

Smart-Tag Based Data Dissemination

Allan Beaufour, Martin Leopold, Philippe Bonnet
Institute of Computer Science
University of Copenhagen
beaufour@diku.dk, leopold@diku.dk, bonnet@diku.dk

ABSTRACT

Monitoring wide, hostile areas requires disseminating data between fixed, disconnected clusters of sensor nodes. It is not always possible to install long-range radios in order to cover the whole area. We propose to leverage the movement of mobile individuals, equipped with smart-tags, to disseminate data across disconnected static nodes spread across a wide area. Static nodes and mobile smart-tags exchange data when they are in the vicinity of each other; smart-tags disseminate data as they move around. In this paper, we propose an algorithm for update propagation and a model for smart-tag based data dissemination. We use simulation to study the characteristics of the model we propose. Finally, we present an implementation based on Bluetooth smart-tags.

1. INTRODUCTION

Data dissemination can be defined as the proactive distribution of relevant data to large numbers of users [4]. The objective is to deliver *the right data, to the right people at the right time*. In the context of sensor networks, data dissemination is a natural basis for monitoring applications.

The mechanisms that underly data dissemination are, first, the matching of data to user interests, and second, the delivery mechanisms that ensure distribution of data to users. In this paper, we focus on the latter. We study a delivery mechanism relying on mobile smart-tags to distribute data to fixed user nodes spread across a wide area. We assume a trivial user request matching mechanism whose goal is that data should be distributed to all user nodes. In other words, sensor data should be replicated to all users. Incorporating complex user requests with the data delivery mechanism we describe in this paper is a topic for future research.

It is generally assumed that wireless sensor networks rely on dense deployments of sensor nodes. A dense deployment is the condition for the establishment of a multi-hop network, where data is routed through multiple nodes on the way from sources to sinks, thus using short-range radio to

save energy [21]. The main challenge with such wireless networks is to account for the transmission cost when performing various forms of in-network processing such as collaborative signal processing [27] or data aggregation [8]. In some scenarios, clusters of sensor nodes are connected via a backbone (say the Internet) [6].

In this paper, we study an alternative form of system where a few sensor nodes are scattered over a wide area¹. Let us consider for instance a national park where weather stations are located at camping places all over the park; electronic display boards are placed along the roads taken by the hikers. The park is big enough, so that it is not a viable solution to cover it completely with wireless Ethernet or to equip each weather station or each display board with long range radio. An interesting solution consists in (a) equipping sensor nodes and display units with short-range radios and (b) equipping hikers with radio enabled smart tags, so that they can disseminate data as they move around. As a hiker walks by a weather station, his smart tag collects the latest measurements; he later transmits them to the display boards along the path he is following. A second hiker crossing this path gets her smart-tag updated and further disseminates data along her path. Note that in this scenario, data dissemination is achieved via a sequence of point-to-point updates between fixed and mobile devices (as opposed to multihop communication in a dense sensor network).

We consider systems where a large number of mobile smart-tags disseminate data from fixed sensor nodes to fixed display units as they move around. We believe that this form of data dissemination is relevant when monitoring large, hostile areas populated by individuals (humans, animals or robots) that can be equipped with smart-tags. For instance, environmental research is a natural target, with sensor nodes scattered over hundreds of square miles, and animals equipped with smart-tags in order to disseminate information across the area. Military applications are another candidate with possibly teleguided flying devices equipped with smart-tags connecting sensor nodes scattered over a large area [3].

In our example, hikers spread data across a national park as they move around. This example raises the following questions:

- *Coverage*: is data disseminated from all sensor nodes to all display units?
- *Delay*: how fast is data transmitted from a sensor node

¹A source is a sensor node that generates measurements or detections, a sink is another sensor node or possibly a gateway to a wide-area network infrastructure.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WSNA '02, September 28, 2002, Atlanta, Georgia, USA.
Copyright 2002 ACM 1-58113-589-0/02/0009 ...\$5.00.

to a display unit?

In order to answer these questions, an approach is to consider smart-tag based data dissemination as a form of epidemics. Data sources are similar to *infectious* individuals and display units are similar to *susceptible* individuals. The contacts between individuals occur indirectly through the movement of smart-tag between static nodes.

The form of data dissemination we consider is however not modeled by simple models such as SIR (Susceptible, Infectious, Removed [2]). In a SIR model, a single disease is propagated in a population where individuals are at first susceptible to the disease, then possibly infected, i.e. infectious for some time (they might infect the individuals they make contact with) and later on immune. First, such SIR models do not allow to model multiple diseases (data sources); second, in our model nodes remain infectious indefinitely; third, SIR models essentially focus on the spread of the epidemic while we are also interested in the time it takes for a node to be updated.

In this paper, we have chosen not to develop a complete stochastic model for smart-tag based data dissemination. This is a topic for future research. Instead we use simulation to study coverage and delay.

A large amount of work has been done in the database community on the topic of epidemic replication protocols [5, 22, 1]. These protocols assume a fully connected network; they are thus not suitable as such in the context we consider. However, the update propagation algorithm we propose for the point-to-point updates between a static node and a smart-tag is similar to the update propagation algorithms from the epidemic replication literature.

Active smart-tags are common place in our daily lives. They are generally used for security (IDs for building entrance) or for tracking (shoe-tags used in running competitions). These smart-tags contain a limited amount of memory and are capable of exchanging data via radio communication. Our implementation uses BlueTag development kits, i.e. alpha versions of Bluetooth based active smart-tags produced by a Danish start-up [15]. Bluetooth is an interesting networking technology because of its support for device discovery. Even if we do not focus on energy consumption, we consider that smart-tags should not broadcast data continuously; they should rather transmit data efficiently as soon as they are in the vicinity of a static node. The question is how efficient are the Bluetooth device discovery and connection establishment procedures: how long should a BlueTag remain in the vicinity of a static node for updates to be propagated?

This paper makes the following contributions:

1. We describe an algorithm for propagating updates between a static node and a smart-tag. This algorithm compares version information (i.e., time stamps) and propagates updates to out-of-date versions. This algorithm also accounts for memory limitation on smart-tags.
2. We propose a model for smart-tag based data dissemination. We simulate this model in order to study delay and coverage for the update propagation algorithm we propose.
3. We present performance results using our implementation with an alpha-version of Bluetooth-based smart-

tags. In particular we study the performance of the Bluetooth device discovery mechanism and the performance of our update propagation algorithm.

This work is part of the Manatee project at University of Copenhagen where we study Bluetooth-based monitoring applications [19].

2. UPDATE PROPAGATION

We consider a system composed of static nodes and mobile smart-tags that can exchange data when they are in the vicinity of each other (there is no connection among static nodes or among smart-tags²). A subset of the static nodes are *data sources*, i.e. they generate 3-tuples (*SourceId*, *Source Value*, *SourceTimeStamp*) where *SourceId* is an identifier of the data source, *SourceValue* is the data produced by the source (typically a boolean, a float or an integer) and *SourceTimeStamp* is the point in time at which the data value was generated. Other static nodes are *access points*, whose role is to exchange data with the smart-tags that pass by, store up-to-date data, and possibly display it to users.

Both static nodes and smart-tags maintain a state composed of the data obtained from data sources. This state is basically an array of 2-tuples (*SourceValue*, *SourceTimeStamp*) indexed by the *sourceId*. Note that for a static node which acts as data source *id*, the smart-tag state is the only data item stored locally, it is regularly updated as measurements are generated³.

Because connection can potentially be disrupted at any point in time, it is a good policy to exchange the most important data first. An access point updates its state based on the data carried by smart-tags in its vicinity and also updates the smart-tag's state to ensure further data dissemination. By contrast, a data source only updates the smart-tag's state. We present both algorithms in the rest of the section.

2.1 Access Points Updates

The algorithm run by access points to handle update propagation is the following: a connection is established with a smart-tag, the static node first updates its state and then updates the smart-tag state. Here is a pseudo-code version of the update propagation algorithm, implemented on access points.

```
loop
{
  begin connection with a smart-tag
  get data from the smart-tag
  update local state
  update smart-tag state
  end connection with the smart-tag
}
```

The static node is responsible for establishing connections with smart-tags. An alternative solution would be to have

²In order to minimize the requirements on smart-tags, we assume that smart-tags do not implement update propagation. As a consequence there are no direct communications between smart-tags. Studying an infrastructure where smart-tags can exchange data with each other is a topic for future research.

³Note also that we focus on the dissemination of single data items; dealing with time series is a topic for future work.

smart-tag broadcast their state – which does not seem reasonable from the point of view of energy consumption.

The update of the static node is straightforward: it is updated with more up-to-date data obtained from the smart-tag. The *SourceTimeStamp* is used to compare the version of the data on the smart-tag and on the static node. If the smart-tag version is more recent than the local version, the local version is updated.

The update of the smart-tag is a bit more subtle. Memory limitation might constrain the size of the state on the smart-tags⁴. In most cases, the smart-tag will not be able to store data from all data sources. Static nodes must take this limitation into account when they update the smart-tag state. We distinguish three policies for the access points:

- *STICKY*: The smart-tag keeps the data it carries when it is more recent than the data on the static node. The static node thus updates only the part of the smart-tag state that is outdated; it picks data from data sources in its state following a round-robin policy.
- *ROUND ROBIN*: The static node updates the whole smart-tag state regardless of whether it is outdated or not; it picks data from data sources in its state following a round-robin policy. Note that the static node state is updated before the smart-tag state and as a result, the smart-tag will always carry the latest version of the data.
- *RANDOM*: The static node updates the whole smart-tag state regardless of whether it is outdated or not; it picks data from data sources in its state randomly (avoiding duplicate data sources).

The *STICKY* policy favors a depth-first dissemination of data (a smart-tag disseminates a data item as far as possible), while *ROUND ROBIN* and *RANDOM* favor a breadth-first approach (smart-tags passing by an access point disseminate data from different sources).

In the next section, we analyze how these policies impact coverage and delay.

2.2 Data Sources Updates

In the case of data sources, the latest item generated should be pushed to the smart-tags that pass by. There is no problem if there is enough memory on the smart-tag. In case of memory limitation on the smart-tag, a replacement algorithm needs to be implemented. If the memory of the smart-tag is full, and if the smart-tag does not already store data from the data source it is connected with, then one data item has to be removed from the smart-tag memory and replaced by the data item from the connected data source. This is a classical replacement algorithm. We chose a form of LRU policy, where the data item with the oldest time stamp is replaced.

Note that this buffer replacement policy can be implemented on the smart-tag; in this case the exchange between the

⁴The smart-tags we have used for our implementation have a memory limited to 2Kb, i.e., a state of approximately 100 3-tuples. We further assume that static nodes are able to store data obtained from all data sources. Note that another important form of limitation concerns the amount of data to be exchanged between a static node and a smart-tag. Such a limitation is dictated by energy constraints on both smart-tags and static nodes. Taking the energy cost into account is a topic for future work

data source and the smart-tag is limited to the data source sending its latest data item. If buffer replacement is implemented on the data source then the data source first needs to obtain the smart-tag state in order to apply the buffer replacement algorithm and then send the resulting state back to the smart-tag.

3. A MODEL FOR SMART-TAG BASED DATA DISSEMINATION

In this section, we present a model of smart-tag based data dissemination that we simulate in order to illustrate its main characteristics. This simulation is not an in-depth analysis of our data dissemination model: first the experiments we present concern a limited number of parameters (in particular we do not study scalability in terms of data sources), second we make simplifying assumptions concerning the movement of smart-tags and the interaction between smart-tags and static nodes, finally we do not take energy consumption into accounts (energy is an important metrics in the context of sensor networks). A complete study of our data dissemination model is a topic for future work.

3.1 Model Characteristics

Our model of smart-tag based data dissemination can be expressed as follows. The system is composed of a fixed number N of static nodes and a fixed number ST of smart-tags. N_{ds} of the nodes are data sources. Smart-tags are mobile and follow fixed paths between static nodes, i.e., the static nodes are vertices and the paths edges in a connected graph. Each smart-tag moves for a given number of *hops*, where each hop corresponds to a move from one vertex to another following one edge.

We simulated this model using a discrete-event simulator in order to study coverage and delay. In our simulation, the static nodes are arranged following a planar mesh of size *size*, i.e., there are $N = size^2$ nodes; the vertex cardinality is two at the corners, three on the edges and four inside the mesh. This topology is neutral and allows us to focus on other parameters of the model. Data sources are uniformly distributed throughout the mesh. All smart-tags move for a constant number of *hops*. Smart-tags move randomly as follows: a smart-tag follows the edge it just took with a probability of 0.1 and follows all other possible edges with an equal probability. Note also that our simulator assumes that communications between a smart-tag and static nodes always succeed. This is an optimistic assumption because, in practice, a smart-tag might not stay long enough in the vicinity of a static node for the update propagation to complete successfully, or because energy cost considerations force to minimize the transfers between static nodes and smart-tags.

In our experiments, *size* is 10 and there are 5 data sources. The parameters are thus the number of smart-tags (ST) and the number of hops each smart-tag travels (*hop*). The idea is to model how data dissemination evolves during a fixed time period, say a day. Each smart-tag moves for a limited number of hops during that period. We choose, for our experiments, a small numbers of hops (20 to 50) and we study how the system scales with the number of smart-tags (20 to 500). Each hop takes one simulation tic. Initially, smart-tags are located at random nodes in the system. The simulation stops when each smart-tag has moved for the

specified number of hops.

We ran a few experiments with only one data source, and with enough memory on the smart-tags to carry data from this data source. Our goal was to study the impact of smart-tag movement on coverage.

In a first experiment, we considered that only one data item was generated at the data source. We measured the coverage, defined as the ratio between the number of nodes that have received data from the source and the total number of nodes in the system. We do not show this graph because of space limitation. We observed that with 20 hops, coverage increases with the number of smart-tags but it never attains 100%. With 50 hops however, coverage reaches 100% when the number of smart-tags is greater than 100. This is a familiar phenomenon with epidemic models where the epidemics spreads or dies depending on whether the *reproduction number*, a characteristic derived from the parameters of the system, crosses a certain threshold (in simple models the reproduction number is a combination of the time individuals remain infectious combined with the rate at which susceptible individuals make contact with infectious individuals).

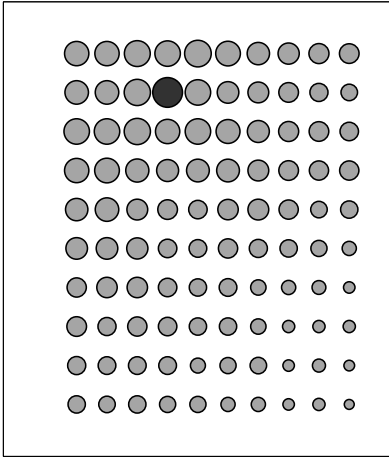


Figure 1: Discs located at the node position in the mesh represent the number of values collected by each node. The data source is located at node (2, 4).

In a second experiment, the data source generated 50 different data items. There are 300 smart-tags in the system; each smart-tag performs 50 hops (a different data item is generated for each simulation tic). Our first experiment tells us that coverage should be 100%. We measured how many of these data items were collected at each node. Figure 1 illustrate a run of this experiment. The figure is composed of a disc for each static node; this disc, located at the node position in the mesh, represents the number of values collected by each node. Here, the data source is located at the node of coordinate (2, 4) (we note (row, column) with (1, 1) the upper left corner on this figure) and it collects the 50 generated values. As expected, nodes placed close to the data source collect more values than those placed far away. For instance, node (2, 5), a neighbor of the data source, gets 37 of the 50 data items generated; all of them in one hop. As a comparison, node (8, 8) collects 9 data items with a delay varying between 21 and 44 hops. The minimum delay is the Manhattan distance between a node and the data source (1

for a neighbor, 10 for node (8, 8)). The maximum delay is 50 hops. A lesson from this experiment is that this model of data dissemination is not adapted for the monitoring of fast changing phenomena, where the data items must be delivered quickly to all nodes; it is however well suited for data items that are valid for a significant period of time (i.e., time enough to propagate to all the nodes).

3.2 Impact of Update Propagation Policies on Coverage and Delay

We have run a second set of experiments to study the coverage and the average propagation delay for the update policies introduced in the previous Section. Figure 2 and Figure 3 present the results.

For these graphs, there are 5 data sources that each produces 1 value. Smart-tags can store data from 3 data sources only. Each smart-tag performs 50 hops. The number of smart-tag in the system is a parameter in the experiments.

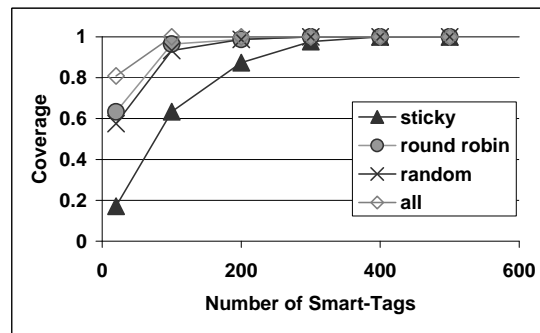


Figure 2: Coverage. There are 5 data sources and smart-tags can store data from 3 data sources only. The STICKY policy does not provide a good coverage compared to the other policies.

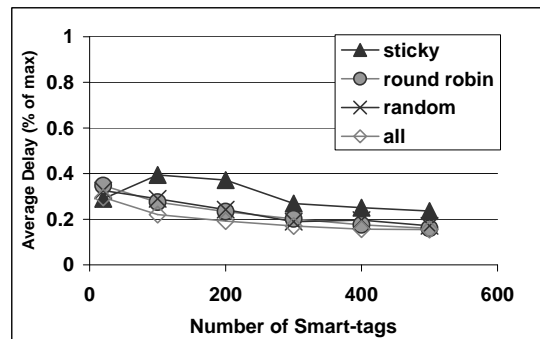


Figure 3: Delay. There are 5 data sources and smart-tags can store data from 3 data sources only. The average delay decreases as the number of smart-tags increases. The ROUND ROBIN and RANDOM policy perform better than the STICKY policy.

We measured the coverage for all data sources as the ratio (sum of nodes state data items that have been updated) / (total number of node's state elements, i.e., $N * N_{ds}$). We consider that a nodes state element had been updated if it contained at least one data item generated at the associated

data source. On Figure 2, the x-axis corresponds to the number of smart-tags in the system (ST) and the y-axis corresponds to the coverage. The graph traces the coverage for the three update policies we have identified to take into account memory constraints on smart-tags; it also traces the coverage without memory limitation as a baseline.

On Figure 3, the x-axis corresponds to the number of smart-tags in the system (ST) and the y-axis corresponds to the propagation delay (on each node, we measured, for each data source the difference between the time the last updated data item had been generated at the source and the time that data item was updated at the node); the y-axis is actually the ratio between the measured number of hops and the maximum number of hops (50). The graph traces the delay for the three update policies, as well as the delay without memory limitation as a baseline. As a general trend, the average delay decreases as the number of smart-tags increases. This is an intuitive result. On the one hand this means that the system only works if there are enough smart-tags in the system. On the other hand, the more smart-tags in the system, the more efficient the system is.

As far as policies are concerned, the STICKY policy does not perform well; coverage is low (less than 10% for 20 smart-tags – such a low coverage is the reason for the low delay with 20 smart-tags) and delays are high. The reason is that the STICKY policy does not favor *cross fertilization*. With the STICKY policy, smart-tags in the same area tend to fill their state with similar data items generated by data sources in their area. Once their state is full, smart-tags tend to keep this state and as a result data dissemination does not benefit from the fact that the paths of smart-tags cross. Data is disseminated to remote node only because a smart-tag travels to the remote node, not because the paths of several smart-tags cross (as it is the case with the ROUND ROBIN and the RANDOM policies). The difference between ROUND ROBIN and RANDOM is minimal. The delay obtained with ROUND ROBIN and RANDOM is close to the baseline; and gets closer as the number of smart-tag increases.

4. BLUETOOTH-BASED IMPLEMENTATION

Bluetooth was first conceived by Ericsson in 1994 as a cable replacement technology. It has now evolved in an ambitious full-fledged standard [10]. There have been a lot of promises over the years. Today, Bluetooth radio chips have yet to fulfill the promises of efficient point-to-multipoint connections and scatternets. Bluetooth is however becoming commonplace for point-to-point connections (involving mobile phones or headsets in particular).

Despite these limitations, we have chosen to use Bluetooth as the underlying networking layer for the Manatee project. Indeed, it is not our purpose to develop a new networking layer. We have chosen Bluetooth because of the wide support it enjoys and because of the numerous challenges it raises: impact of the inquiry/connection establishment procedure in dynamic application environments, relevance of TDMA/frequency hopping as the MAC layer of a multihop network (when scatternets become available), or integration of the Bluetooth stack on low power devices.

In this section, we briefly describe the Bluetooth infrastructure we have used for our implementation and we present performance results for device discovery and update propagation.

4.1 Bluetooth Infrastructure

Let us first briefly review the essential features of Bluetooth. Bluetooth operates in the 2.4GHz royalty free ISM band. It uses a Frequency Hopping Spread Spectrum (FHSS) scheme to guarantee robust point-to-point connections. When two devices communicate, one is denoted as the master and the other as the slave. A slave is following the hopping sequence dictated by its master. There is no slave to slave communication; furthermore, a slave is only allowed to transmit to the master once the master has contacted it. Data is transmitted in slots of a fixed size of $625 \mu s$, i.e. the slots alternate between master send and slave send. There are at most 7 slaves connected to a same master. The Bluetooth protocol stack is layered (the lower layers are implemented in the Bluetooth chips). One of the top layers is called RF-COMM. It provides a serial port emulation over Bluetooth. We describe the device discovery part of the protocol in the next section. We refer the interested reader to [10] for a thorough description of Bluetooth.

Our implementation relies on BlueTags development kits [15], i.e., alpha-versions of Bluetooth-based smart-tags. Figure 4 shows the device we have used for our experiment. Each BlueTag is equipped with an internal antenna and a Bluecore chip from Cambridge Silicon Radio (CSR); the transmission power of the radio is fixed to 2.5mW (class 2 module) allowing a range of approximately 20 meters. Each device has 2 Kb EEPROM memory for data storage and 512 Kb flash memory for embedded applications (i.e., the Bluetooth stack with serial profile and the code for BlueTag specific protocol on the devices we used). In addition to the Bluecore chip, the CSR chip contains an embedded microcontroller (XAP from Cambridge Consultants [18]) that runs the embedded application.



Figure 4: Bluetag. Bluetooth based smart-tag.

In our experiments, the static node is a PC with a commercial PCMCIA Bluetooth card from Brainboxes [17]. Those cards are also equipped with a CSR Bluecore chip. They do not support point-to-multipoint connections. This is a serious limitation for our implementation as a static node cannot handle simultaneous connections with several BlueTags passing in its vicinity (these connections have to be handled serially). In the rest of the section, we consider point-to-point connections between a static node and a BlueTag; handling point-to-multipoint connections is a topic for future work.

4.2 Device Discovery

In the first step of the update propagation algorithm that we presented in Section 2, the static node obtains data from a smart-tag in its vicinity. A possibility would be that smart-tag broadcast data continuously. This seems to be a waste of resources. On the contrary, it seems judicious to rely on a device discovery protocol so that smart-tags only send data when they are located next to a static node. Such a mechanism is at the heart of Bluetooth. This mechanism makes it possible for two devices hopping on different frequencies to discover each other (inquiry) and to establish connection (paging) [14].

In order for two devices to discover each other, they must be in two complementary states at the same time: *Inquiry* and *inquiry scan*. The inquiring device continuously sends out “is anybody out there” messages hoping that these messages (know as ID packets) will collide with a device performing an inquiry scan. To conserve power a device wanting to be discovered usually enters inquiry scan periodically with period T_{inq_scan} and only for a short time known as the inquiry window $T_{w_inq_scan}$. During this period, the device listens for inquiry messages. Since the devices are hopping on different frequencies the inquiry procedure must maximize the chance of two devices “catching” each other. To do this the inquiring device follows a fast half-slot timing, sending messages on two frequencies in one slot and listening on those frequencies on the following two half-slot, while the device in inquiry scan changes its phase very slowly—once every 2048 slots (1.28 s). The half-slot hopping is possible since the ID packets are small enough to allow the channel synthesizer to change frequency twice within one slot.

The inquiring device sends out a short packet (ID) and the inquiry scanning device responds with a frequency hop synchronization (FHS) packet containing among other things information about the devices hopping sequence and the device clock timing. In order to minimize collisions from responding devices, a device receiving an ID packet in inquiry scan will return to its previous state for a random number of slots (RAND) between 0 and 1023 slots before reentering inquiry scan. Upon the next received ID packet it will reply with a FHS packet. After the FHS packet has been returned the inquiry scanning device will move its phase one forward reenter inquiry scan again, meaning that it is likely to hear the next ID packets from the inquiring device, and the procedure starts over with random back off.

During inquiry the two devices follow a dedicated 32 frequency inquiry hop sequence. The inquiring device splits the sequence in two 16 slot (10 ms) parts: The A train and the B train. A single train must be repeated at least $N_{inquiry} = 256$ times before a new train is tried, one try of a train is thus $T_{train} = 4096$ slots long (2.56 s). The specification recommends a total inquiry period of 10.24 s to collect all responses, but inquiry can be aborted prior to this if devices have been discovered. The inquiring devices uses its own clock to determine the phase in the sequence and is thus random compared to any other device.

The Bluetooth specification[14] does not present an analysis of the expected time of inquiry. Here we present an analysis, of this inquiry strategy, by presenting a set of cases each being a building block of the expected duration. This leads to a set of overlapping intervals that correspond to the different cases in which inquiring and scanning devices might meet (see table 1 - the justification for the numbers of slots

associated to the different inquiry operations can be found in [13]) .

- When the inquiring device starts sending ID packets on its A part of the 32 inquiry frequencies the remote device may be listening in either the A part or the B part: These are cases 1, 2 - Figure 5.
- The most probable error is that the reply FHS packet is lost. This is a much larger packet than the ID, so transferring an FHS packet is more error prone than transferring an ID package. Losing an FHS packet can happen both when the inquiry scanning device is found in the A train or the B train: This gives us case 3 and 4 - Figure 6.
- The ID packet may be lost. If the inquiry scanning device wakes up during the first half of the train (length 2048) with time to spare for the random backoff it will get a second chance after $T_{inq_scan} = 2048$. Furthermore if it is in the A train it will get more chances if 2 train switches are performed: Corresponding to cases 5 and 6 - Figure 6.
- There are up to two chances to discover a device during a single train since $T_{train} = 2 \cdot T_{inq_scan}$. If both fail, the next chance of discovering a device is during the next try of the same train. This might happen if the first attempt fails and the random backoff period of the second attempt overlaps with a train switch. If this happens for a device on the B train at least 2 repetitions of each train must be tried to give the missed device a second chance: This is cases 7 and 8 - Figure 7.

The attempt in case 7 and 8 is prone to the same errors as cases 1-6, this means that we expect the pattern shown by cases 1-6 to repeat at intervals of length T_{train} .

Case	Best case	Worst case	no. slots
1	0	$T_{inq_scan} + RAND$	0-3071
2	T_{train}	$T_{train} + T_{inq_scan} + RAND$	4096-7167
3	0	$T_{inq_scan} + 2 \cdot RAND$	0-4094
4	T_{train}	$T_{train} + T_{inq_scan} + 2 \cdot RAND$	4100-8190
5	T_{inq_scan}	T_{train}	2048-4096
6	$T_{train} + T_{inq_scan}$	$2 \cdot T_{train}$	6144-8192
7	$2 \cdot T_{train}$	$2 \cdot T_{train} + T_{inq_scan} + RAND$	8192-11263
8	$3 \cdot T_{train}$	$3 \cdot T_{train} + T_{inq_scan} + RAND$	12288-15359

Table 1: Analysis of the intervals in which the devices are expected to be discovered in each case - not counting the slots used by the package exchange.

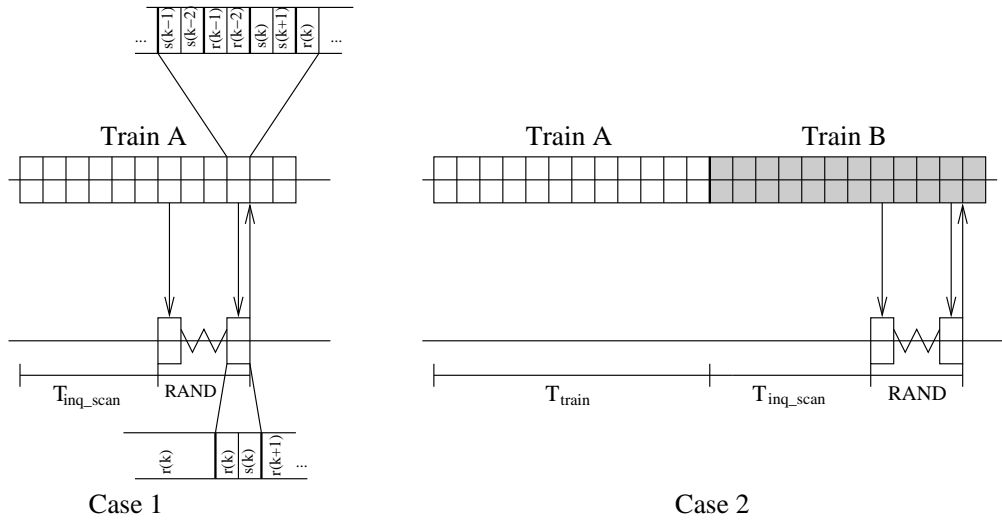


Figure 5: Inquiry with no errors. Since the inquirer splits the sequence in two, there are two possibilities of how long it is going to take to discover a device—even in an error free environment. In the blow-ups, r denotes receiving and s denotes sending.

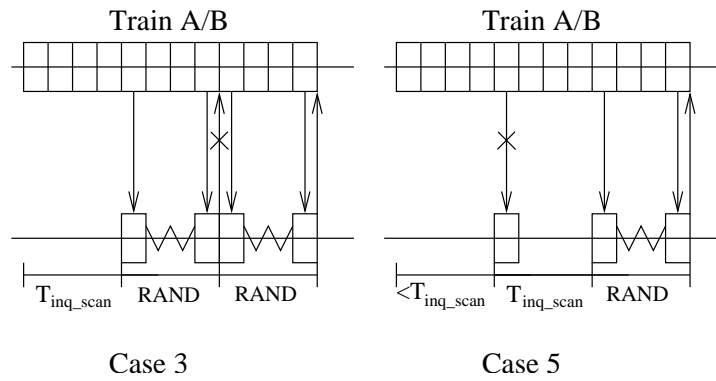


Figure 6: Case 3/4 - the reply FHS packet is lost requiring an additional random backoff. Case 5/6 - The inquiry ID packet is lost requiring $T_{\text{inq_scan}}$ more to discover the device.

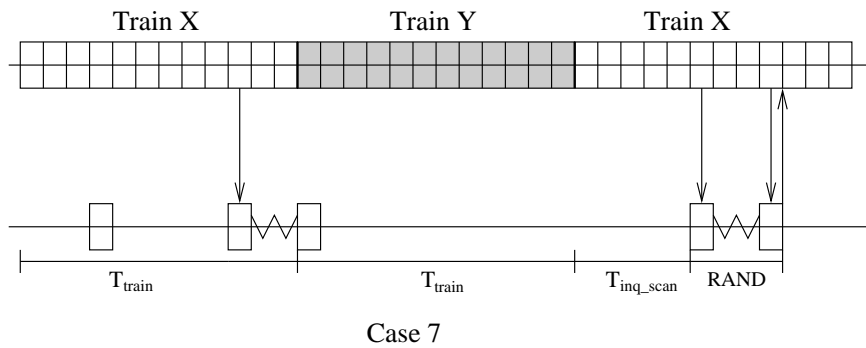


Figure 7: Case 7/8 - The first train is missed, the inquiry scanning devices have to wait until the next try of the same train to get a new chance. The figure shows the random backoff period overlapping with a train switch.

Depending on how we assign likelihood to each of these events we can predict how long the inquiry is going to take. Figure 8 illustrates a possible distribution of the number of device discoveries as a function of time. It is our guess that the vast majority of device discoveries will be found within case 1 and 2. Therefore we expect the average device discovery time to be slightly larger than $\frac{7167}{2} \cdot 625\mu s = 2.24s$ increased only due to packet losses (in cases 3-7).

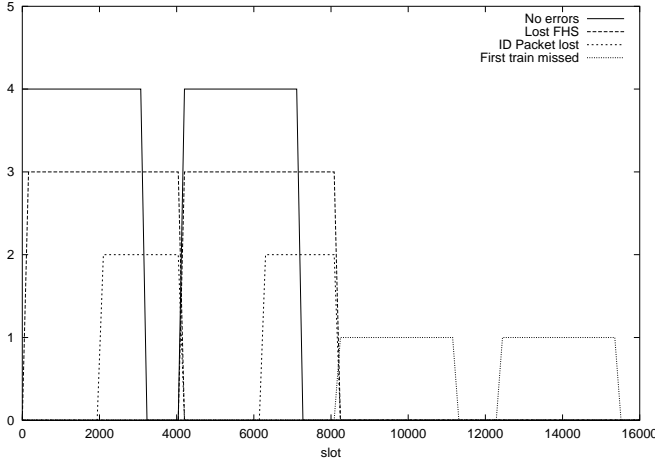


Figure 8: Illustration of the intervals in which device discovery is expected to happen. The figure illustrates the overlap of intervals and the repetitive pattern. The height is chosen arbitrarily, and the shape of the boxes are not representative of the shape of how the actual measurements will look.

Figure 9 shows actual measurements obtained with a BlueTag device and a static node. The experiment duplicates the conditions of the experiments created by Kasten and Langheinrich [11]: the two Bluetooth devices are separated by one meter; 1500 inquiries are performed (10 seconds max) and we measure the time it takes for the devices to discover each other. The graph traces the number of times a device was discovered as a function of time. The result we obtain with the BlueTag is similar to the ones we have obtained with various Bluetooth devices [13]. These results confirm our inquiry time model.

Additional experiments, documented in [13], have shown that inquiry time remains constant as the distance between two devices increase. As a result we can expect a static node to discover a BlueTag when it is 20 meters away (at least). Note that this inquiry procedure is the only mechanism required for tracking applications.

4.3 Update Propagation

In this section, we describe our implementation of the update propagation algorithm and we analyze its performance. We use the BlueZ Bluetooth stack on the static nodes. The BlueZ stack [16] is included in the Linux kernel 2.4 series. It contains an implementation of the upper layers of Bluetooth and provides a standard socket interface for the programmers. BlueZ is the most stable stack we have found [13]. It is however still a work in progress with occasional unstable behaviors (we suspect there are problems related to the PCMCIA card services).

The PCMCIA card we use for our experiments does not

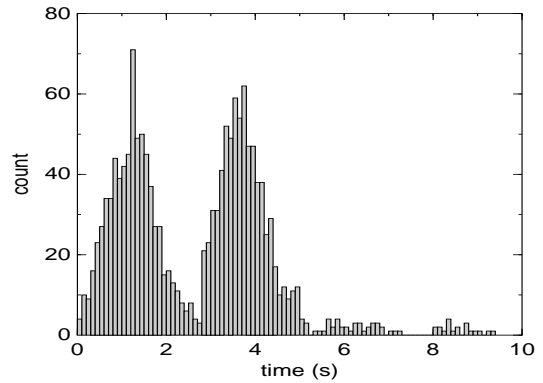


Figure 9: Inquiry Time Distribution - 1 BlueTag

allow point-to-multipoint connections. As a consequence, the static node handles one connection to a smart-tag after another and thus establishes a connection as soon as a smart-tag is discovered.

The connection is established via the RFCOMM layer. The static node is the initiator, and the BlueTag is the respondent. Once a connection is established, the static node uses standard input and standard output to exchange data using BlueTag specific commands such as *RetrieveData* and *StoreData* (each of these commands introduces a 5 byte overhead). In our experiment, we read and write a state of 14 bytes.

The time it takes to propagate the updates is thus the sum of the time it takes to discover a smart-tag, the time it takes to establish an RFCOMM connection and the time it takes to read and write data.

We detailed our results concerning device discovery in the previous section. In the rest of this section, we present the results of an experiment we have run to study connection time and read/write time. In this experiment, a static node and a BlueTag are placed one meter apart; we measure the time it takes to propagate updates (device discovery, connection and read/write).

Figure 10 traces the distribution of connection time once a BlueTag is discovered. We observe that connection time follows a long-tail distribution. Occasionally, it takes up to 20 seconds to establish a connection. In most cases however a connection is established in less than 2 seconds. This suggests an implementation where the static node stops pending connections if they take more than 2 seconds and tries again. We observe a uniform read/write time of approximately 6 seconds. This number is high. This is to be expected because we are using a development kit, for which there has been no focus on the performance of code execution.

Figure 11 shows the median time for update propagation and its decomposition into inquiry time, connection time and transfer time. We see that with the BlueTag development kit, propagation time is dominated by transfer time. This is encouraging as this number will be improved with the next generation of BlueTags. As it stands, our implementation allows update propagation between individuals equipped with BlueTags passing by static nodes with a speed of up-to 10 km/h (10 seconds propagation in an area of 40 meters), i.e., the speed of a person running.

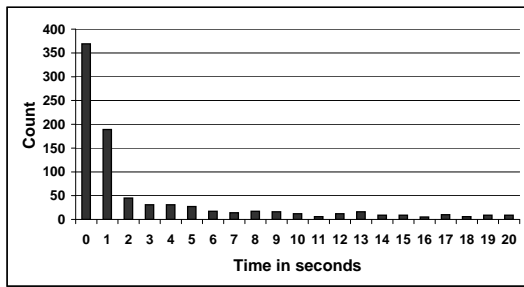


Figure 10: Distribution of connection time - 1 BlueTag

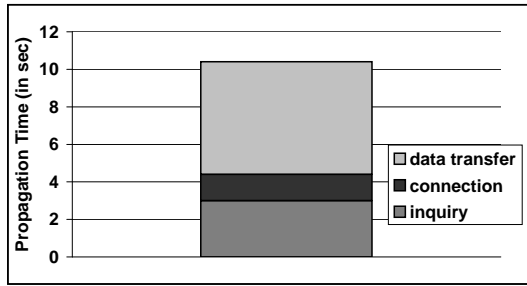


Figure 11: Decomposition of propagation time between a static node and a Blue-Tag into inquiry time, connection time and transfer time

5. RELATED WORK

The Pollen project at XRCE [7] and the Locust Swarm project at MIT Media Lab [24] first proposed to leverage the movement of individuals to transport messages between static devices. None of these projects focused on data dissemination. In [7], PDAs are used to carry message from one source node to a destination node, while in [24], active badges obtain location information from fixed access points. A simulator is used in [7] to obtain preliminary results concerning the delay with which messages are delivered. The implemented prototypes rely on iButtons [9] (that require physical contact) and infrared respectively. In [7], the authors suggest that Bluetooth is a promising technology that should be investigated for this kind of infrastructure.

Our data dissemination model has been inspired by epidemic protocols developed in the context of replication in distributed database systems [5, 22, 1], reliable multicast [25] or resource location [12, 26]. Our solution relies on the same principle of pair-wise communications between pairs of nodes chosen randomly. A characteristic of our approach is that nodes are disconnected and as a result the connection between nodes is dictated by the movement of smart-tags, not by a well-defined protocol.

Most articles about Bluetooth focus on protocol extensions, using simulation to analyze performance. To the best of our knowledge, the few papers studying the performance of Bluetooth using actual devices have been written by the distributed systems group at ETH Zurich. They are developing Bluetooth sensor devices in the context of the Smart-its project [20]; and they are using these devices to study the properties of Bluetooth in the context of a sensor network [11]. In [23], the authors study the performance of the

inquiry procedure as a function of the number of devices within communication range. Their results are complementary with the ones present in this paper regarding the performance of inquiry between two devices. We are planning to use their device as access points in a demonstrator we are currently implementing to disseminate data obtained from the coke machines in our department [19].

6. CONCLUSION

In this paper, we studied smart-tag based data dissemination, a form of data dissemination inspired by epidemic protocols where mobile individuals, equipped with smart-tags, disseminate data across disconnected static nodes spread across a wide area. We used a simulator and an implementation using Bluetooth-enabled smart-tags to illustrate the characteristics of our approach. Our results confirm the intuition that our approach works best for the dissemination of data under loose time constraints when there are lots of smart-tags in the system. Our results also show that a breadth-first approach to the dissemination of data (where smart-tags passing by an access point disseminate data from different sources) is far more efficient than a depth-first approach (where a smart-tag disseminates a data item as far as possible). We showed that Bluetooth-enabled smart-tags were promising candidates for the implementation of smart-tag based data dissemination system. In particular, we focused on the device discovery protocol and proposed a model for explaining the distribution of inquiry times. Finally, we identified numerous areas for future work both theoretical (e.g., a stochastic model of smart-tag based data dissemination) and practical (e.g., incorporating complex user requests, disseminating sequences, taking energy cost into account, handling point-to-multipoint connections in Bluetooth).

7. REFERENCES

- [1] Divyakant Agrawal, Amr El Abbadi, and R. Steinke. Epidemic algorithms in replicated databases (extended abstract). In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona*, pages 161–172. ACM Press, 1997.
- [2] Håkan Andersson and Tom Britton. *Stochastic Epidemic Models and their Statistical Analysis*. Springer Lecture Notes in Statistics, 2001.
- [3] UC Berkeley and MLB Co 29 Palms Fixed/Mobile Experiment. <http://tinyos.millennium.berkeley.edu/29palms.htm>.
- [4] Mitch Cherniack, Michael J. Franklin, and Stanley B. Zdonik. Data management for pervasive computing. In *Proceedings of the International Conference on Very Large Databases VLDB*, 2001.
- [5] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. *Operating Systems Review*, 22(1), 1988.
- [6] Deborah Estrin, Ramesh Govindan, and John S. Heidemann. Embedding the internet: Introduction. *Communications of the ACM*, 43(5), 2000.
- [7] Natalie S. Gance, Dave Snowdon, and Jean-Luc Meunier. Pollen: using people as a communication

- medium. *Computer Networks*, 35(4), 2001.
- [8] John S. Heidemann, Fabio Silva, Chalermek Intanagonwivat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *SOSP*, 2001.
 - [9] iButtons Home Page. <http://www.ibutton.com>.
 - [10] Charles F Struman Jennifer Bray. *Bluetooth Connect Without Cables*. Prentice Hall, 2001.
 - [11] Oliver Kasten and Marc Langheinrich. First experiences with bluetooth in the smart-its distributed sensor network. In *PACT*, 2001.
 - [12] David Kempe, Jon Kleinberg, and Alan Demers. Spatial gossip and resource location protocols. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 163–172. ACM Press, 2001.
 - [13] Martin Leopold. Evaluation of bluetooth communication: Simulation and experiments. Technical report, DIKU 02/03, 2002.
 - [14] Specification of the Bluetooth System Core version 1.1 2002. http://www.bluetooth.org/docs/bluetooth_v11_core_22feb01.pdf.
 - [15] BlueTags Home Page. <http://www.bluetags.com/>.
 - [16] Bluez Home Page. <http://bluez.sourceforge.net/>.
 - [17] Brainboxes Home Page. <http://www.brainboxes.com/>.
 - [18] Cambridge Consultants Home Page. http://www.cambridgeconsultants.com/pd_xap_reduced.shtml.
 - [19] Manatee Project Home Page. <http://www.distlab.dk/manatee/>.
 - [20] Smart-ITs Home Page. <http://www.smart-its.org/>.
 - [21] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.
 - [22] Michael Rabinovich, Narain H. Gehani, and Alex Kononov. Scalable update propagation in epidemic replicated databases. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings*, volume 1057 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 1996.
 - [23] Frank Siegemund and Michael Rohs. Rendezvous layer protocols for bluetooth-enabled smart devices. In *1st International Conference on the Architecture of Computer Systems ARCS - Trends in Network and Pervasive Computing*, 2002.
 - [24] T. Starner, D. Kirsch, , and S. Assefa. The locust swarm: An enviromentally-powered, networkless location and messaging system. In *1st International Symposium on Wearable Computers (ISWC '97)*, 1997.
 - [25] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Middleware'98*, 1998.
 - [26] Robbert van Renesse. Scalable and secure resource location. In *HICSS*, 2000.
 - [27] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, mar 2002.