

iBubble: Multi-keyword Routing Protocol for Heterogeneous Wireless Sensor Networks

Xiaoming Lu*, Matt Spear*, Karl Levitt* and S. Felix Wu*

*Department of Computer Science

UC Davis, Davis, CA 95616

Email: {lu, spearmat, levitt, wu}@cs.ucdavis.edu

Abstract—Many tasks require multiple sensing capabilities; in Wireless Sensor Networks (WSN), it is expensive to deploy a homogeneous network wherein every sensor has the same functionality. Instead, it is economical to deploy a heterogeneous network wherein sensors differ in their capabilities; in such a network, efficient data querying is essential. We propose a multi-keyword routing protocol, iBubble, for Heterogeneous Wireless Sensor Networks (HWSN) where keywords describe sensor functionalities. iBubble provides an efficient query interface for locating data; queries are routed only along paths with nodes matching the query. iBubble utilizes an *intelligent bubbling* mechanism to propagate keywords to the Base-Station (BS). The keywords are aggregated via a novel use of lattices to reduce network cost. We show that iBubble can emulate diffusion and generally produce less traffic by restricting the query dissemination based upon both application type and data value. Our study analytically compares iBubble and diffusion, and formally characterizes the conditions required for iBubble to outperform diffusion in both static (fixed) and dynamic (mobile) networks. We did extensive simulations, our results match our theory and show that iBubble can outperform diffusion in many heterogeneous deployments when keyword distributions are “clustered” enough to satisfy the fraction of the network involved in a query/update by our analytical bound. Additionally, iBubble handles mobility, fault-tolerance, and provides network diagnosis via keyword bubbling. By utilizing keywords, iBubble bridges many routing and energy problems prevalent in WSN, and provides a simple, uniform solution.

I. INTRODUCTION

Wireless Sensor Networks (WSN) are ideal candidates for environmental monitoring, and can be utilized for many applications from disaster recovery to military surveillance. WSN deployment can be of two kinds. The first kind connects numerous identical, possibly multi-purpose, polyfunctional sensors into a homogeneous network. Such a WSN is unlikely to be realized since the sensors would be bulky and expensive. The second kind also utilizes many sensors, but they are relatively inexpensive where different sensors have different functionalities. Each sensor will have a well defined, limited mission. Such a collection of diverse sensors would comprise a large Heterogeneous Wireless Sensor Networks (HWSN).

In Figure 1, we depict a HWSN that is deployed in a national park to provide safety and environment monitoring. Sensors of type 1 have visual detectors and are deployed near camping areas to give early warning of approaching animals. Sensors of type 2 are temperature sensors and are deployed near dry grass to monitor fire hazards. Animals tend

to congregate around the watering hole and auditory sensors, type 3, are deployed to prevent poaching. Sensors of type 4 can detect water level change and are deployed near a river which tends to flood. Finally, the Base-Station (BS) is inside the ranger’s tower, which can query or receive event-driven sensor reports.

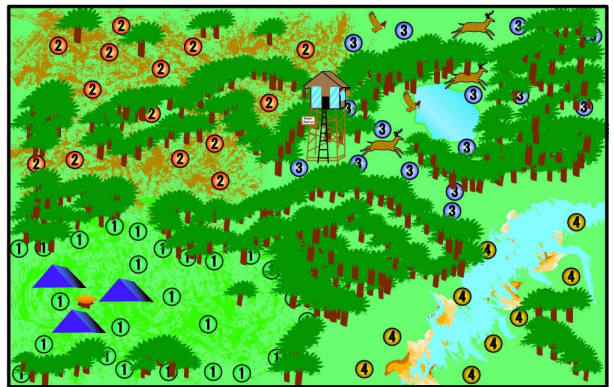


Fig. 1: A forest deployed with heterogeneous sensors to monitor different types of events.

Current WSN often assume a single application running on every node, and therefore are node-centric [1], [2]. With HWSN, searching for sensors that handle a specified application is essential. This requires a data-centric design; much like data searching on the Internet, efficient query handling is challenging.

We propose a novel multi-keyword routing protocol (iBubble) for HWSN. Each node publishes a set of keywords describing its abilities, and these are “bubbled up” towards the BS to allow efficient data and application searching. The keywords can be anything, e.g. types of sensing devices, energy level, sensing device health, or data value descriptions. Once the network has stabilized, the BS can issue a query to find any data subset. For example, the BS could query “water level sensors” and “temperature sensors \wedge temp $>$ 100”. The keywords published by the sensors will help direct the BS’s query towards the nodes with data or have data downstream matching the query.

To minimize the cost of bubbling keywords up to the BS, iBubble aggregates keywords. The aggregation algorithm *preserves information* via a novel application of lattices. This

property allows routing to be more intelligent and application-aware. The network *hardly* needs to be flooded by a query; only those nodes that possess data or have the data downstream which matches the query will handle/forward that query.

At any point during the lifetime of the network, any node may change its keyword set by adding or removing keywords. These changes are bubbled to the BS only on the paths that the node will use for communication. This allows iBubble to handle mobility implicitly.

A unique property of sensor networks is their reliance on batteries; nodes which have higher throughputs or computation times will die earlier. WSN are dynamically changing (due to non-uniform energy consumption [3]) and unplanned mobility (due to either weather, wild animals, or malicious attackers). iBubble provides self-healing by using keywords. Once a sensor detects that it is in a dysfunctional state, it can publish a keyword to inform its neighbors or the BS to take appropriate actions.

iBubble provides a mechanism for communication between the BS and its nodes, but does not address peer-to-peer communication. Using query architecture, iBubble can be used to receive interest-driven, event-driven or periodic data.

We compare iBubble with diffusion, another data-centric routing protocol, and show that in most cases iBubble outperforms diffusion under most heterogeneous deployments. We analytically compare iBubble with diffusion, and formally characterize the conditions required for iBubble to outperform diffusion in both static (fixed) and dynamic (mobile) networks. We performed extensive simulations. Our results match our theory and show that iBubble can outperform diffusion in many heterogeneous deployments when keyword distributions are “clustered” enough to bound the fraction of the network involved in a query/update. For example, in the static case, with 26 keywords in the network, iBubble outperforms diffusion whenever the fraction of the network involved in a query is less than 96%. iBubble provides a simple framework to efficiently execute queries, handle mobility/fault-tolerance, and perform network-health monitoring.

The rest of the paper is organized as follows: we discuss the design objectives of iBubble in Section II; the existing work related to data-centric routing is discussed in Section III; the protocol is described in Section IV. We compare iBubble with diffusion analytically in Section V. In Section VI, we evaluate iBubble by extensive simulation and analyze its security issues. Finally we conclude in Section VII.

II. DESIGN OBJECTIVES

In this section, we describe the design objectives of iBubble with an emphasis on the features that allow iBubble to provide efficient query and self-healing in a HWSN.

The main objectives in the design of iBubble are:

- **Provide efficient data-centric routing.** iBubble constrains queries to only the nodes which have matching data or have matching data downstream.
- **Support a low sink to source ratio.** iBubble is designed to handle a small set of static sinks (BS). Although we

present iBubble as using a single BS, it can be trivially extended to handle a small set of BS using designs such as: (1) nodes communicate only with the closest BS, or (2) nodes maintain separate neighbor-tables for each BS.

- **Provide aggregation mechanism to minimize bubbling cost.** iBubble aggregates via a novel application of a lattice to minimize the cost of keyword bubbling.
- **Support both static and mobile HWSN.** iBubble is efficient under both static and dynamic settings. This is achieved by having nodes bubble keywords to the BS whenever their location changes.
- **Provide simple fault diagnosis and self maintenance.** iBubble handles fault diagnosis by having dysfunctional nodes bubble keywords up to the BS which are related to their fault. Furthermore, iBubble excludes faulty nodes when necessary.
- **Remove reliance on global identifiers.** iBubble has no reliance on global identifiers; instead, nodes can be identified via their keyword set and the path to the BS. This diminishes the value of node impersonation to would-be attackers.

III. RELATED WORK

The emphasis on data-centric routing protocols emerges out of HWSN which deploy multiple sensing devices and are not necessarily task-specific. There has recently been a move to provide data-centric routing protocols [4], [5]. The goals of these protocols are to provide a query interface that allows nodes to find all other nodes with a specified property.

As diffusion’s data-centric routing protocol is closely related to our work, we go into some detail describing its operation, and discuss some of its variations. Diffusion was first introduced in [4]; nodes keep a set of attribute-value pairs describing the services they provide. Nodes which publish data are entitled *sources*, and nodes which request data are called *sinks*. When a sink wishes to subscribe to some set of data, its initial query establishes a gradient between itself and the sink. The exact mechanism of gradient establishment is dependent on the protocol.

Directed diffusion [4] uses a 2-phase-pull mechanism to establish the gradient. First, the sink floods the network with a query describing its interest with a low period and any node which receives the query will reply using flooding if their data matches the query, establishing a gradient. Flooding in both directions removes the reliance on symmetric links, and allows the sink to choose the optimal set of routes. Once the gradient has been established both the query and the data will utilize the same path. If the node moves, the sink must reflood the network to establish a new gradient.

Recently, [5] introduced a 1-phase-pull and push incarnation of diffusion, and compared the various diffusion protocols. The 1-phase-pull operates similarly to 2-phase-pull, except instead of flooding from source to sink, the reverse path is always used. In push diffusion, the sources push the data information to the sinks via flooding, and the sinks rely on the reverse paths as gradients to the nodes which have data matching their

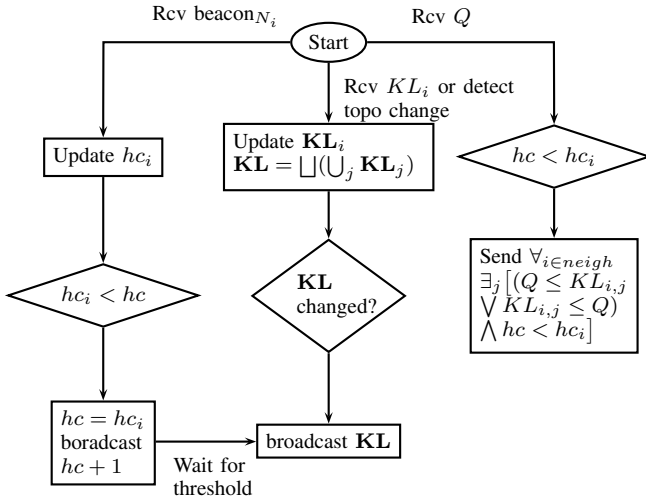


Fig. 3: iBubble’s algorithmic state machine

interest. Push and pull can be switched adaptively based upon network query count using the method in [6].

There has been some work on restricting the range of flooding by utilizing a set of rules [3], [7]–[11]. In particular, Geographic and Energy Aware Routing (GEAR) restricts the query based upon geographic constraints. When a node receives a query, the next-hop node is chosen to minimize the cost of sending; cost is calculated across both geographic distance and the energy of the neighboring nodes.

Keyword-based routing was introduced in Davis Social Links (DSL) [12]. In DSL, keywords are also propagated out, but there is no directional bearing; keywords must be sent in all directions as the goal is to provide peer-to-peer communication. Aggregation is mentioned, but there is no concrete mechanism given as to how to combine keywords. Some of the high-level goals of iBubble are shared with DSL—although DSL is presented as an abstract protocol and is not adapted for sensor networks. Rather than simply applying keyword to routing, we demonstrate in this paper how to use it advantageously in WSN.

IV. PROTOCOL DESCRIPTION

In this section, we provide the details of iBubble. We first give an overview of the actions of iBubble, then discuss the keyword aggregation mechanism, then we discuss how iBubble flexibly handles faults or mobility, and finally we give a complete example.

A. Overview

iBubble has three main actions: (1) initialization, (2) bubbling, and (3) querying. The algorithmic state machine of iBubble is shown in Figure 3, although we omit that all end states loop back to start. The initialization sets up the attributes that are required to control the keyword propagation to the BS. The bubbling phase aggregates the keywords and sends them towards the BS. Finally, the query phase allows the BS to interact with the network in order to find data.

1) *Initialization*: For initialization nodes *must* determine a vector to the BS (their hopcount). The BS waits until all nodes have booted and then broadcasts a hopcount beacon containing the hopcount value 1 ($hc_1 = 1$). When a node N_k receives a beacon containing the hopcount h from neighbor j , it will update its neighbor table with $hc_j = h$. If the value in the beacon is less than the node’s hopcount ($hc_j < hc$), the node will update its hopcount with the smaller value and broadcast one more than its hopcount ($hc+1$). Once a node has updated its hopcount it waits a period of time, which depends on its hopcount, to broadcast its keyword set. The goal of the delay is that nodes which are further away will broadcast their keywords earliest, and nodes closer to the BS will broadcast their keywords later to minimize the bubbling cost. Figure 2(b) shows an example network once the beacon has propagated throughout the network.

2) *Bubbling*: Bubbling establishes the **KL** for every node. Whenever the keyword set for a node is changed it must bubble the new keyword set to the BS, and this change may be because of initialization or because a node’s downstream neighbor updated its keyword set. When a node receives a keyword set (**KL**), it updates its neighbor’s keyword list to be **KL**. Once the node has updated the keyword set of its neighbor, it sets its keyword list to be $\sqcup(\cup_j \mathbf{KL}_j)$, where j ranges over all neighbors with a greater hopcount and the node itself, and broadcasts this to all neighbors with a lower hopcount. It is important to note that we use \sqcup to denote our aggregation function as it is closely related, but not equivalent, to the least-upper-bound. Utilizing the hopcount as a directional vector prevents loops and restricts the keyword bubbling towards the sink.

Internally, each node tracks three keyword sets: (1) personal, (2) published, and (3) all immediate neighbors. The personal keyword set specifies the keywords describing the node. The published keyword set is the aggregation of all its personal and downstream neighbor’s keyword sets. This is utilized for upstream nodes to determine how to forward queries downstream. When a node publishes a keyword set, it marks its personal keyword set to allow neighbors to make proper re-routing decisions. A node maps a local identifier for each neighbor their keyword set and uses it to make routing decisions.

3) *Query*: Queries are always initiated by the BS and propagated throughout the network utilizing the published keyword sets. A query message contains three fields: (1) **Query**, (2) **AppTest**, (3) **UID**. The **Query** (Q) is a set of conjunctions and disjunctions of elements of the keyword lattice, specifying high level query types. **AppTest** is a set of conjunctions/disjunctions of binary operations with the data values to refine the query. The purpose of **Query** is to restrict the flooding out from the BS, whereas, **AppTest** is used to reduce the number of responses when the BS is not interested in all data values. The **UID** is the unique id of the query and a node will not send data to the BS if it has already sent the data matching the unique id.

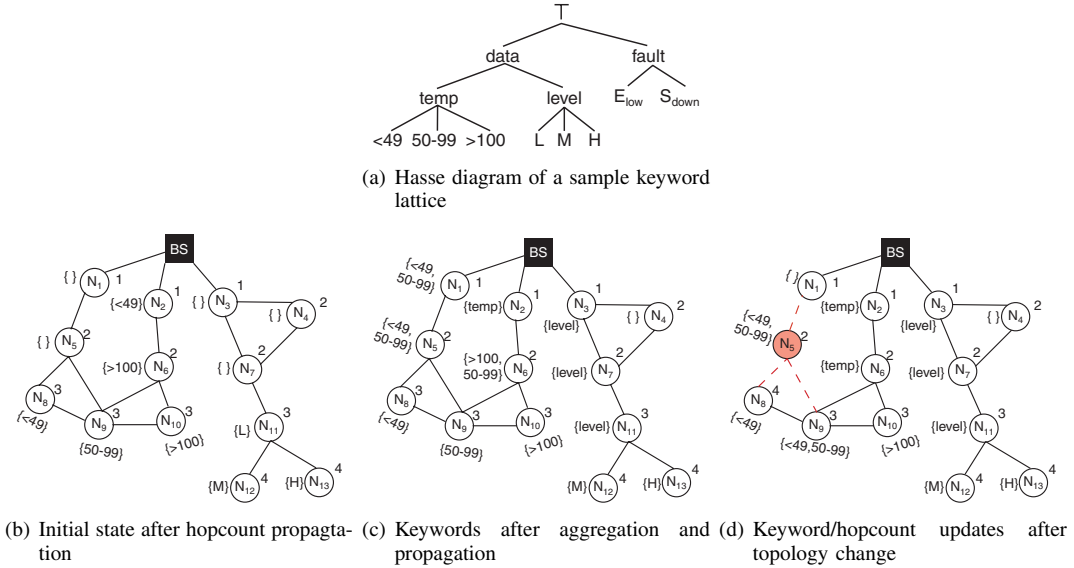


Fig. 2: An example keyword lattice and graphs demonstrating keyword propagation and re-routing

A node only forwards data if there exists a neighbor i with a larger hopcount (hc_i) and a keyword j satisfying the **Query** Q :

$$\exists_j [(Q \leq KL_{i,j} \vee KL_{i,j} \leq Q) \wedge hc < hc_i]$$

and the **Query** has not been seen before. The reasoning behind this definition will be elaborated in Section IV-B. When a node has a keyword matching the query, it will reply on the reverse path of the query.

This allows all three methods of query sending. For random the BS *must* send a query whenever it wishes to get the data. Event-driven is accomplished by having the BS broadcast an initial query specifying the kinds of data to match, and the receiving nodes will record the reverse path; whenever a node's sensing device produces data matching the query, the sensed data will be returned to the BS along these paths. Finally, with periodic there are two mechanisms: (1) the BS can broadcast a query each unit of time, or (2) the query can specify a unit of time for the period.

B. Lattice & Aggregation

Aggregation is essential to minimize the number of keywords that are sent whenever an update occurs. iBubble utilizes a lattice specifying a complete partial order (c.p.o). A c.p.o is an ordering, such that every chain has a supremum (least-upper-bound). For iBubble, it is also very preferable that every element of the powerset of the keywords has a least-upper-bound. To minimize flooding, it is important that the aggregation is *information-preserving*. By abuse of notation, we denote the aggregation function as \sqcup as it is closely related to the least-upper-bound. The function only chooses the least-upper-bound element when all its children are present in the set, otherwise it yields the keywords.

Throughout the paper we depict lattices using Hasse diagrams, in which elements are connected if they satisfy $x \leq$

$y \wedge \exists_z x \leq z \leq y$. That is x is below y and is connected if and only if $x \leq y$ and there is no element z such that $x \leq z \leq y$. As an example of using the c.p.o depicted in the Hasse diagram in Figure 2(a) we compute \sqcup as follows:

$$\begin{aligned} <49 \sqcup 50-100 &= \{<49, 50-100\} \\ <49 \sqcup temp &= \{temp\} \\ data \sqcup fault &= \{\top\} \\ \sqcup \{<49, 50-100, L, M, H\} &= \{<49, 50-100, level\} \end{aligned}$$

The lattice construction serves two purposes: (1) $Q \leq KL_{i,j}$ allows nodes to forward when keywords have been aggregated, e.g. $<49 \leq temp$ and a node can only have $temp$ if it has <49 downstream; and (2) $KL_{i,j} \leq Q$ allows the BS to keep its queries smaller, e.g. the BS could query $Q = \{temp\}$ instead of $Q = \{<49 \vee 50-100 \vee >100\}$. As not every element is comparable, this accomplishes a minimal flooding, e.g. $<49 \not\leq L$ and $L \not\leq <49$.

C. Fault Tolerance & Mobility

iBubble provides fault-tolerance and mobility via a number of mechanisms: (1) redundant paths, (2) re-routing, and (3) keyword set updating. As keywords are propagated along *all* neighbors with lower hopcount, there is a probability of redundant paths. It should be noted that a node's hopcount is the minimum of its neighbors' hopcount plus one. The use of redundant paths prevents a fault along one path from keeping queries from the source and allows a source to choose which one to use for reply. Furthermore, when a fault is detected and there is no redundant path with a lower hopcount, a node will communicate with its neighbors to update its hopcount and route through its new neighbors. When a node fails or moves, the keyword sets of all nodes on the path must be updated. To accomplish this, when a node determines that

another node is no longer replying to queries, it removes that node from its neighbor set and updates its published keyword set. If the keyword set changes, then the node rebroadcasts its keyword set. The addition of a node is handled in the same way: (1) a node determines its new hopcount, (2) it determines its neighbors keyword sets, and (3) it broadcasts its keyword set.

D. Complete Example

In this section we provide a complete example showing the three phases of iBubble and how re-routing works. Figure 2(a) shows the lattice used for the example. Figure 2(b) depicts the network state after the beacon has propagated throughout the network and hopcounts are determined, but before keywords are exchanged. The hopcount for every node is displayed to the right of the nodes and the keyword set is displayed on the left. Some nodes do not yet have a keyword and therefore start with an empty set.

To save time, the nodes which have highest hopcount should broadcast their keywords first. For instance, N_{12} and N_{13} will send to N_{11} their keys, and N_{11} will compute $\lfloor \{L, M, H\} = level$ and will propagate this to N_7 , N_3 , and the BS. Note that N_4 has no keywords after the bubbling phase since it has no downstream neighbors with a larger hopcount. The other branch bubbles similarly. The completion of the bubbling phase is depicted in Figure 2(c).

The BS will send queries once the network has stabilized. For instance, the BS could issue **Query:50–99**; this query will travel along paths N_1, N_5, N_9 and N_2, N_6, N_9 . As an example, N_6 forwards the query to N_9 as $50-99 \leq 50-99$, but not N_{10} as keyword 50-99 and >100 is not comparable using c.p.o defined by the lattice. As N_9 receives the query from two different paths it will use the lowest latency path as the return path.

Finally, to show how re-routing occurs, assume N_5 becomes faulty as in Figure 2(d). When N_8 detects that N_5 is faulty or has moved, it updates its own hopcount to one more than its next lowest neighbor's hopcount, which in this case is N_9 . Once N_8 updates the hopcount, it broadcasts its keyword set, and this is propagated up towards the BS—hence N_6 's published keyword becomes *temp*. When N_1 detects that N_5 is gone, it will delete all the keywords bubbled from N_5 's and will switch back to empty set and broadcast this to BS. When the BS issues a query for *temp* or <49 it will be forwarded to N_2 only since only N_2 has keywords matching the query. The response will travel along the path N_8, N_9, N_6, N_2 to the BS, and the network will be stabilized again. The final state is depicted in Figure 2(d).

V. COMPARISON OF IBUBBLE WITH DIFFUSION

In this section, we compare the expressiveness of the query language and the amount of traffic generated between iBubble and diffusion analytically. We show that iBubble's query language is at least as expressive as diffusion, and then show that iBubble outperforms diffusion for most keyword distributions.

A. Language Expressiveness

To compare expressiveness we show how iBubble can emulate any query in diffusion. In diffusion queries are composed of a set of attribute-value pairs, whereas in iBubble queries consist of a set of conjunctions/disjunctions for **Query** and **AppTest**.

iBubble can emulate diffusion by setting **Query** to \top and placing all the attribute-value pairs into the **AppTest**. If one is aware beforehand that one is interested in specific subsets of data values, then one can move some subsets into the lattice using iBubble. Hence query flooding can be controlled not only by application type, but also by data values.

Figure 4 shows an example of a diffusion query and two emulation possibilities using iBubble. In Figure 4(a), the diffusion query is for a temperature sensor with a reading between 100 and 150 in region 1; Figure 4(b) shows how one might construct a minimal lattice and the resulting query. Finally, in Figure 4(c) the lattice is designed to contain subsets, and the query is able to further restrict the flooding.

B. Amount of Traffic

In this section we compare the amount of traffic generated by iBubble and diffusion. There are three kinds of diffusion: 1-phase-pull, 2-phase-pull and push [5]. Since the only difference between 1-phase and 2-phase pull is that 2-phase pull uses flooding to reply when a keyword is first queried and thus it is more expensive than 1-phase pull; therefore, we omit 2-phase pull in comparing diffusion with iBubble. However, it should be noted that as 2-phase pull floods twice the sink has more flexibility to choose a more optimal route than simply using the shortest/lowest latency path as required by 1-phase pull. Since our assumption is that there are many sources and few sinks we omit push diffusion, as it operates more efficiently than 1-phase only when there are more sinks than source nodes [5]. It should be noted that due to limited space, we omit the proofs that 2-phase-pull is more expensive than 1-phase-pull and 1-phase-pull has the same equation as push diffusion.

We will compute the upper-bound on the amount of traffic for 1-phase diffusion style and for iBubble. We use the following notations:

N	The number of nodes
m	The number of keywords. Assume $m > 0$
M	The number of times a node moved
T	The total number of queries. Assume $T > M + m$
x_j^q	The total number of transmissions for querying keyword j
x_j^r	The total number of transmissions for replying to keyword j
$x_i^{o,n}$	The total number of transmissions for removing N_k 's keywords from old path and adding it to new path to BS in the i^{th} round where k is the index of the node that moved in the round

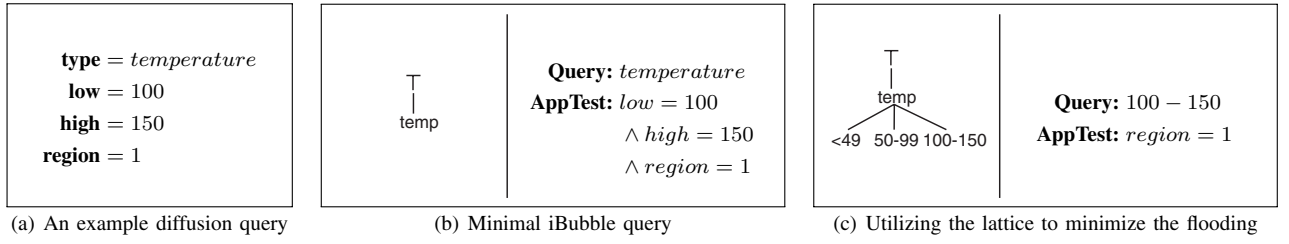


Fig. 4: An example of emulating a diffusion query in iBubble, the Query and AppTest depend heavily on the lattice design.

- $K_{i,j}$ $K_{i,j}$ the number of times the j^{th} keyword is queried in the i^{th} round. Assume: $\forall_i \leq M, \sum_{j=1}^m K_{i,j} > 0$
- c $0 \leq c \leq 1$ the average fraction of the network touched over all queries
- d $0 \leq d \leq 1$ the average fraction of the network updated over all moves

To provide a fair comparison we give the sink in 1-phase pull an oracle that allows it to know when a node moved, otherwise 1-phase pull would have to flood most messages to ensure it received all matching data as it would be unable to trust the established gradients. We also assume that 1-phase pull does m initial queries before movement begins and queries at least once per movement. Then the number of message transmissions is given by:

$$T_{1p} = \underbrace{mN}_{\text{initial flood}} + \underbrace{\sum_{j=1}^m K_{i,j} x_j^r}_{\text{queries/replies by}} + \underbrace{MN + \sum_{i=1}^M \sum_{j=1}^m K_{i,j} x_j^r}_{\text{queries/replies by flood due to mobility}} + \underbrace{\sum_{i=1}^{T-M-m} \sum_{j=1}^m K_{i,j} \cdot (x_j^q + x_j^r)}_{\text{queries/replies along gradient}} \quad (1)$$

This is a lower bound on the number of messages due to mobility as MN assumes only one query per round of movements.

iBubble has a total number of messages exchanged of:

$$T_{iB} = \underbrace{N}_{\text{initial bubble}} + \underbrace{\sum_{j=1}^M x_i^{o,n}}_{\text{bubble due to mobility}} + \underbrace{\sum_{i=1}^T \sum_{j=1}^m K_{i,j} (x_j^q + x_j^r)}_{\text{queries/replies along gradient}} \quad (2)$$

Initially every node broadcasts exactly once to its upstream neighbors and, therefore, the total number of message transmission is N . In both static and mobile cases, keywords are bubbled towards the BS whenever there is a topology change. Queries are never flooded because they use the established keyword set. It should be noted that we skip the beacon initialization cost (N) for iBubble as this is not required for all deployments, for example if the hop-count is known a priori or sensors have GPS units and use their values for the vector.

Subtracting (2) from (1) compares the amount of traffic, iBubble will do better whenever the subtraction result is

positive:

$$T_{1p} - T_{iB} = MN - \sum_{i=1}^M (x_i^{o,n}) + (m-1)N - \sum_{i=1}^{M+m} \sum_{j=1}^m K_{i,j} x_j^q \approx N \cdot (M + m - Md - Mc - mc - 1) \quad (3)$$

$$\xrightarrow[\text{static}]{M=0} c < 1 - \frac{1}{m} \quad (4)$$

$$\xrightarrow[\text{dynamic}]{M>0} \begin{cases} c < 1 - d < 1 - \frac{1+Md}{M+m} & \text{if } d > \frac{1}{m} \\ c < 1 - \frac{1}{m} \leq 1 - \frac{1+Md}{M+m} & \text{if } d \leq \frac{1}{m} \end{cases} \quad (5)$$

Equation (3) is obtained by observing that only a fraction (c) of the network transmits for every query, and only a fraction (d) is updated on every movement. It should be noted that all the reverse path traffic (x_j^r) is canceled out, which shows that we do not have to consider the query response when comparing diffusion with iBubble.

When the network is static ($M = 0$), iBubble outperforms diffusion whenever (4) is satisfied. Intuitively this implies that when the m keywords in the network exhibit ‘‘clustering’’ (nodes with similar keywords lie close geographically), then the query coverage will be small. Even with a relatively high amount of random distribution, iBubble will generally outperform diffusion. For example, when $m = 10$ then $c < 90\%$ will allow iBubble to use fewer total message transmissions than diffusion. Our simulation in Section VI qualitatively validates this theory. Furthermore, there is a decrease in percent savings with increasing c .

However, when the network is dynamic with $M > 0$ moves throughout network lifetime, iBubble will outperform diffusion when (5) is satisfied. When $d > \frac{1}{m}$, the condition $1 - d$ is sufficient, but *not* necessary, since even if $c + d > 1$ iBubble could still possibly outperform diffusion. Intuitively, as more of the network is updated on a move (d), the less of the network that can be used in a query and vice-versa. In contrast when $d \leq \frac{1}{m}$, $c < 1 - \frac{1}{m}$ is sufficient for iBubble to outperform diffusion. Since $c < 1 - \frac{1}{m}$ can be easily achieved as explained in the static case, in our simulation we concentrate on $c < 1 - d$ case. In a clustered environment, d will be 0 when a node moves within the cluster, but if a node moves out of the cluster $d > 0$, c will have to decrease accordingly for iBubble to outperform diffusion. In Section VI, we qualitatively show that there is indeed a direct correlation between increasing

$c + d$ and decreasing message transmission percent savings.

In summary, iBubble almost always outperforms diffusion in a static network, and outperforms diffusion in most dynamic situations.

VI. EVALUATION

In this section, we evaluate the performance and security issues of iBubble. First, we compare iBubble with 1-phase-pull diffusion in terms of transmission counts with and without iBubble’s initial bubbling phase. We chose 1-phase-pull for comparison between diffusion and iBubble, since 1-phase-pull applies to applications where there are fewer sinks than sources following the assumption of iBubble and it is less expensive than 2-phase-pull diffusion. We did not need to compare query response traffic, since the response traffic is identical as we showed in Section V. Finally, we compare iBubble’s number of keywords exchanged under three different topologies to show the importance of keyword aggregation. Finally, we provide a security analysis of iBubble; while the current design did not focus on security it is important to understand the shortcomings for future research.

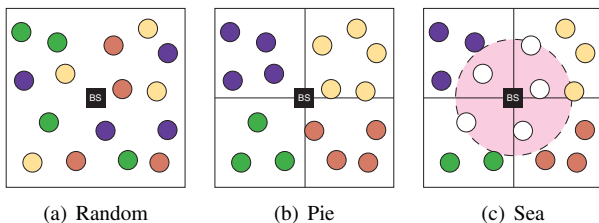


Fig. 5: The three key-distribution schemes

In iBubble, nodes rarely use same-hopcount neighbors (siblings) to communicate with BS unless under the presence of fault or malicious nodes; instead communications relies heavily on upstream and downstream neighbors. This kind of communication achieves great deal of traffic reduction in a network with one or a small number of base-stations. Figure 6 shows that the average neighbors per node (average node degree) increases when more nodes are deployed, but the average sibling neighbors per node increases more rapidly than the up or downstreams. Since the average number of up and downstreams per node in the network is the same, we use one line to represent both. When comparing iBubble with

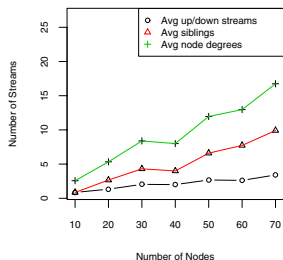


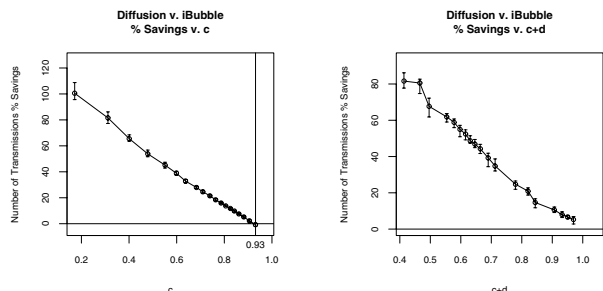
Fig. 6: Stream Utilization Study

diffusion in terms of amount of traffic generated, we found

that in most cases iBubble does much better than diffusion, and the results conform to our theoretical analysis in Section V. Our simulation also shows that whenever the number of keys per node is low, iBubble greatly outperforms diffusion.

In our simulations, nodes are uniformly distributed over a $1000m \times 1000m$ area, the BS is located at the center, the communication radius is 240m, and bidirectional symmetric communication is assumed. Figure 5 shows the three key distribution schemes we consider: (1) Figure 5(a) shows a random distribution of keys—each node independently and uniformly chooses a fixed subset of keywords (we call this *keywords per node*) from a set of keywords ($m = 26$) at the beginning of the simulation, (2) Figure 5(b) shows a pie keyword distribution of keywords, where the area is divided into pie slices and every node in a pie slice has the exact same set of keywords, and (3) Figure 5(c) shows a sea distribution of keywords, where the area is divided into slices like pie and in each slice every node has the same set of keywords, but nodes which are inside a radius of $750m$ from the BS have no initial keywords and only route messages. In all simulations, there is only one keyword per query message.

A. Diffusion Traffic Comparison



(a) savings vs. c ($M = 0$)

(b) savings vs. $c + d$ ($M > 0$)

Fig. 7: iBubble savings under different c, d

We analyzed the percent savings of transmitted messages versus c in Figure 7(a). The simulation results show that there is a linear decrease in the percent savings with increasing c . iBubble begins to lose to diffusion at $c = 0.93$, which matches closely our theory. In Figure 7(b), we compared $c + d$ (equation 5) versus percent savings of transmitted messages. Our simulation matches our theory that iBubble outperforms diffusion when $c + d < 1$. We also observe that d is often less than .1 even in a random key distribution scheme.

Figure 8 compares iBubble and diffusion with and without query setup phase. Each keyword is queried exactly once. In Figure 8(a), we compared the traffic generated by diffusion with iBubble and show the percent savings of the number of message transmissions under a static topology, *ignoring the bubbling phase*. The keywords per node denotes the number of keywords each node is initialized with. iBubble outperforms diffusion for all number of nodes and all key sizes, and it is apparent that the number of keywords per node is the dominant factor in performance. More keywords as well as

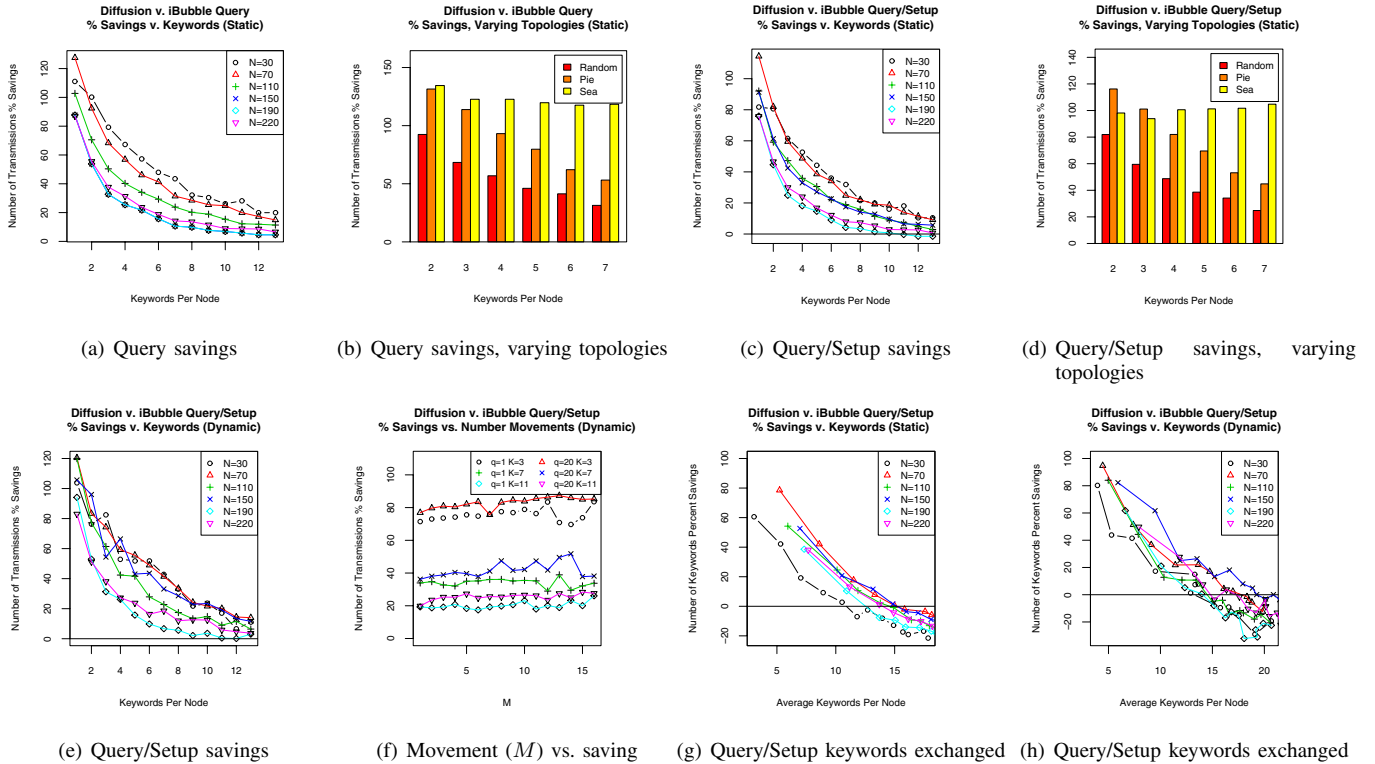


Fig. 8: Diffusion vs iBubble Traffic Analysis for Static and Dynamic Scenarios

higher node density tends to degrade iBubble’s performance. We then looked at the percent savings based upon topology, again ignoring the bubbling phase. The results are displayed in Figure 8(b). The number of keys per node at initialization varies between 2 and 7, and the number of nodes is held constant at 70. It is apparent that iBubble always outperforms diffusion, and improves when using a pie or sea topology. Sea gives the best results as the number of nodes with any given keyword will be lower.

In Figure 8(c), we compared iBubble and diffusion including the bubbling and aggregation phase under static topology. Recall that in static topology, the constraint for iBubble to win over diffusion is that $c < \frac{m-1}{m}$, where c is the fractional network touched by the query and m is the number of keywords in the network. In our simulation, $m = 26$, therefore, $c < \frac{25}{26} \approx 96\%$ by theory.

In Figure 8(e), we considered a dynamic network where the number of times a random node moved is $M = 5$ and between each move there were $q = 3$ queries. We first analyze the percent savings of transmitted messages versus the initial number of keywords per node; much like Figure 8(c) increasing the number of keys per node or the density decreases the percent savings, but iBubble never loses to diffusion over the simulations. The reason is that even though c approaches 1 d stays relatively small (always below .1)—in fact even when $c + d > 1$ by a small margin, iBubble still outperforms diffusion, albeit by a very small margin ($\approx 5\%$).

in Figure 8(f), we considered the percent savings of trans-

mitted messages versus M . Here M random nodes move, and between each move there are q queries. The simulation shows that iBubble scales well with increasing M maintaining its percent savings. Fewer queries per move yields a worse percent savings. This is due to the cost of updating paths. When there are fewer queries per move, there is less chance for iBubble to offset the cost of path updating (keyword re-bubbling) with the controlled flooding savings.

Finally, we considered the overall cost in terms of keywords transmitted as the number of keywords a message has is a good predictor of the overall traffic. In Figure 8(g) we consider the percent savings of transmitted keywords in the static scenario ($M = 0$), and in Figure 8(h) the dynamic case is shown ($M > 0$). Both diffusion and iBubble specify one keyword per query, but iBubble has extra bubbling stage where a message might contain one or more keywords per message. The x-axis shows the average number of keywords per node *after* the bubbling. Only random keyword distribution scheme is used, as nodes in a similar region share the same keys in the pie and sea distributions and thus the savings remains fairly constant for pie and sea. iBubble still outperforms diffusion when the number of keywords per node is low, but the cutoff is much lower (about 10 keywords per node in static and 15 in dynamic). The average number of keywords per node is the dominant factor as it is directly correlated to c . Both figures show that iBubble will always outperform diffusion as long as only a moderate fraction of the network needs to be involved in communication for query and bubbling.

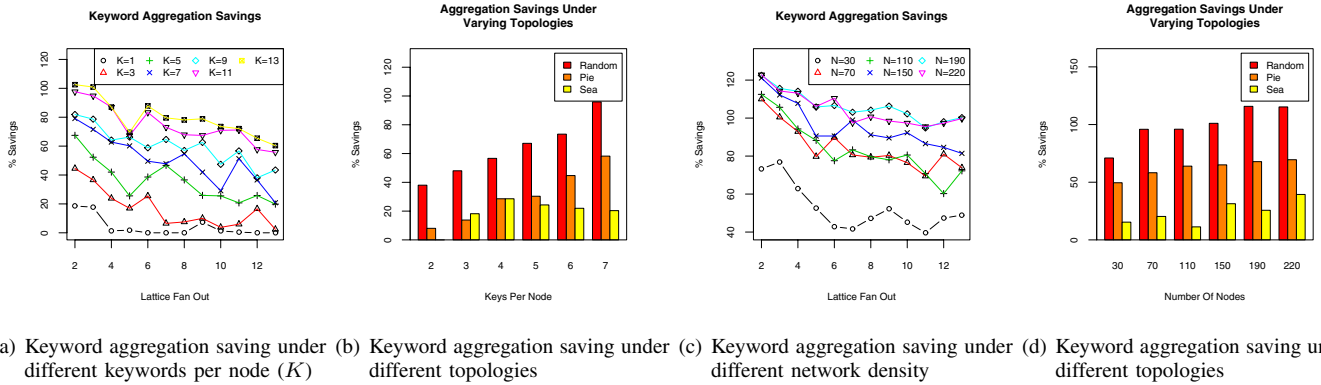


Fig. 9: iBubble Keyword Aggregation Study

B. Aggregation Savings

To preform aggregation, we construct a lattice deterministically based on the single parameter, fanout. Fanout specifies the number of children each vertex will have, and in turn, determines the lattice height ($\log_{fanout} m$). The keywords (a to z) are ordered and used as bottom leaves of the lattice. Every fanout number of keywords are set to be the children of a new vertex with a new keyword label. This process is repeated until a fanout number of nodes are combined to create the \top . This construction is not necessarily ideal, but provides a simple mechanism for generating lattices. In a real deployment, lattice design would be an essential component of the iBubble design and can be tuned for specific applications.

To show the importance of aggregation, we compared the number of keywords exchanged with and without aggregation during the bubbling phase. Figure 9(a) shows that for the random topology, if we have more keywords per node initially, aggregation yields greater savings as the likelihood that \sqcup will combine increases. Also apparent is that the larger the fanout the lower the savings, due to \sqcup requiring that all children be present under a vertex before being combined, which is unlikely with higher fanout. When keyword size is $m/2$, the lattice has only two levels, with minimum aggregation savings. Figure 9(b) shows that aggregation savings is highest for random, and that pie yields more savings than sea. The importance of aggregation with respect to network density is shown in Figure 9(c), using random keyword distribution. A higher number of nodes yields higher savings as there is a higher likelihood of keywords combining. Finally, we compared across the differing key distribution schemes in Figure 9(d). Again, random yielded the best keyword aggregation savings, an pie outperformed sea.

VII. CONCLUSION

We have proposed iBubble, a novel data-centric routing protocol that allows queries based upon a set of keywords. iBubble restricts and directs the query to where there is data and provides source mobility, fault-diagnosis, and self-healing support without flooding the network. iBubble uses a novel aggregation scheme based upon c.p.os and lattices to minimize

the cost of initialization. We compared iBubble with diffusion in terms of total traffic and query language expressiveness and showed that in most cases iBubble significantly outperforms diffusion in various heterogenous deployments. By utilizing keywords, iBubble bridges many routing and energy problems prevalent in WSN, and provides a simple, uniform solution.

REFERENCES

- [1] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, pp. 234–244.
- [2] D. Johnson, "Routing in ad hoc networks of mobile hosts," in *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.
- [3] R. Shah and J. Rabaey, "Energy aware routing for low energy ad hoc sensor networks," in *In Proc. IEEE Wireless Communications and Networking Conference (WCNC), Orlando, FL, March 2002*.
- [4] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Mobile Computing and Networking*, 2000, pp. 56–67.
- [5] J. Heidemann, F. Silva, and D. Estrin, "Matching data dissemination algorithms to application requirements," USC/Information Sciences Institute, Tech. Rep. ISI-TR-571, April 2003.
- [6] Z. Yuan-yuan, X. Bo, and Z. Zi-ming, "Adaptive directed diffusion routing in wireless sensor networks based on application," in *Electrical and Computer Engineering, 2005. Canadian Conference on, Vol., Iss., 1-4 May 2005, ages: 2147- 2150*.
- [7] K. Harras, K. Almeroth, and E. Belding-Royer, "Delay tolerant mobile networks (dtmns): Controlled flooding schemes in sparse mobile networks," in *IFIP Networking 2005*.
- [8] A. Rahman, W. Olesinski, and P. Gburzynski, "Controlled flooding in wireless ad-hoc networks," in *Wireless Ad-Hoc Networks, 2004 International Workshop on, Vol., Iss., 31, May-3 June, 2004, Pages: 73-78*.
- [9] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," in *UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0023, May 2001*.
- [10] N. B. Chang and M. Liu, "Controlled flooding search in a large network," *IEEE/ACM Trans. Netw.*, vol. 15, no. 2, pp. 436–449, 2007.
- [11] U. Lee, J. Kong, J.-S. Park, E. Magistretti, and M. Gerla, "Time-critical underwater sensor diffusion with no proactive exchanges and negligible reactive floods," in *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 609–615.
- [12] L. Banks, S. Ye, Y. Huang, and S. F. Wu, "Davis social links: Integrating social networks with internet routing," *To appear in ACM SIGCOMM 2007 Workshop on Large-Scale Attack Defense (LSAD), August 27, 2007, Kyoto, Japan*.