XEvent: An Event Notification System over Distributed Hash Table (DHT) Networks

Ruili Wang, Weixiong Rao and Chengqi Zhang

Abstract—**In this paper, we propose a novel event notification system, named as XEvent. The system is built over a Distributed Hash Table (DHT)-based Peer-to-Peer (P2P) system by combining a content-based system with a specific event topic. XEvent can support basic topic-based message subscription by use of XML schema as event topic, and further filter the whole message contents using XPath as the filtering language. The unique features of XEvent include: (1) XEvent inherits the excellent features provided by DHT, including scalability, adaptation and self-maintenance; (2) XEvent provides the expressive topic-based and content-based event filtering; (3) XEvent can filter and deliver event messages to subscribers with high efficiency by building a delivery tree based on the subscriber's XPath filters, and (4) XEvent can provide fault-tolerance for node failure.**

According to the results of simulation and tests on our XEvent prototype built over a new Bamboo-DHT system, XEvent can achieve the scalability, expressiveness, high efficiency and reliability for the event notification service.

Index Terms— **Distributed algorithms, Computer networks, Query languages, Message systems**

I. Introduction

An event notification service can be implemented by a centralized event broker or by a network of distributed event brokers. The first case, every client application can act as a publisher, a subscriber, or both, and connect to a single central broker server, which acts simply as a message filtering and routing engine. Obviously, this centralized manner, lack of scalability, results in the problem of single point of failure. The second case, multiple servers can act as event brokers and are cooperatively to form a distributed, coherent message routing, matching and filtering engine for clients. This distributed manner can be implemented on a wide-area network (WAN) and be scalable to a large number of clients.

Recently, a new event filtering model, called a content-based event filtering model [1, 2], allows subscriptions to evaluate the entire content of an event message, not just a traditional event topic. The content-based event system provides a more powerful and flexible event filtering than traditional topic-based event systems [1, 2]. However, what makes content-based event system challenging is that the system must be scalable to a large number of subscribers, publishers and event messages. Also, at the same time, the system must support a flexible and expressive event filtering mechanism, i.e. scalability and flexibility [1].

Distributed Hash Table (DHT) [3, 4, 5] provides unique features for a large scalable distributed system: scalability, adaptation and self-maintenance. Thus, the distributed applications built over DHT can inherit such features provided by DHT including completely decentralized, scalable, and self-organizing; and maintain guaranteed routing efficiency under the joining, departure and failure of nodes in a network. Although some event notification systems [9, 10] built over DHT systems [4, 5] have already been implemented, such systems are topic-based and cannot provide flexibility and expressiveness as they lack a content-based filtering mechanism.

The emergence of XML (eXtensible Markup Language) as a standard for inform4ation exchange on the Internet has led to an increased interest in content-based publish/subscribe system [6, 7, 8]. Using XML as the message format can allow structural information within message documents. Using XPath as an XML query language can provide the filtering on both the structure and the contents of published XML information. Although some XML-based filtering systems like [6, 7, 8] can provide a flexible message filtering and matching mechanism, such systems are all a centralized solution. They lack scalability and cannot be extended to a distributed environment.

In this paper, we propose a novel event notification system, named as XEvent. The system is built over a Peer-to-Peer (P2P) DHT networking by combining a content-based system with a specific event topic. XEvent can support basic topic-based message subscription by use of XML schema for event topic, and further filter the entire message contents using XPath as the filtering language. The unique features of XEvent include: (1) XEvent inherits the excellent features provided by DHT, including scalability, adaptation and self-maintenance; (2) XEvent provides the expressive topic-based and content-based event filtering; (3) XEvent can filter and deliver event messages to subscribers with high efficiency by building a delivery tree based on the subscriber's XPath filters, and (4) XEvent can provide fault-tolerance for node failure. According to the results of simulation and tests on our XEvent prototype built over a new Bamboo-DHT system [14] (developed by Berkeley), XEvent can achieve the scalability, expressiveness and high efficiency and reliability for the type-based event notification service.

R. Wang is with the Institute of Information Sciences and Technology, Massey University, New Zealand, Private Bag 11222, Palmerston North, New Zealand. (corresponding author's telephone: 0064-6-3569099 ext 2548; fax: 0064-6-3502259 e-mail: r.wang@massey.ac.nz).

W. Rao is with the Information System and Service Department, Shanghai General Motor, China 201102. (e-mail: rweixiong@yahoo.com.cn).

C. Zhang is with the Faculty of Information Technology, University of Technology, Sydney, Broadway NSW 2007, Australia. (e-mail: chengqi@it.uts.edu.au).

The remainder of the paper is organized as follows. Section II introduces the basic model of XEvent. Section III gives an overview of XEvent. In section IV we describe the basic operations in XEvent. Our XEvent prototype built on Bamboo[14] and experimental results are shown in section V. Related work is introduced in section VI, and section VII gives the conclusion.

II. BASIC MODEL IN XEVENT

View: In XEvent, we decompose an XML message of tree structure into multiple sequential elements from the root to all leaves. We call the sequential elements with the element content as Data View $\boldsymbol{V}_d$, which includes an element path $\boldsymbol{V}_{dp}$ and element content $\boldsymbol{V}_{dc}$. $\boldsymbol{V}_{dp}$ is the sequential elements from a root to a leaf, and all elements is separated by the parent-child connector '/' to show the parent-child relationship between two elements. $\boldsymbol{V}_{dc}$ represents the element contents including the element texts and attributes. Fig. 1 shows an XML message and its $\boldsymbol{V}_d$.

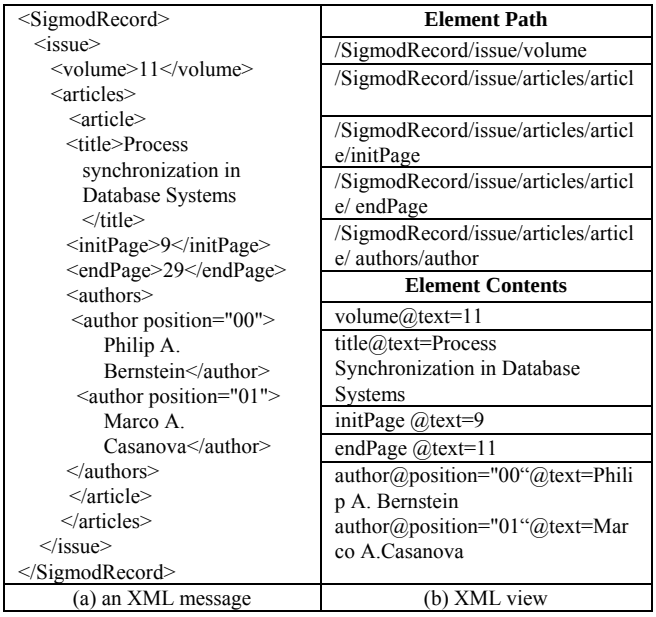


Fig. 1. An XML message and its XML views

Similarly with $\boldsymbol{V}_d$, we can define the schema view ($\boldsymbol{V}_s$) based on an XML schema, which can be also modeled as a tree-like structure. An XML schema only specifies the structure information without content, while $\boldsymbol{V}_s$ only defines the sequential elements from the root to the leaves as the element paths ($\boldsymbol{V}_{sp}$). Fig. 2 shows an XML schema (DTD) and view $\boldsymbol{V}_s$.

XPath Model: we use XPath as the filtering language providing an expressive filtering against XML messages in publication. XPath can support both path structure and element content filtering. The path structure filtering in XPath supports a parent-child operator '/' or an ancestor-descendant operator '//'. Also XPath allows the use of a wildcard operator '*'. XPath also allows element content filtering against both element attributes and element texts. We call the sequential elements in an XPath filter $\boldsymbol{F}$ from the head element of $\boldsymbol{F}$ to the first element followed by a relative path separator in $\boldsymbol{F}$ as a key $\boldsymbol{K}$ of $\boldsymbol{F}$. For example,

- If F1 = /SigmodRecord/issue/articles/article/title then K1 = F1 = /SigmodRecord/issue/articles/article/title
- If F2 = /SigmodRecord/issue/articles//title, then K2 = /SigmodRecord /issue /articles

We define two operations for the XPath filter $\boldsymbol{F}$: "+"and "-". If $\boldsymbol{F}_1 = \boldsymbol{E}_1 \wedge \ldots \wedge \boldsymbol{E}_{k-1} \wedge \boldsymbol{E}_k \wedge \ldots \wedge \boldsymbol{E}_n$ and $\boldsymbol{F}_2 = \boldsymbol{E}_k \wedge \ldots \wedge \boldsymbol{E}_n$, then $\boldsymbol{F}_3 = \boldsymbol{P}_1 - \boldsymbol{F}_2 = \boldsymbol{E}_1 \wedge \ldots \boldsymbol{E}_{k-1}$ and $\boldsymbol{F}_3 + \boldsymbol{F}_2 = \boldsymbol{F}_1$. Also, if $\boldsymbol{F}_1 = \boldsymbol{E}_1 \wedge \ldots \wedge \boldsymbol{E}_{k-1} \wedge \boldsymbol{E}_k \wedge \ldots \wedge \boldsymbol{E}_n$ and $\boldsymbol{F}_3 = \boldsymbol{E}_1 \wedge \ldots \wedge \boldsymbol{E}_{k-1}$, then $\boldsymbol{F}_4 = \boldsymbol{P}_1 - \boldsymbol{F}_3 = \boldsymbol{E}_k \wedge \ldots \wedge \boldsymbol{E}_n$ and $\boldsymbol{F}_4 + \boldsymbol{F}_3 = \boldsymbol{F}_1$.

A covering relation between two XPath filters is defined that if $\boldsymbol{F}_1$ is more specific than $\boldsymbol{F}_2$, then $\boldsymbol{F}_2$ covers $\boldsymbol{F}_1$. The relation is also written as $\boldsymbol{F}_2 \supseteq \boldsymbol{F}_1$ or $\boldsymbol{F}_1 \subseteq \boldsymbol{F}_2$, *if and only if* both the path structure filtering $\boldsymbol{P}_2$ and the element content filtering $\boldsymbol{C}_2$ of $\boldsymbol{F}_2$ covers the path structure filtering $\boldsymbol{P}_1$ and the content filtering $\boldsymbol{C}_1$ of $\boldsymbol{F}_1$. Based on the XPath covering relationship, we can construct an XPath covering graph. In the graph, a node represents an XPath filter and an edge represents the covering relationship between XPath filters. If $\boldsymbol{F}_1 \subsetneq \boldsymbol{F}_2$, the node representing $\boldsymbol{F}_1$ is the parent of the node representing $\boldsymbol{F}_2$. As a result, the most specific filter is the root of the graph, and the least specific filter is the leaf node of the graph. Given the filters shown in Fig. 3(b), we know the following XPath covering relations, and the XPath covering relation graph is constructed as Fig. 3(a):

- $\boldsymbol{F}_1 \subseteq \boldsymbol{F}_3 \subseteq \boldsymbol{F}_5 \subseteq \boldsymbol{F}_2 \subseteq \boldsymbol{F}_6 \subseteq \boldsymbol{F}_7 \subseteq \boldsymbol{F}_8$;
- $\boldsymbol{F}_4 \subseteq \boldsymbol{F}_6 \subseteq \boldsymbol{F}_7 \subseteq \boldsymbol{F}_8$.

III. OVERVIEW OF THE XEVENT SYSTEM

In XEvent, we define the XML topic (schema) as the event topic. The event messages in XML format are instances of the event schema. XEvent supports subscriptions depending on both the event topic and event content filtering. The introduction of event schema into the event system can be useful for both publishers and subscribers. The schema can validate an XML message and enforce the validity of XML messages; Event subscribers can directly subscribe a certain topic of event messages based on an XML schema, and further subscribe the message using XPath as the filtering language.

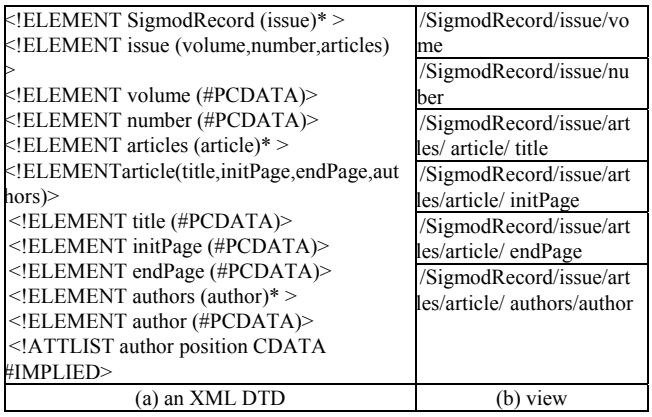
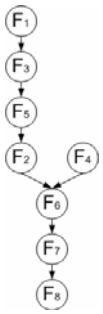


Fig. 2. Event schema (DTD) and element path of schema views

We build XEvent (an event notification system) over DHT by arranging a set of cooperating peers in a distributed topology. Each peer in XEvent can act as three roles: publisher, subscriber or



Filter	XPath
F1	/SigmodRecord/issue/articles/article/authors/author [@position="00"]
F2	/SigmodRecord//author[@position = "00"]
F3	/SigmodRecord/issue/articles/article/*/author[@position = "00"]
F4	/SigmodRecord/issue/articles/article/authors/author [@position>"00"]
F5	/SigmodRecord/issue/articles/article//author [@position = "00"]
F6	/SigmodRecord//author[@position >= "00"]
F7	/SigmodRecord//author
F8	//author

(a) (b)

Fig. 3. XPath Relation Graph

broker. The basic operations in XEvent include: schema registration, event subscription and message publication. These operations are the three steps for an event message to be delivered from a publisher to subscribers. Firstly, the registration of an XML schema as event topic into XEvent is mandatory. Secondly, the subscribers can subscribe an interested event by use of the event schema as event topic and an XPath filter as content filter. Thirdly, when a message producer publishes an XML message after the validation of the XML schema, the event message is filtered over XEvent and forwarded by intermediate peers until the message is delivered to its interested subscribers.

In XEvent, we build two layers of overlay network over the IP network (Fig. 4), i.e. event sketch tree and delivery tree. The event sketch tree (described in section 4.1) is built based on the event schema (topic) view. The event delivery tree (described in section 4.2) is organized logically based on the XPath relation graph while physically within the event sketch tree. Using the delivery tree, the event is directly forwarded to its most specific and matched XPath filter node in the event delivery tree (described in section 4.3). If the message matches the most specific XPath filter, all descendant filters of the most specific filter are also matched with the message. Such a delivery tree can greatly reduce the overhead of message routing and matching. Concerning the reliability, an event sketch tree can self-heal its node failure and provide the reliability for the upper event delivery tree (described in section 4.4), and the event delivery tree can devote itself to message filtering and delivery without care of node maintenance.

IV. Basic Operations in XEvent

In this section, we give the description of XEvent operations: schema registration, event subscription, message publication and maintenance of event delivery tree.

A. Schema Registration

During schema registration, an event skeleton tree $\boldsymbol{T}$k is built. The event topic (schema) $\boldsymbol{S}$, which is a tree structure, can generate views $\boldsymbol{V}$s as defined in section 2. The destination host of Hash($\boldsymbol{S}$) in DHT namespace is termed as $\boldsymbol{H}$t. As the definition of XPath key in section 2, we term $\boldsymbol{F}$root as the element paths $\boldsymbol{V}$sp, and $\boldsymbol{K}$root as the key of $\boldsymbol{V}$sp. As a result, $\boldsymbol{K}$root = $\boldsymbol{F}$root = $\boldsymbol{E}$1/…/$\boldsymbol{E}$n since all path separators in $\boldsymbol{F}$root = $\boldsymbol{E}$1/…/$\boldsymbol{E}$n are '/'. For an event schema $\boldsymbol{S}$ in XEvent and the element paths $\boldsymbol{F}$root of $\boldsymbol{V}$s, we introduce the destination of Hash($\boldsymbol{K}$root) in DHT namespace as the root node of $\boldsymbol{T}$k, termed as $\boldsymbol{H}$root.

When building $\boldsymbol{T}$k, $\boldsymbol{F}$root can be further decomposed into multiple sub-filters $\boldsymbol{F}$s and each of $\boldsymbol{F}$s is the sequential elements from the head element of $\boldsymbol{F}$root to other elements of **F**root. For

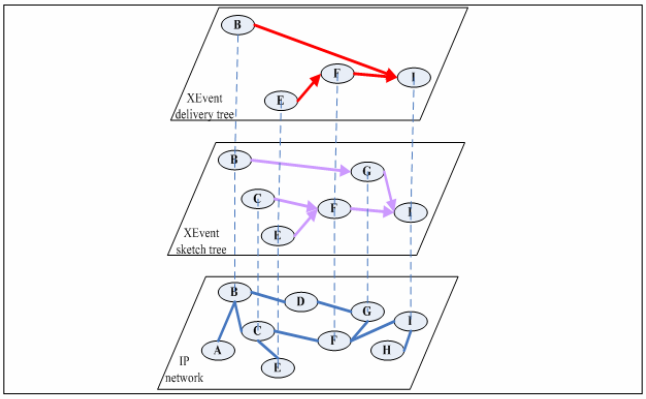


Fig. 4. XEvent Overlay Network

$\boldsymbol{F}$root = $\boldsymbol{E}$1/…/$\boldsymbol{E}$n, the sub-filters $\boldsymbol{F}$s can be $\boldsymbol{E}$1/…/$\boldsymbol{E}$n-1/$\boldsymbol{E}$n, $\boldsymbol{E}$1/ …/$\boldsymbol{E}$n-1, …, $\boldsymbol{E}$1/$\boldsymbol{E}$2, $\boldsymbol{E}$1. For each sub-filter $\boldsymbol{F}$s, the key $\boldsymbol{K}$ of $\boldsymbol{F}$s = $\boldsymbol{F}$s because all elements in sub-filter $\boldsymbol{F}$s are connected by '/'. Then the skeleton tree $\boldsymbol{T}$k, rooted at $\boldsymbol{F}$root, is built based on the filter covering the relation graph of all sub-filters $\boldsymbol{F}$s. Each sub-filter $\boldsymbol{F}$s is physically located the destination host of Hash ($\boldsymbol{K}$) in the DHT namespace. The connection of two sub-filters represents the covering relationship of two sub-filters, the parent sub-filter is more specific than its children, and the root sub-filter is the most specific. Fig. 5 shows the event sketch tree and its sub-filters of the event schema (topic) in Fig. 2.

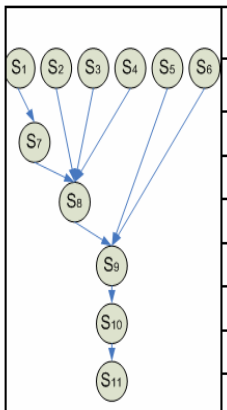
TABLE I
XPath Instantiation

XPath Filter	XPath instantiation Result
F1=//artitles//title	/SigmodRecord/issue/articles// title
F2=/*/issue//article/*/author	/SigmodRecord/issue//article/*/ author

B. Event Subscription

In XEvent, event subscribers can subscribe their interested message by an event schema (topic) and by an XPath filter. As the topic subscription is simple, we only discuss the XPath subscription in this paper. The XPath subscription can be implemented in two phases: XPath instantiation and event delivery tree building.

XPath Instantiation: we have defined the absolute path separator ('/'), and the relative path separator ('/*/' and '//'). Depending on whether an absolute path separator '/' at the head of an XPath filter or not, we category an XPath filter $\boldsymbol{F}$ into $\boldsymbol{F}$a or $\boldsymbol{F}$r, where $\boldsymbol{F}$a has an absolute path separator '/' at the head of the XPath filter, and $\boldsymbol{F}$r has a relative path separator '//' or '/*/' at the head of the XPath filter. For example, *//author* or */*/issue/articles/*



S1	/SigmodRecord/issue/articles/article/authors/author
S2	/SigmodRecord/issue/articles/article/endPage
S3	/SigmodRecord/issue/articles/article/initPage
S4	/SigmodRecord/issue/articles/article/title
S5	/SigmodRecord/issue/number
S6	/SigmodRecord/issue/volume
S7	/SigmodRecord/issue/articles/article/authors
S8	/SigmodRecord/issue/articles/article
S9	/SigmodRecord/issue/articles
S10	/SigmodRecord/issue
S11	/SigmodRecord

Fig. 5. Event sketch tree and its sub-Filters

are ***F***r. ***F***r can decompose into ***F***r except the relative path separator at the head, i.e. ***F***r = {∧***F***a} where ∧ = '/*/' or '//'. The XPath instantiation actually replaces a relative path separator at the head of ***F***r into an absolute path separators, i.e. replace the '/*/' or '//' with '/'. For these relative path separators, which are located at the intermediate or the end part of an XPath filter, like */SigmodRecord//author* or */SigmodRecord/issue/articles/article/*/author*, no instantiation is needed.

XPath instantiation can be done as follows. Given an even schema ***S*** and XPath filter ***F***r = {∧ ***F***a} where ∧ = '/*/' or '//', the instantiation result of ***F***r against ***S***, termed as ***F***i, is to replaced ***F***r with ***F***i = {***F***s ∧ ***F***a} where ∧ ≒'/', if there exists one element path ***V***ep is satisfied with ***F***r and a sub-filter ***F***s of ***V***ep where (***V***ep - ***S***)* is satisfied with ***F***a. XPath instantiation can be implemented in the similar way as [6] by use of XPath as Finite State Machines (FSM) and ***V***ep as the message. Table 1 shows two XPath filter, and the result of XPath instantiation against the event schema in Fig. 2.

Building event delivery tree: When an XPath filter ***F*** is submitted to XEvent by a subscriber, the XPath filter ***F*** is first checked that ***F*** is ***F***a or ***F***r. If ***F*** is ***F***a, ***F*** is directly used to build an event delivery tree; otherwise ***F*** needs to instantiate locally. After instantiation, the XPath instantiated result ***F***i is used to build an event delivery tree.

The key of building an event delivery tree is to make use of the fact: for event schema ***S***, ***F***root of ***S***, and an XPath filter ***F*** (***F***i or ***F***a), there always exists a sub-filter ***F***s of ***F***root, i.e. a sub-filter ***F***s is XPath key ***K*** of ***F*** (***K*** = ***F***s) if ***F*** is satisfied with the XML event message that is instance of ***S***. For such an XPath filter ***F***, the destination of Hash(***K***) in DHT namespace, as a result, is the same as the destination of Hash(***F***s) where is located at the event sketch tree. The event delivery tree ***T***d for an event topic (schema) ***S*** consists up the XPath filters for the event topic ***S***, which are organized based on the XPath filter relation graph. As a result, ***T***d, logically organized by an XPath filter covering relation graph, is physically located within ***T***k. For a new XPath filter ***F***, the subscription into XEvent can be done in both folders: physically inserted into ***T***k and logically inserted into ***T***d. For the former, it can be easily implemented by locating the destination host of Hash(***K***) since ***K*** = ***F***s; for the latter, it can be implemented by finding ***F***'s parent (child) filter at ***T***d, termed as ***F***p (***F***c), and inserting ***F*** between ***F***p and ***F***c.

The process of finding ***F***p is described as follows. Given an XPath filter ***F***, the destination ***H***k of Hash(***K***) in DHT is located at ***T***k. Based on the definition of XPath covering relation and the building of ***T***k, ***F***p is physically located at ***H***k, or at parent node of ***H***k in ***T***k, or at ancestor node of of ***H***k in ***T***k, and impossible at children nodes of ***H***k in ***T***k. Then ***F***p can be found by first traversing all existing XPath filters at ***H***k to determine the least specific XPath filter as ***F***p. If no XPath filters exist at ***H***k, then the same procedure is done to find ***F***p at parent node of ***H***k, …, ancestor node of ***H***k until the root of ***T***k. If no XPath filters exist at the parent node of ***H***k, ancestor node of ***H***k and the root of the event skeleton tree ***T***k, ***F***p returns null, which means ***F*** being the most specific filter and becoming the new root of ***T***k. Finding ***F***c can be done in the similar way as finding ***F***c. After insert, the link between ***F*** and ***F***p is established. If both ***F*** and ***F***p are located at the same host, the link within the local host is actually a pointer from ***F***p to ***F***; otherwise, a physical link between different hosts is needed. Since multiple XPath filters share the same XPath filter key, these XPath filters with the same key are located at one node in ***T***k. A physical link can contain multiple links between XPath filters. As a result a message (for example a Heartbeat message) from a parent node to a child node in ***T***k can aggregate the information of multiple XPath filter links to decrease the communication traffic.

In order to improve the performance of message delivery, we introduce a repair mechanism for the event delivery tree ***T***d (see section 4.4). Since there exists the case that the root XPath filter of ***T***d is not located at the root node of ***T***k, in the repair mechanism a virtual XPath filter located at the root node of ***T***k is used to link the current root XPath filter of ***T***d; Otherwise, the repair mechanism is not needed when the current root XPath filter of ***T***d is not located at the root node of ***T***k.

Concerning the efficiency of finding ***F***p, the IP hops of finding ***F***p (***F***c) is limited within the depth of ***T***d not the number of XPath filters, which can greatly decrease the communication cost. Thus, the cost of finding the least specific filter in a local host is negligible compared with the communication cost in DHT-based P2P system. Fig. 6(a) shows the XPath filter covering relation graph after instantiation of the XPath filters in Fig. 3(b), and Fig. 6(b) shows the event delivery tree ***T***d and its physical location in ***T***k. Here note: (1) ***F***8 (*//author*) after XPath instantiation becomes */SigmodRecord/issue/articles/article/authors/author*. Then the new XPath covering relation graph in Fig. 6(a) is twisted from the one shown in Fig. 3(a). (2) The filters in ***T***d are located within ***T***k, but the physical link between filters in ***T***d does not follow the physical link in ***T***k. ***S***1 and ***S***8 in ***T***d is directly linked not via ***S***7.

C. Message Publication

In XEvent, only the valid XML message against the event topic (schema) can be published into XEvent. A valid XML message

* About definition of -, see the XPath model in section 2

can be decomposed into XML views ***V***d with element contents ***V***dc and element path ***V***dp. For ***V***d, the destination of Hash(***V***dp) in the DHT is a root node ***H***root of the event sketch tree ***T***k for the event schema as described in section 4.1. Based on the repair mechanism of the event delivery tree ***T***d, there is always a root XPath filter ***F***r at ***H***root. If ***F***r is virtual, the incoming ***V***d is delivered to its child XPath filter for further message filtering. Otherwise, ***V***d is matched with ***F***r. If ***V***d is satisfied with the ***F***r, ***V***d is also satisfied with all of its children. As a result, the XML message can be directly routed to the subscribers who subscribe ***F***r and all children of ***F***r. If the XML view ***V***d is unsatisfied with ***F***r, the traversing the event delivery tree from ***F***r to its children is needed until the XML view ***V***d is satisfied with an XPath filter ***F*** in ***T***d. Then, we can determine that the XPath filter ***F*** and its all children filters in ***T***d are interested in the XML message. Also, the XML message is delivered to the subscribers who subscribe ***F*** and its children filters. If an XPath filter ***F*** receives a previous duplicated ***V***d, the XML view ***V***d will not be forwarded and it will be discarded. In section 4.2, we introduce the repair mechanism of ***T***d. We argue that if a virtual XPath filter exists at the root of ***T***k, the matching can directly be forwarded to its child filter without traversing the child node to find the root filter of ***T***d.

About efficiency of message publication, since XPath filters with the same key are located at the same node of ***T***k, the forwarding cost within the local node is negligible compared with the forwarding cost between physical nodes. Also, the forwarding hops are limited within the depth of ***T***k not of ***T***d, which can greatly decrease the communication cost.

D. Maintenance of an Event Delivery Tree

The requirement of reliability needs XEvent to be robust against node failure. To repair the node failure in ***T***k, a node in ***T***k is required to periodically exchange Heartbeat messages with its parent/child nodes to detect failure. Each XPath filter in the node has a soft state with leases for its parent/child XPath filter.

We describe failures and their repair mechanisms as follows. Firstly, failure of ***H***t can be detected by the Heartbeat message from the root node ***H***root in the event sketch tree ***T***k, but the failure cannot cause any hamper of message publication or event subscription except for those XPath filters which need

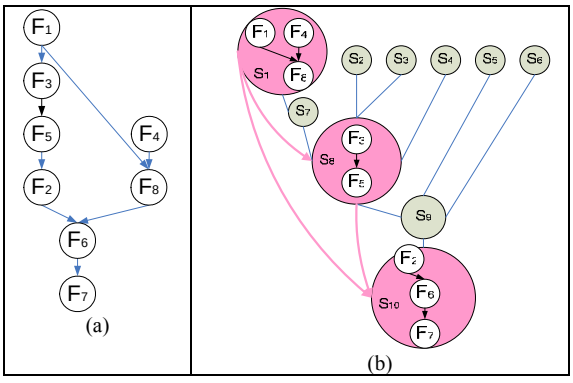


Fig. 6. Event Delivery Tree

instantiation at ***H***t. The failed ***H***t can be healed by any publisher, subscriber or third-party to re-register the event topic (schema) while without re-building the entire event delivery tree ***T***d. Secondly, failure of ***H***root in ***T***k can be detected by the Heartbeat message from ***H***t, and can be healed by ***H***t to re-lookup the destination host of Hash(***F***root) at DHT as the new ***H***root. Thirdly, failure of any other node in ***T***k except ***H***root can be detected by its parent node. The parent node, destination of Hash(***E***1/…/***E***k-1/***E***k) in DHT, can re-lookup the destination of Hashing (***E***1/…/***E***k-1) to heal the failed child.

Even before the node in ***T***k is recovered, XEvent provides a reliable message forwarding mechanism. The XML view ***V***d can be first routed to the children node of ***T***k, not the failure node where its children XPath filter is located. This begins the message matching against the XPath filters at the children nodes.

V. Experimental Evaluation

In order to evaluate XEvent performance, we implement the prototypes of basic SCRIBE [9] and XEvent using the discrete event simulator over Bamboo [14], a recent DHT-based P2P system. The simulation environment is a network topology with 5050 routers, which were generated by the Georgia Tech random graph generator using the transit-stub model [15]. The prototype codes including SCRIBE and XEvent run on 100,000 end nodes that were randomly assigned to routers in the core with uniform probability. Each end system was directly attached by a LAN link to its assigned router. During the evaluation, we designed the following test scenario: (1) *SigmodRecord.dtd.* [16] as an event topic (schema) is registered into XEvent; (2) 4096 XPath filters are generated by the IBM XPath query generator[17] to subscribe their interested messages; (3) XML-based event messages, which are validate with the XML event topic (schema), are published into XEvent continuously as a message stream. Finally, XEvent routes the published messages to all satisfied subscribers. From the time that the messages are published to the time that all satisfied subscribers get the messages, we evaluate XEvent and SCRIBE. For prototype implementation of SCRIBE, we filter the coming message content at the end host of SCRIBE, which is the destination of Hash (XPath key).

To facilitate our comparison, we use two matrices: Relative

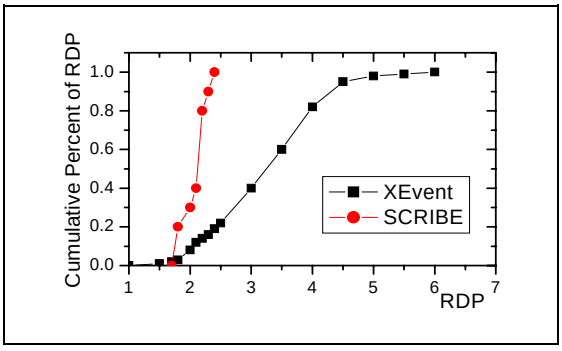


Fig. 7. The cumulative distribution of Relative Delay Penalty (RDP) of both XEvent and SCRIBE.

Delay Penalty (RDP) and physical link stress, which were originally proposed in [13] and were sequentially used in [10]. In addition to these two matrices, stability is also evaluated.

Relative Delay Penalty (RDP) is a measure of the increase in delay that applications incur while using overlay routing [13, 10]. Using RDP on both XEvent and SCRIBE [10], it is the ratio of DHT overlay unicast routing distances to IP unicast routing distances. Assuming symmetric routing, IP Multicast and naive unicast both have a RDP of 1. We use the ratio of the amount of total overlay DHT unicast routing hops to the amount of all naïve unicast hops as the RDP. Fig. 7 shows the cumulative distribution of RDP of both XEvent and SCRIBE. The X-axis represents the RDP value and y-axis represents the cumulative percent of RDP. XEvent has an average RDP value of 1.9 and SCRIBE has the value of 4.2. It can be explained that all communication in XEvent is within the event sketch tree, while the communication in SCRIBE is scattered through out the whole network.

Link Stress is a measure of how effective SCRIBE and XEvent prototypes are in a distributed network load across different physical links. It refers to the number of identical copies of a packet carried by a physical link. IP multicast has a stress of 1, and naïve unicast has a worst case stress equal to number of receivers. We calculate the ratio of the total number of messages that are sent over links to the total number of links as the average link stress. Fig. 8 shows the link stress comparison of XEvent and Scribe. The X-axis represents the Log scale of Link stress, i.e. the duplicate message copies across the physical link. The Y-axis represents the Log scale of link number. Obviously, XEvent has much less link stress than SCRIBE. It can be easily explained that the link between the nodes in XEvent event sketch tree can contain multiple logical links between XPath filters of XEvent event delivery tree. Also the logical link of XPath filters within the same local host does not require any physical communication.

Stability: To evaluate the stability of XEvent, we need to design an unstable scenario by re-designing step 3 in normal scenarios. We separately design 10%, 20% and 30% nodes crashed in the network model from normal message publication in the time interval of 1000 seconds, then we count the ratio of the nodes, which can receive the message, to all live nodes. In Fig. 9, X-axis represents time and Y-axis represents the ratio in percent. From the figure, the time interval for XEvent (with the self-healing mechanism) to recover 100% message receiving is shorter than for SCRIBE. Also, the failure makes the data lost of some nodes in SCRIBE since we simply implement the basic SCRIBE without the group maintenance mechanism.

VI. Related Works

Recent work on the scalable design of structured P2P overlay networks has introduced a new class of structured networks called Distributed Hash Tables (DHT). Well known representatives include [3, 4, 5]. All of these systems were built to allow efficient key lookups. Nodes in a P2P network send messages to each other based on a unique name, generated from a secure one-way hash of some unique string. Assigned a unique ID to each node by DHT, a message is delivered to the destination host in a fault-resilient fashion. The P2P routing and locating infrastructure can provide scalability, fault-tolerance, self-maintenance and adaptation for upper applications.

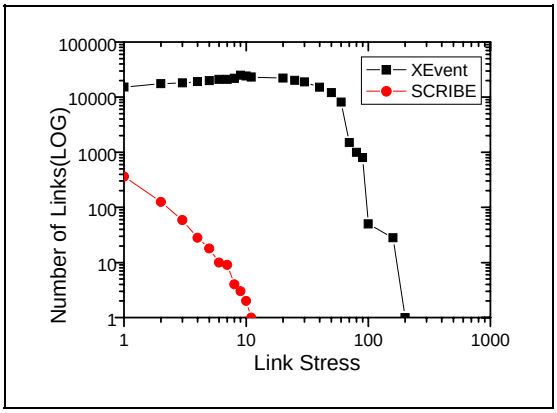


Fig. 8. The link stress comparison of XEvent and Scribe.

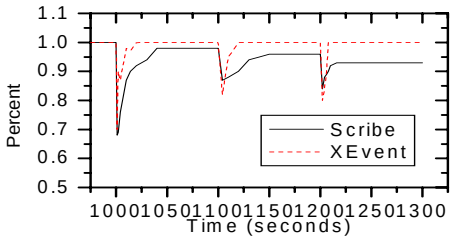


Fig. 9. XEvent stability

Several event notification systems have already developed over DHT. SCRIBE [9] is a topic-based event notification system that is built on top of "Pastry" [4]. "Pastry" is a DHT system developed at Rice University. Subscribers join topics of interest, where each topic is identified with a Pastry-level key. Bayeux [10] is built on top of Tapestry [5]. However, neither SCRIBE [9] nor Bayeux [10] supports content-based subscriptions and publications. Hermes [11] provides a type- and attribute-based routing schema over DHT, but the introduction the rendezvous node into Hermes can result in limited scalability and a single point of failure, also the content filtering is rather limited. An extension of traditional Bloom filters, called multi-level Bloom filters [12], can be used to route path queries in a P2P system, and build content-based overlay networks by linking together peers with similar content. However, the P2P system is totally different from the DHT-based structured P2P systems. Therefore, the system will not or cannot implement a flexible filtering mechanism like XEvent.

Some pioneer researchers in database field have incorporated XML into publish /subscribe system [6, 8]. An XML document filtering system, for Selective Dissemination of Information (SDI) has been proposed in [6], which allows users to define their interests using XPath query language. Using a modified Finite State Machine (FSM), a novel indexing mechanism and matching algorithms are developed. YFilter, combines all of the queries into a single Non-deterministic Finite Automaton (NFA) [8]. The approach exploits commonality among path expressions by merging common prefixes of the query paths so that they are

processed at once. All of these approaches are based a single centralized filtering engine and are hard to scale to a large distributed network.

TABLE II
ABBREVIATIONS LIST

$\boldsymbol{V}_d$	XML Data View
$\boldsymbol{V}_{dp}$	Element path in $\boldsymbol{V}_d$
$\boldsymbol{V}_{dc}$	Element content in $\boldsymbol{V}_d$
$\boldsymbol{V}_s$	Schema View
$\boldsymbol{V}_{sp}$	Element path in $\boldsymbol{V}_s$
$\boldsymbol{F}$	XPath filter
$\boldsymbol{S}$	Event schema (topic)
$\boldsymbol{K}$	Key of $\boldsymbol{F}$
$\boldsymbol{F}_{root}$	Element paths $\boldsymbol{V}_{sp}$
$\boldsymbol{K}_{root}$	Key of $\boldsymbol{F}_{root}$
$\boldsymbol{H}_{root}$	Destination Host of $\boldsymbol{F}_{root}$ in DHT namespace
$\boldsymbol{T}_k$	Event sketch tree
$\boldsymbol{T}_d$	Event delivery tree
$\boldsymbol{F}_s$	XPath Sub-Filter of $\boldsymbol{V}_{sp}$
$\boldsymbol{F}_a$	XPath Absolute Filter
$\boldsymbol{F}_r$	XPath Relative Filter
$\boldsymbol{F}_i$	XPath Instantiation result
$\boldsymbol{F}_p$	XPath Parent filter in $\boldsymbol{T}_d$
$\boldsymbol{F}_c$	XPath child filter in $\boldsymbol{T}_d$
$\boldsymbol{H}_t$	Destination Host of Hash($\boldsymbol{S}$) in DHT namespace

VII. CONCLUSION

We have presented XEvent, a novel hybrid event notification system built over Distributed Hash Table (DHT)-based Peer-to-Peer (P2P) system. XEvent can support both event topic and event content subscription, which can provide expressiveness for subscribers by the use of XML as event message format and use of XPath as subscription filter.

XEvent make use of DHT to build two layers of an overlay network: event sketch tree and event delivery tree. An event sketch tree can provide the stability for the upper event delivery tree. The event delivery tree can devote itself to swiftly match and deliver messages to the interested subscribers, without considering the churn in DHT.

The initial simulation results indicate that XEvent can provide an efficient event notification system with quick recovery mechanisms and low Relative Delay Penalty and link stress compared with the SCRIBE [9] system.

REFERENCES

[1] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Content-based addressing and routing: A general model and its application," Department of Computer Science, University of Colorado, USA, Technical Report CU-CS-902-00, Jan. 2000.

[2] R. Chand and P. Felber, "A scalable protocol for content-based routing in overlay networks," in *IEEE International Symposium on Network Computing and Applications (NCA'03),* Cambridge, Massachusetts, Apr. 2003.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proceedings of ACM SIGCOMM'01*, San Diego, CA, Aug. 2001.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *MiddleWare*, Germany, Nov. 2001.

[5] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," University of California Berkeley, USA, Technical Report CSD-01-1141, 2001.

[6] M. Altinel and M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," in *Proceedings of the 26th International Conference on Very Large Data Bases,* Cairo, 2000, pp 53–64.

[7] A. K. Gupta and D. Suciu, "Stream processing of XPath queries with predicates," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, California, USA, 2003, pp. 419–430.

[8] Y. Diao and M. Franklin, "Query Processing for High-Volume XML Message Brokering," in *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, 2003, pp. 261–272.

[9] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications,* vol. 20, no. 8, pp. 1489–1499, Oct. 2002.

[10] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant WideArea Data Dissemination," in *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video,* USA, June 2001.

[11] P. R. Pietzuch and J. Bacon, "Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware," in *Proceedings of the 2nd International Workshop on Distributed Event-based Systems,* USA, 2003.

[12] G. Koloniari, Y. Petrakis and E. Pitoura, "Content-Based Overlay Networks of XML Peers Based on Multi-Level Bloom Filters," in *International Workshop on Databases, Information Systems and Peer-to-Peer Computing,* Germany, 2003.

[13] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," in *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, USA, 2000.

[14] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *Proceedings of the USENIX Annual Technical Conference (USENIX),* Boston, Massachusetts, June 2004.

[15] Georgia tech internet topology model. http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html

[16] http://www.cs.washington.edu/research/xmldatasets/

[17] http://www.alphaworks.ibm.com/tech/xmlgenerator