEE Computer Society, 2005.

[70] Zhiyong Xu, Rui Min, and Yiming Hu;. HIERAS: a DHT based hierarchical P2P routing algorithm. In *Proceedings of the International Conference on Parallel Processing*, pages 187 – 194, October 2003.

[71] Xiaoyu Yang and Yiming Hu. A DHT-based Infrastructure for Content-based Publish/Subscribe Services. In *Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing*, P2P '07, pages 185–192. IEEE Computer Society, 2007.

[72] Xiaochuan Yu and Toong Shoon Alvin Chan. A Time/Space Efficient XML Filtering System for Mobile Environment. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01*, MDM '11, pages 184–193. IEEE Computer Society, June 2011.

[73] Xiaochuan Yu and Toong Shoon Alvin Chan. A Hypercubic Event-dissemination Overlay Using Structure-aware Addressing for Distributed XML-based Pub/sub System. In *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, HPCC '12, pages 179–186. IEEE Computer Society, June 2012.

[74] Xiaochuan Yu and Toong Shoon Alvin Chan. A hypercubic overlay using bloom-filter based addressing for a non-dedicated distributed tag-based pub/sub system. To appear in *Proceedings of the 11th IEEE International Symposium on Parallel and Distributed Processing with Applications*, ISPA'13, July 2013.

[75] Xiaochuan Yu and Toong Shoon Alvin Chan. Hope: A fault-tolerant distributed Pub/Sub architecture for large-scale dynamic network environment. To appear in *Proceedings of the 12th IEEE International Conference on Ubiquitous Computing and Communications*, IUCC'13, July 2013.

[76] Xiaochuan Yu and Toong Shoon Alvin Chan. Towards robust XML dissemination in large-scale dynamic network environment. Submitted to *Journal of Computer and System Sciences*, 2013.

[77] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, University of California at Berkeley, 2001.

[78] Yingwu Zhu and Yiming Hu. Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services. *IEEE Trans. Parallel Distrib. Syst.*, 18(5):672–685, May 2007.