RECOUP: Efficient Decentralised Configuration Update for Wireless Sensor Networks

S. Pennington, A. Waller, T. Baugé
Thales Research and Technology (UK) Ltd.
Worton Drive, Worton Grange, Reading, RG2 0SB
e-mail: {Sarah.Pennington, Adrian.Waller, Timothy.Bauge}@thalesgroup.com

Abstract – In this paper we present Reliable Configuration Update (RECOUP), an efficient protocol for updating the configuration of a Wireless Sensor Network (WSN). The protocol involves an initial flooding of the network followed by a ‘local repair’ mechanism to update nodes that miss the initial flood. We have simulated the protocol in TOSSIM, a TinyOS simulator and compared its performance to that of a simple flooding algorithm. We show that RECOUP is more efficient as it uses fewer messages to update the network and has a lower packet loss. We also demonstrate that RECOUP guarantees to deliver the configuration update to all nodes in the network.

I. INTRODUCTION

A WSN is a wireless ad-hoc network comprising many intelligent sensing devices, called sensor nodes or nodes, and one or more base stations. Sensor nodes contain a small processor, sensors, a means of communicating wirelessly and a power source, usually a battery. They are therefore resource-constrained with limited processing power, limited memory and restricted energy.

Each sensor node supports a multi-hop routing algorithm, which enables each node to forward data for other nodes that are out-of-range of the base station (see Fig. 1). The base station acts as a gateway between the sensor nodes and the end user and will usually possess more energy and communication resources and computational power than sensor nodes.

Although originally developed for military applications, WSNs are now used in many civilian applications such as environment and habitat monitoring, healthcare applications and traffic control.

Within a WSN, it may be necessary to employ an adaptive management framework, which allows for certain configuration parameters to be changed after the initial deployment of the network. For example, the level of security for sending and receiving messages in a WSN may need to be changed. The security level can be increased to provide higher protection against attacks from malicious sensor nodes and it can also be decreased again when the threat subsides to conserve power. In order to implement this framework, all sensor nodes in the network need to be instructed how to set their configuration. This needs to be done in a reliable, secure, synchronised, and efficient way.

RECOUP is a method for ensuring that all nodes in a network receive configuration management messages, which inform a node to update its configuration. It ensures that all nodes have a consistent configuration, and allows recovery of situations where some nodes are in an inconsistent state. It is particularly suited to applications that send small configuration payloads so that the configuration is contained within a single packet or where power is limited, particularly for communications such as in WSNs. In addition, the protocol is appropriate for applications where speed of update is of importance or where nodes need to recover from an inconsistent state, for example new nodes joining the network.

In this paper we evaluate the performance of the RECOUP protocol and compare it to a flooding protocol that broadcasts each configuration update multiple times. The rest of this paper is organised as follows. Section II provides an overview of related work. The RECOUP protocol is described in Section III. Section IV describes the methodology used in the simulations, while the simulation results and evaluation of the protocol are provided in Section V. The conclusions are described in Section VI.

II. RELATED WORK

Protocols to achieve reliable reconfiguration in wired networks exist, such as reliable IP multicast. However, these are not appropriate for WSNs due to their significant overheads in terms of bandwidth used and maintenance of state on network nodes. Moreover, these approaches do not inherently allow recovery from an inconsistent network state, such as may occur when a new node joins.

Reconfiguration, taking into account the significant constraints of WSNs is a relatively new area of research for which few solutions have been proposed. Of those that do exist, most, such as MOAP [1], Deluge [2] and TinyCubus [3] are designed for distributing code updates, which are assumed to be large. Their main aim is to save communications and memory costs where multiple packets of data need to be sent and speed of update is a minor issue.

Fig. 1. Typical multi-hop WSN architecture.
Many assumptions and optimisations used by these approaches, such as the use of negative acknowledgments from receivers to signal missed packets, are not valid for small, single packet, updates which need to be disseminated rapidly. In addition, use of a periodic broadcast by existing nodes of the current code version to allow new nodes or nodes that have been out of range to update themselves is relatively inefficient and will lead to significant delays in (re-) incorporating such nodes in the network.

Finally, synchronisation of the updates is rarely considered. It is generally assumed that the network will be unable to perform its usual operations (e.g. collecting sensor data) while the code update is being distributed. For code updates, this is not a problem as these will be relatively infrequent. However, for frequent security level changes for example, this would be a significant issue. In [4], small configuration updates are considered, but the use of TCP/IP is suggested for point-to-point updates, which is inefficient, and reliable broadcast updates are left as future work.

### III. RECOUP

RECOUP describes a procedure for the distribution of configuration management messages in a WSN, such as security management messages in an adaptive security framework. The protocol has two major features. Firstly, new configuration updates will be flooded throughout the sensor network using a smart flood mechanism that is included in the protocol. Secondly, if nodes have missed updates, e.g. due to being out of range when the last update was sent or because they are new nodes being added to the network, then the protocol provides a ‘local repair’ (update) mechanism, which is triggered on the next transmission from the out-of-date sensor. This provides a rapid repair mechanism should some nodes have an inconsistent configuration and ensures that the network has a very high probability of being in a consistent state throughout.

The need to update the configuration will be triggered by a management application. For the following description, we assume, without loss of generality, a gateway sensor node that is connected to a PC running the management application and a network of sensor nodes as shown in Fig. 1. Note that the protocol makes no assumption as to the communication pattern of the sensor network (peer to peer, tree structure, etc), or the network’s topology (single or multi-hop). The protocol’s operation is described in detail below. Note that this protocol is patent pending.

When the configuration of the sensors needs to be changed, the management application on the PC will send a request to the gateway node to change the configuration of the network. The gateway will then send out an update to all the sensor nodes in its immediate neighbourhood using a link-level broadcast of a ‘configuration management message’ that contains this new configuration.

To enable nodes to tell which configurations are more recent, either version numbers or time stamps can be used within this message. For the rest of the description below, version numbers are used to simplify the description without loss of generality.

When a sensor node first joins the network, it has an invalid version number to indicate that the node has not yet received a valid configuration update. It will update to the first valid configuration it receives. A flag is used in the configuration management message to show whether it contains a valid version number. When the new node receives a sensor data message from another node, it will broadcast its invalid configuration with its flag set to false. This alerts other nodes in the network that it is out-of-date and nodes with a valid configuration will broadcast their configuration with the flag set to true. On receiving a valid configuration, the new node will immediately update to this configuration.

When a sensor node that has a valid configuration receives a configuration management message, it compares the version number of the message with its stored version number, as shown in Fig. 2. If the received message is more recent than...
the previous message the node received, it will immediately change its configuration to that indicated by the configuration management message. At the same time, the node will also change the version number of its configuration to match that of the received message. It then broadcasts this new configuration in a configuration management message to its neighbours. In this way, the update will be flooded throughout the network. If the received version is less recent than that stored, it will broadcast its own, more recent, configuration to its neighbours. However, if the versions are the same, it will simply ignore the message, preventing the message from being flooded indefinitely.

A local repair is described in Fig. 3 and is achieved as follows. When a sensor node receives any message that is not a configuration management message, it first checks that the configuration of the node that sent the message is the same as its own configuration (note that the protocol requires that the configuration of the sending node can be unambiguously determined from the received message). If the two configurations match, then the message is processed normally. If they do not match, then the node that received the message may either have a more recent or older configuration. To determine which is the case, it sends a configuration management message to its neighbours with its own settings and version number. As described above and in Fig. 2, on receipt of this message either the other nodes will update their configuration (if they have older versions) or will respond with the newer configuration (if they have newer versions).

In the case that on receiving a message a node detects a mismatch in configurations, the received message may either be processed normally or dropped, depending on the requirements of the application. For example, when sending security management messages using this protocol, if the current security policy requires packets to be sent with authentication but the received message is unauthenticated, then the data should be regarded as potentially compromised and the packet should be dropped by the node. However, if the received message were authenticated, then regardless of the current security policy any compromise of the data would be detectable. In this case, the packet could be processed normally. In this way, the amount of application data lost due to lack of synchronisation of configurations is kept to a minimum. However, it should be noted that application data packets might still be dropped, even if link layer reliability protocols are present. Critical data should only be sent if protected by a higher layer reliability protocol.

IV. EVALUATION METHODOLOGY

The full RECOUP protocol and a flooding algorithm were implemented in the nesC programming language on the TinyOS v1.1.15 platform [5]. To evaluate the protocol, we used TOSSIM, a bit-level simulator designed for the TinyOS platform [6], to simulate 25 sensor nodes laid out in a 5x5 grid with 20ft spacing, and with the base station positioned in the top left corner of the grid.

The metrics we used to evaluate the protocol were driven by the claims of the RECOUP protocol. We claim that RECOUP is significantly more energy-efficient than previous approaches, especially for small configuration updates, because it sends fewer messages than existing technologies for the same level of reliability. We have assumed that the radio on the sensor node uses the most power and that the packet length used by different protocols will be roughly the same. Consequently the number of packets required to update the configuration of the network gives the best indication of the efficiency of the protocol. We observed the number of configuration management messages required to update the network.

Furthermore, we claim that RECOUP achieves a faster update of configurations throughout the network than previous approaches and mismatches in configuration will lead to at most only a small loss of application data. We therefore observed the packet loss and the time taken to update the configuration of the network.

A. TOSSIM

TOSSIM, a sensor network simulator based on the TinyOS operating system, is a discrete event simulator, which compiles directly from TinyOS code. It simulates the network at the bit level and models each individual sensor capture and every interrupt, allowing users to debug, test and analyse algorithms in a controlled and repeatable environment [6].

TOSSIM provides two built-in radio models: simple and lossy. The simple radio model places all the nodes in a single radio cell with error-free transmission, while the lossy model
places nodes in a directed graph. Each edge \((u, v)\) in the graph represents the bit error rate when mote \(u\) sends to mote \(v\). [6]

For the simulations, we used the lossy radio model and used TOSSIM’s LossyBuilder Java tool to generate the bit error rates for each mote pair. LossyBuilder assumes the sensor nodes have a radio range of 50 feet and assigns each mote pair a two-way loss rate [6], which is calculated using empirical data observed in [7]. LossyBuilder generates packet loss rates for each mote pair by sampling Gaussian packet loss probability distributions for each distance, fit to match the empirical data [6]. It then translates packet error rates into independent bit error rates.

**B. RECOUP Implementation**

The implementation comprises two main functions: data collection and the RECOUP protocol. The sensor data collection application is based on Surge, a sample application that comes packaged with TinyOS [5][7]. It collects light data every 10 seconds and uses the MintRoute [5][7] multi-hop routing protocol, also provided with the TinyOS distribution, to forward data to the base station.

MintRoute is based on a dynamic spanning tree rooted at the base station (Mote 0 by default). It selects a parent by eavesdropping on received messages and attempts to use the parent with the lowest depth in the tree and with the highest estimated link quality. A new parent is selected when the estimated link quality drops below a certain threshold [7].

The full RECOUP protocol as described in Section III was also incorporated into the implementation.

**C. Flood implementation**

The implementation of the flood algorithm also uses the Surge sensor data collecting application with the MintRoute routing protocol that are packaged with TinyOS [5][7].

As there are few flooding protocols available for delivering small updates, we have based the flood on a smart broadcast provided with TinyOS in order to compare RECOUP to an example of the current state-of-the-art. It operates as follows. The gateway broadcasts each configuration management message eight times, back-to-back. The more times a message is broadcast, the more nodes it reaches but the more congested the network becomes. If the update were broadcast fewer times, the network would be less congested as there would be fewer configuration management messages sent. However, the update would be less likely to reach all the nodes in the network. We chose to broadcast the update eight times as this would deliver the update to the majority of nodes in the network without congesting the network excessively.

Each broadcast configuration management message contains a version number, which is incremented each time the configuration of the network is updated. On receiving an update, a node will check the version number of the configuration update and will only forward those that contain a more recent version than its stored version. When a node forwards a configuration management message, it is also broadcast eight times.

**D. Framework**

The RECOUP protocol and the flood were evaluated within the context of an adaptive security framework. We made several assumptions about the framework. We assumed the framework used only two security levels: high and low. However, to simplify the implementation, we did not employ any security mechanisms but simply indicated the current security level of a node in each sensor data message sent. A sensor data message received in the wrong mode was dropped and version numbers were used in the configuration update messages rather than timestamps to compare configurations. Each broadcast was performed only once.

**V. RESULTS**

A. Packet Loss

We measured the percentage of routed sensor data packets lost using both RECOUP and the flood to update the configuration of the network. Fig. 4 shows the percentage of original sensor data packets sent over the radio that are not received at the base station against the frequency of changing the configuration of the network. For each frequency of changing the network configuration, we calculated the average packet loss. We then calculated the 5-point moving average of these average packet losses. Fig. 5 shows the 5-point moving average of the average packet loss for each frequency of sending configuration updates. Both the performance of RECOUP and the flood are shown.

Fig. 4 demonstrates that the packet loss increases as the frequency of configuration changes increases. Furthermore, it also indicates that there is more variability in the packet loss for the flooding protocol. Fig. 5 suggests that the packet loss varies linearly with the frequency of changing configuration for the RECOUP protocol. This is because each time the configuration is changed, there is a small period of time when some nodes are in the wrong configuration and so any sensor data these out-of-date nodes send is dropped. However, this is not the case for the flood. Furthermore, Fig. 5 shows that the average packet loss for the flood protocol is higher than that for RECOUP.

Further analysis of the simulation output suggests that this
difference in the packet loss is due to the higher overheads of the flood protocol. The higher number of configuration messages sent using this protocol causes the network to become congested and send queues to overflow, meaning that messages are dropped. This also affects the routing protocol, making it more difficult to establish a route to the base station.

B. Configuration Messages

To indicate the overhead of both RECOUP and the flood, we measured the number of configuration management messages required to update a network of 25 nodes. Fig. 6 shows the number of configuration management messages sent during a 30-minute simulation against the number of times the network configuration was changed during the simulation for both the RECOUP protocol and the flooding protocol. A linear regression line has been added to the chart.

From Fig. 6 it can be seen that the flooding protocol sends approximately eight times the number of configuration messages as the RECOUP protocol. This is as expected, since the flood broadcasts each configuration management update eight times whereas RECOUP only broadcasts each update once. Therefore the RECOUP protocol is more efficient in terms of the overhead of the protocol. It also indicates that the additional overhead caused by RECOUP’s local update mechanism is small compared to the initial flood.

C. Time taken to update the network configuration

We compared the time taken to update a network of 25 nodes using RECOUP to that using the flood. The configuration of the network was changed every 50 seconds for five virtual minutes and the time each mote updated its configuration was recorded. Fig. 7 shows the number of nodes that are in the wrong configuration with time.

From Fig. 7 it can be seen that both the flooding and the RECOUP protocol deliver the configuration update to all nodes in the network. It demonstrates that the flooding protocol delivers the update to the nodes faster than the RECOUP protocol, which suggests that the increased packet loss of the flooding protocol is not due to nodes being in the wrong state but congestion in the network. Although the results show that all nodes received the configuration update with the flooding protocol, it does not guarantee delivery to all nodes. While unlikely, it is possible that a mote could miss all eight broadcast messages and will be out of sync with the network until the next configuration update.

For the RECOUP protocol, the time taken for all 25 nodes to update depends on the number of nodes that miss the initial flood. Fig. 7 indicates that although some nodes miss the initial flood, the local update mechanism enables them to receive the update the next time they send a sensor data message. Fig. 7 shows that the longest time taken to update all the nodes in the network was 9.3 seconds, which corresponds to the length of time before the last out-of-date mote sent its next sensor data message after the update.

To show that the RECOUP protocol guarantees to update nodes that move into range of the network, we simulated a 5x5 grid of nodes with two additional nodes out of range of the network. At zero seconds, the configuration of the network

Fig. 5. 5-point moving average of the average packet loss with frequency of configuration changes.

Fig. 6. Number of configuration messages with frequency of configuration changes.

Fig. 7. Speed of updating the network.
was updated and the time at which the nodes updated was recorded. After 50 seconds had elapsed, these two nodes were then moved so that they were in range of the network. Fig. 8 shows the time at which the nodes updated with both the RECOUP protocol and the flood.

To determine the effect of new nodes joining the network, we simulated a 5x5 grid of nodes with two additional nodes placed within range of the network but initially switched off. At zero seconds the configuration of the network was updated and after 20 seconds the two new nodes were switched on. Fig. 9 shows the time at which each node updated to the new configuration of the network using both RECOUP and the flood.

Fig. 8 indicates that RECOUP delivered the update to the nodes 110 seconds after they moved within range of the network. From Fig. 9 it can be seen that the RECOUP protocol delivered the update to the two new nodes after 115 seconds. The delay relates to the length of time required for the routing protocol to find a path from each of the two new nodes to the base station. The out-of-date nodes receive the new configuration update when they route a sensor data message to another up to date node. From both Fig. 8 and Fig. 9 it can also be seen that the flood does not deliver the update to either nodes that join the network or to nodes that move within range of the network after the initial flood. These experiments illustrate that, due to its data plane driven local repair mechanism, the RECOUP protocol guarantees to deliver configuration management messages to all sensor nodes in the network, whereas the flood does not.

VI. CONCLUSIONS

In this paper we introduced the RECOUP protocol. We compared its performance to a flooding protocol that broadcasts an update eight times.

We found that the flooding protocol results in eight times the number of configuration update messages being sent compared to the RECOUP protocol. Furthermore, the average packet loss for the RECOUP protocol was lower than that for the flooding protocol. The high number of packets sent using the flooding protocol caused congestion in the network which led to queue overflows and packets being dropped.

Both protocols delivered each update to all nodes in the network, although the flooding protocol resulted in a faster update of the network than the RECOUP protocol. However, we demonstrated that the flooding protocol does not guarantee to deliver the update to nodes that join the network after the initial flood. Conversely, the ‘local repair’ feature of the RECOUP protocol means that it is able to deliver the update to all nodes that miss the initial flood.

ACKNOWLEDGMENT

The work presented in this paper was partially funded by the FP6 EU project “e-SENSE, Capturing Ambient Intelligence for Mobile Communications through Wireless Sensor Networks”, contract number: IST-4-027227-IP, www.ist-e-sense.org and the FP7 EU project “SENSEI, Integrating the Physical with the Digital World of the Network of the Future”, contract number: 215923, www.sensei-project.eu.

REFERENCES