TinyMQ: A Content-based Publish/Subscribe Middleware for Wireless Sensor Networks

Ke Shi and Zhancheng Deng
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China
keshi@mail.hust.edu.cn

Xuan Qin
Research and development Office
Huazhong University of Science and Technology
Wuhan, China
kjcqx@mail.hust.edu.cn

Abstract—**The emergence of Wireless Sensor Networks (WSN) technologies gave rise to the need for abstraction mechanisms that can simplify the data communication tasks. Under this respect, the publish/subscribe paradigms play an important role since they can be used to abstract the WSN in terms that are closer to the needs of applications designed mainly to detect and notify upon events of interests. In this paper, we propose TinyMQ, a content-based publish/subscribe middleware for wireless sensor network. In TinyMQ, an overlay network is constructed on top of the underlying WSN, in which sensor nodes can be logically connected independent of their geographical position. Over this overlay network, message mapping and routing schemes are presented to implement rendezvous-based publish/subscribe. The simulation study demonstrates TinyMQ is efficient in terms of both costs and time.**

Keywords-publish/subscribe; content-based; wireless sensor network

I. INTRODUCTION

In WSNs (Wireless Sensor Networks), the sensor nodes cooperatively sense the environment and send the sampled data to the base station. Since many nodes may transmit a large amount of data synchronously, how to collect, transport, and process the data efficiently is the main research domain of the WSN applications [1]. Event-driven publish/subscribe is considered as an efficient method for data collecting and transmitting tasks in WSN. It is because data is sensed and transmitted only when certain events occur. Compared with the request/response model in which information exchange occurs by issuing requests and waiting for corresponding responses, publish/subscribe model can reduce the network overload significantly [2].

Implementing an efficient publish/subscribe mechanism in the WSNs is challengeable [3]. WSN is an ad-hoc, self-organized and totally distributed network. The subscriber nodes, which are interested in some events, do not know whether, when and where the events may occur. And the publisher nodes do not know which nodes are interested in these events either. Publish/subscribe mechanism in WSN must be distributed. Moreover, the WSN is resource-limited (in terms of computation ability, storage ability, and communication ability), event mapping and routing policies should be carefully designed to achieve publish/subscribe functionality and prolong the life of the WSN as much as possible.

Rendezvous-based model is a frequently used model for distributed publish/subscribe. In this model, the queries and events are sent to a set of selected nodes called rendezvous nodes that act as the brokers where the interests of the subscribers and publishers match. The mapping and routing policy should guarantee that there exists a rendezvous node that receives both the query and its matched event. When using predetermined physical node IDs, a query or event may be mapped to an ID that does not exist because the node that initiates the query or event has only local network information. Location-based method maps the query or event to a set of geographic locations and publishes it at the nodes nearest these locations. However, most sensor nodes are deeply embedded. It is impossible from them to integrate GPS-like positioning devices. And most existing position algorithms for WSN consume a lot of computing and energy resource [4].

Gossip-based approach does not require location information. In this model, each query or event is spread throughout the network by probabilistic broadcasting [5]. The queries and events are sent to almost all the nodes, which guarantee the high probability that the interests of subscribers and publishers match. However, since each query or event must be sent to almost all the nodes, network overload may be very high and serious congestion may occur frequently.

A more efficient mechanism that does not require location information is needed to address this problem. Gossip-based approach does not concern the content of certain query/event and sent a query/event message randomly. To guarantee that an event meets every query, excessive transmission becomes necessary. It causes unnecessary query and event forwarding in an unstructured network environment. We think, with a carefully designed overlay network (structured network environment), a query/event message can be forwarded more efficiently based on content-related mapping and routing. We propose TinyMQ, a content-based publish/subscribe middleware for wireless sensor network. In TinyMQ, an overlay network is constructed on top of the underlying WSN, in which sensor nodes can be logically connected independent of their geographical position. The unique keys represented by the binary strings chosen from *{0, 1}* is used as the logical

 ISBN: 978-1-61208-144-1

addresses of in the proposed overlay network, which enables hash based content-related message mapping and routing. This mapping mechanism guarantees the events meet the queries in the certain rendezvous nodes.

The rest of this paper is organized as follows. In Section 2, we review the related researches. Section 3 describes the details of TinyMQ including overlay maintaining, message (event and query) mapping and routing. In Section 4, the performance of the proposed scheme is evaluated. Finally, in Section 5, we present the conclusion and future works.

II. RELATED WORK

Several publish/subscribe approaches in WSN have been proposed in the literature. This section introduces these existing approaches briefly.

The Mires [6] middleware addresses the implementation of publish/subscribe communication for wireless sensor network applications. It adopts topic-based publish/subscribe mechanism that can reduce the number of transmissions and energy consumption because only the messages referring to the subscribed topic are sent to a sink node. In Mires, the publisher nodes must know which nodes the events are sent to and which paths the events are followed. The implementation detail and performance evaluation are not given either.

TinyDDS [7] is a lightweight implementation of the OMG DDS Specification Standard in WSNs. TinyDDS can adaptively perform event publication according to dynamic network conditions and autonomously balances its performance among conflicting objectives. It leverages an evolutionary multi-objective optimization mechanism to seek the optimal tradeoffs among objectives and adjust parameters in its event routing protocol. However, topic-based DDS is designed for the traditional distributed system that has a lot of difference with WSN. The main goal of TinyDDS is to provide the interoperability between WSNs and access networks. The goal of TinyMQ is to provide the interoperability among the nodes in a WSN.

PUB-2-SUB [8], a publish/subscribe framework for P2P networks, is based on two key design components: the virtualization component and the indexing component. The virtualization component maintains a naming structure by assigning to each node a unique virtual address. A tree-like overlay network is constructed and maintained. The indexing component determines the corresponding subscription and notification paths for given queries and publications, in which routing is based on the virtual addresses of the nodes. PUB-2-SUB$^+$ is an extension of PUB-2-SUB for WSNs [9]. It is location-free and content guide. However, load balance is not considered in PUB-2-SUB$^+$, which can cause high throughput and even congestion in some region where many events and queries are directed, such as the region near the root nodes. The nodes in this region may exhaust their energy and crash down quickly.

Compared with PUB-2-SUB+, TinyMQ partitions the whole network into multiple trees. A query/event is mapped to multiple nodes in the different trees. It facilitates load-balancing and high-availability. TinyMQ also allows cross-tree routing, which relieves the burden of root nodes.

III. TINYMQ

TinyMQ is a content-based publish/subscribe middleware for WSNs, which employs a general message mapping and routing mechanism on top of an overlay network to provide scalable and adaptable publish/subscribe. This section will explain how TinyMQ solves the existing problems.

A. Architecture

In TinyMQ, a sensor node can act both as a producer and a consumer of information, playing the role of publisher and subscriber, respectively. Publishers and subscribers exchange messages in form of events and queries.

TinyMQ consists of two layers: overlay layer and pub/sub layer. The overlay layer maintains a naming structure by assigning to each node a unique virtual address. The sensor network is organized in a logical topology despite node failures and network churn. Publish/subscribe layer provides message mapping and routing functionality for event subscription, event publication and event notification. The message mapping and routing mechanism determines the corresponding subscription and notification paths for given queries and events, in which routing is based on the virtual addresses of the nodes.

For events and queries, the initiated nodes select the rendezvous nodes through message mapping and sent corresponding messages to these selected rendezvous nodes through message routing. The rendezvous nodes check the events against the queries in order to determine whether dispatching the events to the subscribers or not.

B. Overlay network

We assume that the sensor network is a connected graph of n stationary nodes $\{S_1, S_2, \ldots, S_n\}$ and that there are one or more nodes $\{S_1^*, S_2^*, \ldots, S_m^*\}$ $(m \geq 1)$ that are reliable like the sink nodes or base stations. Two nodes are called "neighbors" if they can communicate directly with each other.

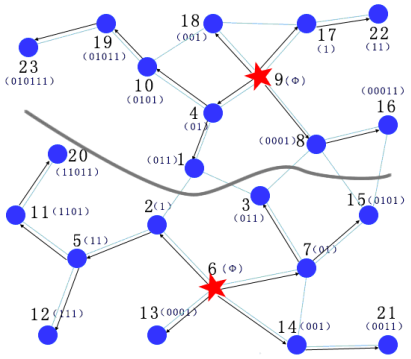

Figure 1. The overlay network with two root nodes (6, 9)

Each node is assigned a binary string chosen from *{0, 1}* called key value to indicate its logical address, denoted by *key*(S_i). Using 0 or 1 symbols facilitates designing hash-based mapping mechanism.

TinyMQ maintains a multi-tree overlay on top of this kind of network, in which the stable sink nodes serve as root nodes and initiate the overlay constructing process. Every node belongs to only one tree. The nodes belonging to the different trees may have same key value, which means more nodes can act as rendezvous nodes for certain query/event message.

The form and structure of logic address is similar with PUB-2-SUB$^+$. The key value of S_i, a child of S_i*, is the shortest string of key(S_i*) + '0*1' unused by any other child node of S_i*. However, PUB-2-SUB$^+$ maintains only one tree across the whole network, which may lead to serious congestion near the root node. Although PUB-2-SUB$^+$ claims that it can build multiple trees to relieve the congestion, each tree must contain all the nodes. It will cause large coordinating and maintaining overhead. In TinyMQ, there are m trees rooted by S_i* across the whole network. Each node only belongs to one tree. The network is partitioned into m tree-based clusters. Therefore, the possible congestion near the root node can be avoided and better load balance can be achieved. An example of this kind of overlay is illustrated in Fig. 1, where the network comprises of 23 nodes, two of which (node 6 and node 9) serve as the root nodes.

Given a node X, the key zone of X, denoted by *kzone*(X), is defined as the set of all the binary strings, of which *key*(X) is the longest prefix. In Fig. 1, the key zone of root 6 is *kzone*(6) = {'0', '00', '000', '0000', '0000*'}, and for a normal node 10, *kzone*(10) = {'0101', '01010', '01010*'}.

In this multi-tree overlay, each node X has a *state*, represented by *info*(X) = < *key*(X), *root*(X), *dist*(X) >, where *key*(X) is the key value of node X, *root*(X) is the root of tree which X resides, and *dist*(X) = < *dist*(X, S_1*) , *dist*(X, S_2*) , …, *dist*(X, S_m*) > is the vector of shortest distances from the root nodes to X. The distance is measured by the number of hops. If X is the root node, let key(X) = φ and *root*(X) is itself. The key value is used to route messages inside a tree, whereas the distance information is used to route messages from a tree to another. Exception for the root nodes, all the above information is initially unknown for every node but will eventually be filled as nodes exchange information with their neighbors. Specifically, once a node X updates its *info*(X), it advertises the *info*(X) to all its neighbors. Upon receipt of such a message, each neighbor Y updates its state as follows:

(1) Update distance information: set *dist*(Y, S_i*) = *min*(*dist*(Y, S_i*), *dist*(X, S_i*) + 1) for each root node S_i*.
(2) If *key*(Y) has not been set, update key value and root information,
    (a) Set root(Y) = *root*(X) (i.e., Y is in the same tree as X).
    (b) Set key(Y) to a binary identifier according to the Key prefix rule: *key*(Y) is a string (shortest length preferred) of the form *key*(X) + "0…01" unused by any other neighbor node of X. X is called the "parent node" of Y.

C. Message mappings

We now consider the message mapping and routing mechanism and analyze how to accomplish publish/subscribe in WSNs. The notations that we will use in the following sections are defined in Table 1.

TABLE 1 NOTATIONS FOR MESSAGES

Notations	Definitions
d	The number of dimensions in the event space
e	An d-dimensional event
$e.a_i$	A i-th attribute of event
Q	A d-dimensional subscription (query)
$Q.c_i$	A i-th constraint of subscription
Ω	A d-dimensional event space
Ω_i	Projection of Ω on i-th dimension
$\sum$	A d-dimensional subscription space

For each publish/subscribe application, we assume the interest has a fixed number of attributes called the dimension. The event is defined as some new information that a node wants to publish. The queries of interest are those that specify a lower-bound and an upper-bound on each event attribute. A query Q is represented as a d-dimensional interval vector $\{ c_1 , \ldots, c_i ,\ldots\}$, where $c_i = [q^l_i, q^h_i]$, $q \in \{0,1\}^k$ is the constraint on the *ith* dimension. A query subscribes to all events belonging to this interval. Each query is subscribed to all the trees and each event is published to all the trees. In PUB-2-SUB$^+$, the event mapping mechanism only deals with one dimension events.

We use the key space-split mapping mechanism proposed in [10]. In this mechanism, message mapping is based on a collection of hash functions $h_i : \Omega_i \rightarrow [0, 1]^l$; h_i maps attribute values in Ω_i to bit strings of length l. Given the set of h_i functions, a set of hash functions H_i is defined for constraints $Q.c_i$ to return sets of l-length bit strings as follows: $H_i(Q.c_i) = \{h_i(e) \mid e \in \Omega_i \wedge e$ satisfies $Q.c_i\}$.

In order to implement the matching, TinyMQ distribute across the nodes in the system the tasks of storing subscriptions, matching events against subscriptions, and delivering notifications to subscribers. Subscriptions in $\sum$ and events in Ω should be assigned to nodes through two mapping functions: SK: $\sum \rightarrow 2^N$ and EK: $\Omega \rightarrow 2^N$.

$SK(Q)$ returns a set of key values, named rendezvous key values of Q that indicate the corresponding rendezvous node responsible for storing Q, $EK(e)$ complements EK by returning the rendezvous key values of e. The nodes with the same key values are the rendezvous nodes responsible for matching e against subscriptions registered in the system. So if $e \in Q$, then $EK(e) \cap SK(Q) \neq \Phi$, which we call this property as *mapping intersection rule*.

The SK function returns all possible concatenations of bit strings: $SK(Q) = \{s_1 \circ s_2 \circ \ldots \circ s_t \mid s_i \in \{0,1\}^l \wedge s_i \in H_i(Q.c_i)\}$ which returns several rendezvous key values. To satisfy the mapping intersection rule, the EK mapping is defined as $EK(e) = h(e.a_1) \circ h(e.a_2) \ldots \circ h(e.a_t)$, i.e., it returns a single rendezvous key value. It is base on the idea of cascade hashing.

For example, in a monitoring application for cold chain logistics, a query is Q = {0 < *temperature* < 0.5， 0.85 <

humidity < 0.9}, and a event is e = {*temp* = 0.25, *humidity* = 0.88}, the mapping process is

$h_1=(\lfloor|temperature|\cdot 10\rfloor)_2$

$h_2=(\lfloor|humidity|\cdot 10\rfloor)_2$

$H(Q.c_1)=\{h_1([0,0.5])\}=\{h_1(0),h_1(0.1),h_1(0.2), h_1(0.3),h_1(0.4),h_1(0.5)\}=\{00,01,10\}$

$H(Q.c_2)=\{h_2([0.85,0.9])\}=\{h_2(0.8),h_2(0.9)\}=\{00\}$

$SK(Q)=\{H(Q.c_1)\times H(Q.c_2)\}=\{0000,0100,1000\}$

$h(e.a_1)=h_1(0.25)=\{01\}$

$h(e.a_2)=h_2(0.88)=\{00\}$

$EK(e)=h(e.a_1)\circ h(e.a_2)=\{0100\}$.

The query is sent to the nodes with the common prefix of 0000, 0100, or 1000. The event is sent to the nodes with the common prefix of 0100. These nodes are the rendezvous nodes.

D. Message routing

For ease of presentation in this section, we assume: (1) query $Q=[q_l, q_h]$ or $Q=\{q_1, q_2, \ldots, q_n\}$ (as $q_x \in \{0, 1\}^m$) is an event set which consists of several events, and every event will be presented by several binary strings; (2) subscription notation <Q, S_i*> indicates that Q will send to some nodes that belong to tree rooted by S_i*, and analogously, event notation <e, S_i*> indicates that e will be sent and stored in some nodes that belong to the tree rooted by S_i*

ALGORITHM 3.1 SUBSCRIPTION ROUTING

1	Initially, the subscription starts at the subscriber node of Q, send to the nodes of the tree Tree(S_i*)
2	At a node X that receives <Q, S_i*>
3	Quit if node X already received this Q before
4	If ($root(X) \neq S_i$*)
5	$Y = min\{dist(Y_i, S_i^*) \mid Y_i$ is the neighbor node of node $X\}$
6	Send the subscription <Q, S_i*> to the node Y
7	Else
8	Let set $Z = \{str \in \{0, 1\}^k \mid Key(X)$ is a prefix of $str\}$
9	If ($(Q - Z) \neq \Phi$)
10	Let $NK = \{Key(Y_i) \mid Y_i$ is the neighbor node of X but is not the child node of $X\}$
11	And let $M = \{< K_i, K_j, pre_{ij} > \mid$ as for $K_i (\in Q - Z)$, exists a $K_j (\in NK)$ that pre_{ij} is the largest common prefix of $K_i, K_j\}$
12	Traverse the set M
13	If (pre_{ij} is longer than $pre_{i,parent}$)
14	forward <Q, S_i*> to node K_j
15	else
16	forward <Q, S_i*> to parent node K_{parent} of the node X
17	else if $((Q \cap Z) \neq \Phi)$
18	store Q at X if $kzone(X)$ overlap with Q
19	forward Q to all children of X in Tree(S_i*)

Every node has been assigned a key value as a result of an overlay establishing process. A query can be initiated by any node in the WSN at any time. The routing algorithms to subscription (query) and publication (event) are presented below in Algorithm 3.1 and 3.2, respectively.

In the Algorithm 3.1, when the node X receives the query Q, if the node X does not belong to Tree S_i*, the subscription will be send toward Tree S_i* (Line 4-6). Otherwise, the subscription will be stored and processed in the node X (Line 18) if it overlaps the zone $kzone(X)$ of node X, or send to child nodes of node X (Line 19) if it overlaps the zone $kzone(Y)$ of the child nodes of node X, or to neighbor nodes (Line 13, 14), or to parent node (Line 15, 16) of the node X according to the largest common prefix (Line 8-12).

ALGORITHM 3.2 EVENT ROUTING

1	Initially, the event starts at the publisher node of e, send to the nodes of the tree Tree(S_i*)
2	At a node X that receives <e, S_i*>
3	If ($root(X) \neq S_i$*)
4	$Y = min\{dist(Y_i, S_i^*) \mid Y_i$ is the neighbor node of node $X\}$
5	Send the event <e, S_i*> to the node Y
6	Else
7	If ($key(X)$ is not the prefix of e)
8	Let $NK = \{Key(Y_i) \mid Y_i$ is the neighbor node of X but is not the child node of $X\}$
9	And let $M = \{< e, K_j, pre_j > \mid$ as for e, exists a $K_j (\in NK)$ that pre_j is the largest common prefix of $e, K_j\}$
10	If (pre_j is longer than pre_{parent})
11	forward <e, S_i*> to node K_j
12	else
13	forward <e, S_i*> to parent node K_{parent} of the node X
14	else if ($key(X)$ is the prefix of e)
15	find the child node Y such that $key(Y)$ is a prefix of e
16	if (Y exists) then forward e to Y
17	else search node X for those subscriptions matching e

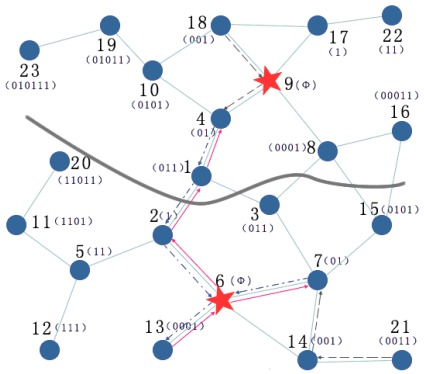

Figure 2. Message routing

In the Algorithm 3.2, at the node X receiving the event e, the event will be send toward Tree S_i* (Line 3-5) if the node X does not belong to Tree S_i*. Otherwise, the event will be stored and processed in the node X (Line 17) if it belongs to the zone $kzone(X)$ of node X, or send to child nodes of node X (Line 15-16) if it belongs to the zone $kzone(Y)$ of the child node of node X, or to neighbor nodes (Line 10, 11), or to

parent node (Line 12, 13) of the node *X* according to the largest common prefix (Line 7-9).

As shown in Fig. 2, based on our proposed routing algorithms, the subscription "01" is disseminated though 13→6→7 and 13→6→2→1→4 respectively; and the event "01" is disseminated through 21→14→7→6→13 and 18→9→4→1→2→6→13 respectively. Node 7 (with the common prefix 01) is the rendezvous node in the tree rooted from node 6, and node 4 (with the common prefix 01) is the rendezvous node in the tree rooted from node 9.

IV. EVALUATION STUDY

We use OPNET simulator to evaluate TinyMQ performance. The simulated network consists of 1500 sensor nodes uniformly placed in a 500m × 200m field, each node having a communication radius of 20m.

The event space is 2-dimension and any event in this space can be mapped into a bit string whose length is less than 128. Each subscription is a random continuous event set in this event space. Every node in the network can subscribe and publish.

In the simulation, 10,000 events and 100 queries are generated and disseminated. The length of the query interval (determining the size of event set) follows Zipf's distribution. The reason to choose Zipf's distribution is that recent work observed a Zipf like long tail distribution [11] of object annotation and the query terms in ad hoc, self-organized applications like WSN applications.

To evaluate the efficiency of TinyMQ, we consider the following performance metrics: subscription efficiency, notification delay and effect of failure. We also compared TinyMQ with PUB-2-SUB$^+$ in terms of subscription efficiency and notification delay. PUB-2-SUB$^+$ is selected because it is also a content-based subscribe/publish middleware built on a logic overlay without location information.

To measure subscription efficiency, we compute the average number of nodes that store a query and the number of hops that the query is forwarded until reaching those nodes. Fig. 3 and Fig. 4 show TinyMQ achieves higher subscription efficiency both in terms of storage and communication especially when the number of trees is larger. In Fig. 3, the x axis represents the number of trees, and the y axis represents the number of nodes storing the query. In Fig. 4, the x axis represents the number of trees, and the y axis represents the number of hops that the query is forwarded. For example, when the number of trees is 10, the average number of nodes storing the subscription decreases from 23 to 18, and the average number of hops forwarding the query decreases from 18 to 10. It is because the trees are not overlapped in TinyMQ and distance-based inter-tree routing can balance the load.

Notification delay is computed by the equation $d_t/d_{optimal}$, where d_t is the total hop count distance from the source node to the destination node via the rendezvous node, and $d_{optimal}$ is the shortest hop count distance from the source node to the

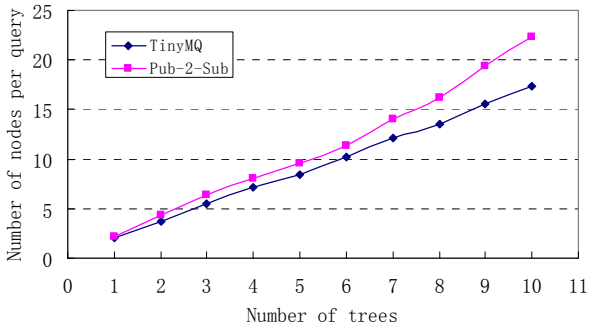

Figure 3. Number of nodes per query

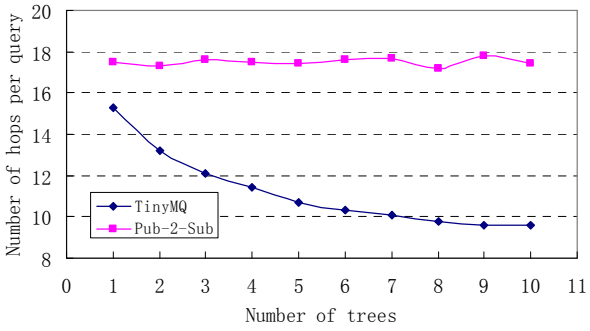

Figure 4. Number of nodes per query

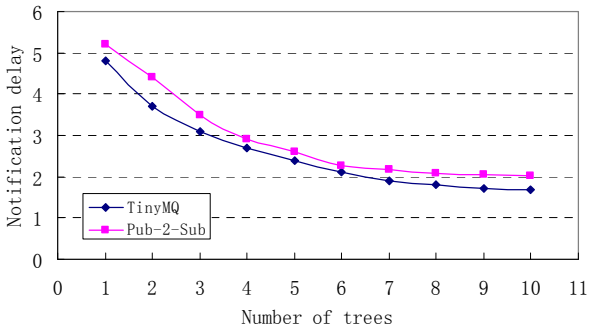

Figure 5. Notification delay

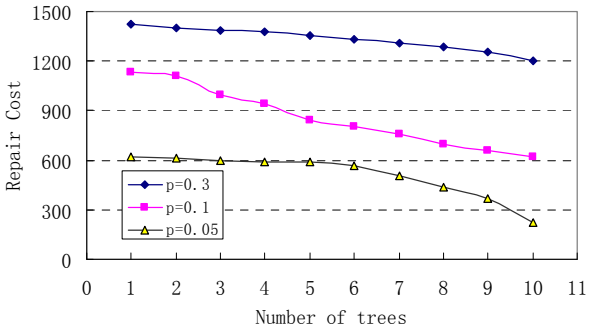

Figure 6. Repair cost

 ISBN: 978-1-61208-144-1

destination directly. Fig. 5 illustrates notification delay of TinyMQ is lower than PUB-2-SUB$^+$. The x axis represents the number of trees, and the y axis represents the value of $d_t/d_{optimal}$. Since both TinyMQ and PUB-2-SUB$^+$ do not create new network links, $d_{optimal}$ value should be same. The main reason for this improvement in notification delay is that the numbers of hops per query is lower in TinyMQ.

To evaluate the effect of failure, we let a random fraction p of the network down and investigate the impact in terms of repair cost which is defined that the total number of nodes that need to update its state to remain valid. Fig. 6 illustrates the repair cost versus different number of failing nodes. The x axis represents the number of trees, and the y axis represents the number of nodes that must update their status. Since TinyMQ needs to maintain the distance information, most nodes need to update its information. When the number of trees increase, the network partitioned into more small clusters, the number of affected nodes decreases due to this separation.

V. CONCLUSIONS AND FUTURE WORKS

Without location information the dissemination of queries and events in publish/subscribe services in sensor networks usually relies on a gossiping protocol which ignores the message content during the dissemination. Our proposed middleware, TinyMQ, provides an efficient content-based publish/subscribe architecture in WSNs. TinyMQ is based on a partition of the network, and nodes in each tree are assigned unique key values in such a way, that is convenient for content-based routing. Our simulation results have demonstrated several promising properties of TinyMQ in terms of subscription efficiency and notification delay.

Future work includes implementing and deploying TinyMQ in real sensor nodes such as MicaZ or IRIS. The issues of optimizing multi-tree based logic topology according to the network environment and studying the impact of node mobility will be also addressed.

ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China under Grant No.50805058, the Fundamental Research Funds for the Central Universities (HUST 2010MS083), and International S&T Cooperation Program of Hubei Province under Grant No. 2010BFA008.

REFERENCES

[1] T. He, et al., Vigilnet: an integrated sensor network system for energy-efficient surveillance, ACM Transaction on Sensor Networks, vol. 2, no. 1, pp. 1–38, 2006.

[2] L. Fiege, M. Cilia, G. Mühl, and A.P. Buchmann, Publish-subscribe grows up: Support for management, visibility control, and heterogeneity, IEEE Internet Computing, vol. 10, no. 1, pp. 48–55, 2006.

[3] J.-H. Hauer, V. Handziski, A. Köpke, A. Willig, and A. Wolisz, A component framework for content-based publish/subscribe in sensor networks, in: Proc. of the 5th European Workshop on Wireless Sensor Networks, pp. 369-385, 2008.

[4] M. Rudafshani and S. Datta, Localization in wireless sensor networks, in: Proc. of the 6th International Conference on Information Processing in Sensor Networks, pp. 51–60, 2007.

[5] P. Costa, G.P. Picco, and S. Rossetto, Publish-Subscribe on Sensor networks: a semi-probabilistic approach, in: Proc. of the 2nd IEEE International conference on Mobile Ad-hoc and Sensor Systmes, pp. 322-332, 2005.

[6] E. Souto, et al., Mires: a publish/subscribe middleware for sensor network, in: Proc. of the 2nd IEEE International conference on Pervasive and Ubiquitous Computing, pp. 37-44, 2006

[7] P. Boonma and J. Suzuki, Toward Interoperable Publish/Susbscribe Communication Between Wireless Sensor Networks and Access Networks, in: Proc. of International Workshop on Information retrieval in Sensor networks, pp. 1-6, 2009.

[8] D.A. Tran and C. Pham, PUB-2-SUB: A Publish/Subscribe Content-based Framework for Cooperative P2P Networks, in: Proc. of the 8th IFIP Networking Conference, pp. 770-781, 2009.

[9] D.A. Tran and C. Pham, A Content-Guided Publish/Subscribe Mechanism for Sensor Networks without Location Information, Journal on Computer Communications, vol. 33, no. 13, pp. 1515-1523, 2010.

[10] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg, Content-Based Publish-Subscribe over Structured Overlay Networks, in: Proc. of the 25th IEEE International Conference on Distributed Computing Systems, pp. 1-10, 2005.

[11] W. Acosta and S. Chandra, On the need for query-centric unstructured peer-to-peer overlays, in: Proc. of the 5th IEEE International Workshop on Hot Topics in Peer-to-Peer Systems, pp. 1-8, 2008.

 ISBN: 978-1-61208-144-1