Multi-protocol Scheduling for Service Provision in WSN

Michael Breza, Shusen Yang and Julie McCann
Department of Computing, Imperial College London, London, U.K.

Keywords: Wireless Sensor Networks.

Abstract: Currently, Wireless Sensor Network (WSN) systems are made of aggregates of different, non-related protocols which often fail to function simultaneously. We present a self-organising solution that focuses on queue length scheduling. To start, we define a network model and use it to prove that our solution is throughput optimal. Then we evaluate it on two different WSN test-beds. Our results show that within the theoretical communication capacity region of our WSN we outperform the current solutions by as much as 35%.

1 INTRODUCTION

Wireless sensor networks (WSNs) are networks of small micro-controllers with sensors used to measure phenomenon in the environment and then communicate it via low-powered radio. They are designed to be diminutive in both physical size and acquisition price. Small size enables their deployment in large numbers and in a broad spectrum of inaccessible or dangerous environments.

The capabilities of a WSN node vary depending on the class of node; a shoe box size node is basically a laptop, a matchbox size node has 16 MIPS micro-controller with only hundreds of kilo-bytes of memory. It is the latter form of sensor network which is the focus of this paper. Consequently the largest challenges involved in the development of WSN systems are their limited resources. Typically they are battery powered, so energy is a key constraint and the greatest consumer of energy on a WSN node is the radio transceiver. Radio communication is vital to transform a group of sensor nodes into a coherent, usable distributed sensing and computing system - where the real power of WSNs lie.

In this work we focus on environmental monitoring, a major application area for WSNs (Martinez et al., 2006; Werner-Allen et al., ; Cardell-Oliver et al., 2005). Environmental applications typically have large numbers of sensors in different locations. All the nodes sample environmental data at the same time, and then send that data to a data collection node(or base-station). The core WSN requirements for this class of application is: time synchronisation, dissemination of data to all of the nodes, and data collection from the nodes. The current state of the art in WSN system design, similar to general computing systems, is that it uses separate protocols for each of these functionalities. Protocols will share lower level network stack information in order to optimise themselves (called cross-layer optimisation) but each protocol will have its own message type (or use multiple message types) and therefore incur its own radio overhead.

For example, the TinyOS (Levis et al., 2005) operating system comes with a code library, which contains many of the protocols used to enable the aforementioned services. TinyOS protocols commonly used for WSN are the Flooding Time Synchronisation Protocol (FTSP) (Maróti et al., 2004) to synchronise the sensor nodes and enable them to take time correlated samples, the Deluge protocol (Hui and Culler, 2004) to disseminate updated code images to the entire network, and the Collection Tree Protocol (CTP) (Gnawali et al., 2009) to forward data collected by the sensor nodes to a base-station for user processing.

Each protocol uses one or more message types, and has their own communication requirement. As we show, when combined, the summation of all of the communication required by all protocols means that collisions are likely and probable. Interference can occur at the node level with one protocol starving the other, and at the network level with nodes causing radio communication failure for other nodes. This results in communication starvation for the service protocols causing the protocols to fail.

We are not the only researchers to notice this phenomenon. Periodic heavy communication required by Deluge was shown to starve the Mint-Route data col-

Breza M., Yang S. and McCann J..
Multi-protocol Scheduling for Service Provision in WSN.
DOI: 10.5220/0004312800140022
In *Proceedings of the 2nd International Conference on Sensor Networks* (SENSORNETS-2013), pages 14-22
ISBN: 978-989-8565-45-7

lection protocol off the beacons it needed to enable data forwarding in (Hui and Culler, 2004; Langendoen et al., 2006). This caused the Mint-Route collection protocol to fail, delivering only 2% of the data sampled by the sensors. We have also observed the same problem, where high data collection data rates cause data dissemination to fail.

Two current approaches to solve this problem are the Fair Waiting protocol (Choi et al., 2007) and the Unified Broadcast layer (Hansen et al., 2011). The Fair Waiting protocol is based on the notions of fairness of channel usage. A scheduler keeps track of a protocols usage, the more a protocol is used, the less it gets scheduled. This mechanism was coupled with a MAC layer back-off mechanism, where a node would increase its medium access back-off time to reduce the chance of collisions with other protocols. The Unified Broadcast layer is a layer between the application layer and the network layer where all broadcast messages would be combined into one large message. This layer handles the marshaling and marshaling of the data and the delivery of the data to the correct protocol. It is transparent to the application layer protocols.

Clearly, some form of system is required to manage these protocol services. The challenge of this system is to maximise the use of the resources available to it - within its capacity region. We begin by showing the extent of the problem in section 2. Then our two pronged approach to this problem is presented. At the network layer we propose the use of a queue-length scheduling scheme to maximise node resources which we prove to be throughput optimal in section 3. We evaluate our scheduler and show that they outperform the current state of the art in sections 3.3 and 4. Finally we conclude in section 5.

2 THE EXTENT OF THE PROBLEM

To understand multiple-protocol conflicts we developed a WSN temperature sensing application in our laboratory. We used the state-of-the-art protocols provided in the TinyOS libraries so that the application would be representative of the current standard way to create a WSN application. The WSN consisted of 12 sensor nodes. The nodes sensed temperature and sent this to a base station in a multi-hop fashion, see Figure 1. We used CTP to collect data and route it to a collection base station, and Deluge to disseminate updated code images over the network. Both protocols were chosen as they are the most popular for their purpose. To create a baseline measurement, we observed the percentage of data received by the base station using CTP in isolation. CTP worked well, averaging 99% data collection at a rate of 25 data messages received at the base station every minute (the maximum capacity stated in (Gnawali et al., 2009)). We also ran the network with just Deluge. We found that Deluge performed very well, and that all the nodes were updated and rebooted within a maximum time of one minute.

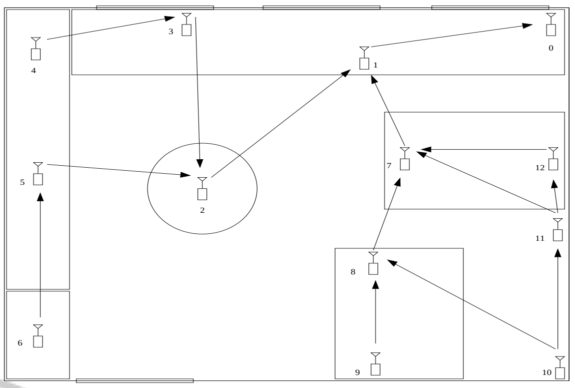

Figure 1: Schema of laboratory with node placement and desk locations.

We combined CTP and Deluge together and looked at the same metrics: data collection rate (for CTP), time to reboot, and percentage of the network rebooted (updated by Deluge). Our first attempt was with sampling and sending data once every minute. Deluge updates were sent once every 30 minutes. The data collection rate remained very high, around 99%, but all of the Deluge updates failed to occur with in a 30 minute window. When this period was extended to two hours, the updates still failed to occur. Reducing the CTP data send rate to once every 10 minutes resulted in a 99% success rate over a three hour period. Deluge also worked, but took eight minutes and six seconds for its first update and only succeeded in 10 out of 13 nodes used in the trial.

It is clear that when two protocols share the same network stack, there is a chance of disruption occurring to at least one of the protocols. This can be the result of problems at one of two points in the network stack, the MAC layer or the network layer. The MAC layer may be unable to provide the protocols with the bandwidth they need to function correctly. The network layer may not be scheduling the protocols in an efficient way, causing starvation to one or more protocols. In either case one or more protocols may not have the communication bandwidth which they require to function properly.

Experiments were performed in order to test the origin of the protocol failure we observed. To test the MAC layer, the number of collisions detected by the radio were recorded. This was done by measuring the CCA failures indicating the degree of network congestion. Problems at the network layer were measured by recording the number of packets dropped by the scheduling layer due to insufficient buffer space.

As results showed that there were no MAC layer or scheduler problems during steady state CTP operation when there were no Deluge updates. When Deluge was used, both MAC layer and scheduler problems were observed. There were four times more MAC layer collisions than messages dropped by the scheduler. In the worst case the MAC layer indicated 11 failures while the scheduler had four lost packets. More common results were four to six MAC layer errors with one or two scheduler errors. Errors occurred in 100% of the trials run with a data rate of a packet every 20 seconds or more. Trials performed with a data rate of less than 10 seconds failed to provide the bandwidth needed for both protocols, and Deluge failed to complete the dissemination of an entire code image to any node. We conclude that a scheme that both schedules the protocols and better shares network medium is required.

2.1 Network and Channel Model

We model a one-hop WSN as a fully-connected, directed graph $G(N,\ L)$. The set of all sensor nodes is N. All of the directional radio links between the nodes is the set L. Time is divided into equal length, non-overlapping time slots $t = 1,\ 2,\ 3, \ldots$.

We denote the set of all of the protocols as P. Each node maintains a message queue for each protocol in P. The number of messages in a queue at node $x \in N$ for protocol $p \in P$ at time slot t is $Q_x^p(t)$. All of the queue message populations for all of the protocols on all of the nodes is represented as the vector $\boldsymbol{Q}(t)$. This vector can be seen as a matrix of $|N|$ rows, one for each node, and $|P|$ columns, one for each protocol. The value of each location in $\boldsymbol{Q}(t)$ our $|N| \times |P|$ matrix is the corresponding number of messages in the queue, $Q_x^p(t)$.

At time slot t, every protocol $p \in P$ at every sensor node $x \in N$ adds messages to its queue with a rate $r_x^p(t)$(packets per slot). We assume that $r_x^p(t)$ is independent and identically distributed (i.i.d.) over all time with a finite second moment where the expected value of the square of the rate at which a protocol adds to its message queue is less than the square of the maximum finite message addition rate of all of the protocols.

$\mathbb{E}[(r_x^p(t))^2] \leq (\mathrm{r}^{\max})^2$. This assumption is safe for environmental monitoring applications.

The rates at which each protocol adds messages to its queues for all of the protocols on all of the nodes is represented as the matrix $\boldsymbol{r}(t)$. This can be seen as a matrix of $|N|$ rows, one for each node, and $|P|$ columns, one for each protocol. The value of each location in $\boldsymbol{r}(t)$ our matrix of size $|N| \times |P|$ is the corresponding rate of message addition to the queue, $Q_x^p(t)$.

2.2 Traffic and Data Queue Model

The rate at which a wireless link $(x,y) \in L$ can forward data, or send messages, at slot t is denoted as $c_{x,y}(t)$. As our links are directional, this is only the rate from node x to node y. The reverse rate may be different. We assume that $c_{x,y}(t)$ is independent and identically distributed (i.i.d.) over time (t), and with a finite second moment $\mathbb{E}[(c_{x,y}(t))^2] \leq (\mathrm{c}^{\max})^2$.

The broadcast capacity $CB_x(t)$ of a node $x \in N$ is the lowest rate $c_{x,y}(t)$ of all of that node's links to its neighbours.

$$CB_x(t) = \min_{y \in N - \{x\}} c_{x,y}(t), \forall x,y$$

The amount of data of a protocol p sent by a node x in a time-slot t is $f_x^p(t)$.

This leads us to the intuitive conclusion that the sum of all of the data sent by all of the protocols on a single node must be less that or equal to that node's broadcast capacity $\sum_{p \in P} f_x^p(t) \leq CB_x(t)$. The amount a node sends in a time-slot can not exceed its capacity.

The queue length $Q_x^p(t+1)$ is updated from time period t to $t+1$ by the equation:

$$Q_x^p(t+1) = \max(0,\ r_x^p(t) - f_x^p(t) + Q_x^p(t))$$

The queue length update equation is simply the number of messages added in the previous time slot minus the number of messages sent in that time slot, added to the queue length at the start of that time slot.

To avoid message loss due to simultaneous transmission of multiple nodes, only one node in N can transmit during a time slot. This is because our local area network $G(N,L)$ is fully connected. Any other nodes transmitting at that time will cause a collision.

Every node x has a contention-free transmission vector with $|N|$ dimensions called S_x. Each value of the vector corresponds to the CB_x of a given node $x \in N$ for all nodes in the network, including the local node. The x^{th} entry of the vector is for node x and is the broadcast capacity for that node. The vector S_x itself represents only node x broadcasting at the rate CB_x and all of the other nodes remaining silent. Therefore, the value of all of the other entries is zero.

The region Π is the convex hull over the $|N|$ dimensional space of all of the contention-free transmission vectors (S_x). The space is defined as the region where all nodes can broadcast without causing contention with one-another. The term a_x is the long term probability of the node x broadcasting.

$$\Pi = \{S \mid S = \sum_x a_x S_x,\ 0 \le a_x \le 1,\ \sum_x a_x = 1\}$$

So, no combination of node sending schedules which is outside of the space contained within the convex envelope Π can be scheduled by any scheduling policy. There will be contention and message loss.

2.3 The Network Capacity Region

The network capacity region is the set of all of the rates at which all of the protocols can add new messages into their queues ($\boldsymbol{r}(t)$) which can be scheduled successfully. This means that the queue sizes will not increase to infinity and that no messages will be lost. Message loss due to overflowing queues is the problem that we witnessed in our experiments above. When the Deluge protocol began to send data it found that the network capacity was not available to it, and it failed.

The long term data broadcasting rate of a node x is the sum of the long term data rates of all of its protocols p.

$$\overline{f_x} = \lim_{t\to\infty} \frac{1}{T} \sum_{p\in P} \sum_{t=1}^{T} f_x^p(t)$$

There is a vector $\bar{\boldsymbol{f}}$ of size $|N|$ which expresses the average data rate required by every node in the network. The value of the x^{th} location in the vector is the long term data rate of node x.

We formally define the network capacity region as Λ and say that the data rates required by all of the protocols are in the capacity region $\boldsymbol{r} \in \Lambda$ if there is a scheduling algorithm which can provide the required capacity.

$$\bar{\boldsymbol{f}} \in \Pi \tag{1}$$

$$\mathbb{E}[f_x^p(t)] \ge \mathbb{E}[r_x^d(t)] \quad \forall x \in N,\ p \in P \tag{2}$$

These two conditions mean that as long as the requirements of all of our protocols are in the capacity region, then it is possible to schedule them, and ensure that the protocols queues do not overflow and we do not lose messages. If the communication requirements of any of our protocols pushes our total data rate $\bar{\boldsymbol{f}}$ outside of the convex envelope of contention-free schedules, then no scheduling policy will be able to prevent message loss. If the protocol is not robust to long term message loss and communication failure, like we observed with Deluge when CTP was sending at a high data rate, then the protocol will fail.

3 THE GREEDY QUEUES SCHEDULER

The solution we propose follows from the analysis above and combines medium access control and protocol scheduling. The protocol obtains each node's link quality between a node and all its neighbours to maximise the chance of that message's successful reception by its neighbours. The queue length of the different protocols wanting to use a node's radio is used to determine the next protocol to have radio access.

At the beginning of every time slot t, every node x identifies the protocol queue with the largest number of messages $p_x^* \in P$. It then uses that value and its broadcast capacity $CB_x(t)$ to compute a weight $w_x(t)$.

$$w_x(t) = CB_x(t) \max_{p\in P} Q_x^p(t) \tag{3}$$

This weight is broadcast to all of a node's neighbours. When a node wants to broadcast a message, it consults its table of node weights. If it is the node with the highest weight x^* in the network at time slot t, i.e. $x^* = \arg\max_{x\in N} w_x(t)$ it will broadcast the next data packet for protocol p_x^*. The short back-off chosen by the node with the highest weight is less than the minimum of the randomly chosen range of the initial CSMA back-off to ensure that the chosen node communicates first.

$$f_x^p(t) = \begin{cases} \min(CB_x(t), Q_x^p(t)) & \text{if } x = x^*,\ p = p_{x^*}^* \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

If the node does not have the highest weight, then it will use the normal random CSMA backoff time, and send the next packet from the protocol with the largest queue length p_x^*.

The Greedy Queue scheduler is a single-hop scheduler. It is completely decentralised and uses only local information. Each node makes a local decision how to act (when to communicate, and with what protocol) based on local information and the information of its local area neighbours. Next, we will prove our protocol is throughput optimal as long as the network communication is within the network communication capacity.

3.1 Performance Analysis

This scheduling policy will ensure that the populations of all of the queues in the network do not exceed their limits, and that no protocol has to drop packets. This assurance can only be given as long as the data sending rates of the nodes are within the channel capacity policy previously defined as $\boldsymbol{r} \in \Lambda$.

Theorem 1. *Given arriving traffic* $\boldsymbol{r}$ *such that* $\boldsymbol{r} + \varepsilon \in \Lambda$ *for some* $\varepsilon > 0$*, our scheduling scheme will stabilise the network.*

Proof. To prove that our protocol queues will remain bounded, we define the Lyapunov function:

$$V(t) = \sum_{x \in N} \sum_{p \in P} (Q_x^p(t))^2 \tag{5}$$

Next, we consider its conditional expected drift:

$$\begin{aligned}
&\mathbb{E}[\triangle V(t)|\boldsymbol{Q}(t)] \\
&= \mathbb{E}[V(t+1) - V(t)|\boldsymbol{Q}(t)] \\
&= \mathbb{E}[\sum_{x \in N} \sum_{p \in P} ((r_x^p(t) - f_x^p(t) + Q_x^p(t))^2 - (Q_x^p(t))^2)|\boldsymbol{Q}(t)] \\
&\leq |N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2 \\
&+2\mathbb{E}[\sum_{x \in N} \sum_{p \in P} (r_x^p(t) - f_x^p(t))Q_x^p(t))|\boldsymbol{Q}(t)] \\
&\leq_a |N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2 - 2\varepsilon[\sum_{x \in N} \sum_{p \in P} Q_x^p(t)|\boldsymbol{Q}(t)]
\end{aligned}$$

The inequality $\leq_a$, is because of (2) and our max-weight scheduling scheme (4). Taking an expectation over $\boldsymbol{Q}(t)$ and a telescopic sum from $t = 1$ to T, we get:

$$\begin{aligned}
\mathbb{E}[V(T)] - \mathbb{E}[V(1)] \quad \leq \quad & T|N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2 \\
& -2\varepsilon \sum_{t=1}^{T} (\sum_{x \in N} \sum_{p \in P} \mathbb{E}[Q_x^p(t)])
\end{aligned}$$

Dividing both sides by T and taking an lim sup we see the long term expected queue length of each protocol on each node is less than or equal to a maximum bound determined by the protocol's message injection rate and the proximity of that value to the edge of the network capacity region.

$$\limsup_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{x \in N} \sum_{p \in P} \mathbb{E}[Q_x^p(t)] \leq \frac{|N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2}{2\varepsilon}$$

□

Then we see that as long as we remain in the capacity region, this bound is a finite number and therefore is less than infinity.

$$\frac{|N||P|(\mathrm{r}^{\max} + \mathrm{c}^{\max})^2}{2\varepsilon} < \infty$$

□

This proof clearly shows that as long as the long term rate at which all of the protocols on all of the nodes inject messages into the network is in the capacity region, the long term expected value (or the long term average) of all of queue lengths for all of the protocols on all of the nodes will be less than or equal to a finite value determined by the distance from the message injection rate to the edge of the capacity region. This finite value is less than infinity. Therefore proving that the Greedy Queues scheme is throughput optimal within a given capacity region provided by the system under consideration.

Please note that in our proof we use a lim sup but we could also just use a lim for two reasons. Firstly, we assume environmental monitoring applications with a stable data rate. Secondly, we are only concerned about the upper bound of the long term data rate, not the lower bound, because the lower bound will not cause our queues to overflow.

3.2 The Greedy Queues Scheduler Implementation

We implemented the Greedy Queues Scheduler for typical low power sensor nodes using TinyOS (Levis et al., 2005) on the MicaZ and Telosb. The first modification to the default TinyOS networking stack is to give each protocol a message queue. In the current Active Message implementation, each protocol gets its own queue with a length of one. This means that if the protocol tries to send another message, it receives an error message indicating that there is no more buffer capacity. We lengthen the message queue to 4 messages and implement it with FIFO semantics. A queue depth of 4 messages was chosen for so as to not consume too much memory. The second modification we made was to include the weight defined in 3 in the broadcast message header. The weight is calculated using the packet reception ratio (PRR), that is, the ratio of packets sent to a neighbours over packets received by that neighbour using the four bit link estimator (Gnawali et al., 2009).

3.3 Single-hop Evaluation

Our aim is to enable the overlapping protocols to function as best possible within the network's communication capacity region. To test the performance of Greedy Queues we constructed a WSN running a temperature sensing application. The sensor nodes sense temperature and send data with a predefined frequency to a base-station for logging.

We used a network of MicaZ motes with CC2420 radios. The transmission power level was set to full. This gave us 0dBm of output power, and ensured that all of the nodes in laboratory were connected to all of the other nodes. The nodes had a minimum of one meter distance between the aerials to prevent near field radio interference. The size of the network was varied from four nodes to 32 nodes.

We used FTSP for time synchronisation (Maróti et al., 2004), Deluge to disseminate new code images (Hui and Culler, 2004), and CTP (Gnawali et al., 2009) for data collection, to represent a typical environmental monitoring application.

Protocol performance is application specific. For Deluge it is the portion of the network that receives and acts upon an update. We measured the time from starting the dissemination, to the time the node rebooted. For the data collection protocol, CTP, we measure the percentage of sensor data packets delivered to the base-station. The results are an average of 10 experimental runs lasting three hours each. The data is taken from message log of data sent from the base-station to a PC via serial port. For FTSP we measure the number of messages with synchronised time stamps.

Temperature samples are sent to the base-station every second. This data rate represents a heavy work load. Once the network settles down and the collection routes are established, Deluge is used to disseminate a new code image throughout the network. The code image consists of 38 pages, each page being 1024 bytes in size - the default page size for Deluge.

3.4 Single-hop Results

Since these experiments represent the search for a scheduling policy which will allow multiple communication protocols to co-exist, it is important to take the three graphs together. The first graph (Figure 2) shows the average time which the Deluge reprogramming protocol takes to reprogramme our test-bed. The times to disseminate need to be taken with graph (Figure 3) which shows how increasing neighbour population affects the percentage of successful dissemination trials for each scheduler. In cases where no dissemination events were successful, the time to disseminate drops to zero. When no dissemination was successful, Deluge completely failed. The third graph (Figure 4) shows CTP success rates as the network

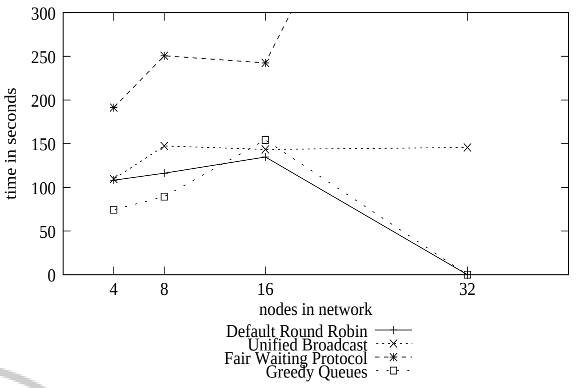

Figure 2: Plot of the time for each protocol to disseminate an entire Deluge binary.

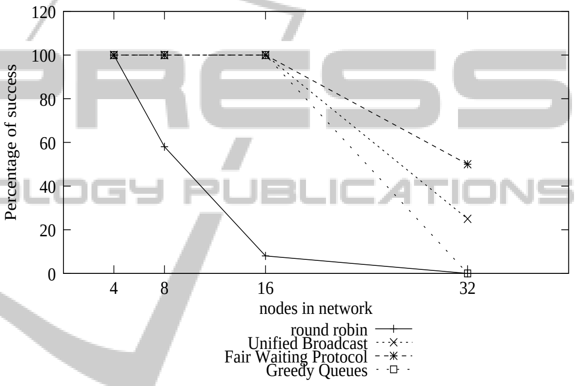

Figure 3: Plot of the percentage of trials where the Deluge binary was successfully disseminated.

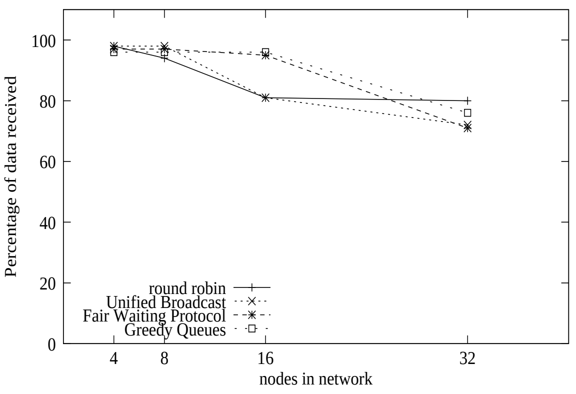

Figure 4: Plot of the percentage of data message delivered using CTP during the Deluge dissemination.

becomes congested and Deluge fails.

We see that Deluge works well with all of the schedulers when the local neighbourhood population is only four nodes. The time to disseminate is fairly similar for all. The Fair Waiting Protocol works, but is almost twice as slow as the others. The percentage of successful dissemination is 100% for all of the scheduling policies, and all of the packets sent by the sensing nodes are received by the CTP base station. These results give us a base case, where both protocols run well as we are well within the capacity region.

When the population of the network is doubled to eight nodes, the performance of the schedulers begin to vary. The Unified Broadcast scheme hits an upper bound for dissemination time at this point. In subsequent trials its dissemination time remained fixed. The time for FWP to disseminate increases to an average of 250 seconds, taking it off of the graph. The Greedy Queues scheduler has the lowest dissemination time, 10% faster than the default Round Robin scheduler. All protocols successfully complete 100% of the Deluge trials except for the default Round Robin scheduler. The percentage of data messages received by the CTP base station is still very high for all scheduler, in the 94% range.

The performance at a network population of 16 nodes is still good. The Deluge dissemination time for all schedulers remains constant, or increases only by a small amount. The percentage of Deluge trials successfully completed is still 100% for all schedulers except the default Round Robin scheduler, it succeeds in only 5% of its trials. The percentage of CTP data messages received by the base station remains constant from the previous network populations for the Greedy Queue scheduler and the Fair Waiting Protocol, and degrades by about 15% for the default Round Robin scheduler and the Unified Broadcast layer.

At 32 nodes the phenomenon of protocol failure becomes apparent. At this point the schemes are trying to operate outside of the capacity region, as defined in section 2.3, that is, beyond the constraints of any scheduler. The time taken to disseminate Deluge updates increase for the Fair Waiting Protocol to almost ten minutes, but only 50% of the disseminations are successful as can be seen in Figure 3. Unified Broadcast fared worse with about 30% success, taking the same time as for 16 and 8 nodes. The Greedy Queue and the default Round Robin schedulers have no successful dissemination attempts. In the case of the Greedy Queue scheduler this was expected. The throughput optimality of our scheduler is only defined while the nodes send data at a rate which remains in the capacity region.

The reception averages for CTP are for the most part good. The Fair Waiting Protocol and Unified Broadcast both manage about 70% data packet reception. The Greedy Queues protocol is more successful and manage approximately 75% data reception.

In all cases the FTSP protocol functions properly. This is because in a single-hop environment only the FTSP synchronisation root will broadcast a sync beacon once every three seconds and therefore requires very little communication overhead to function properly.

These results show us that we are able to create a scheduling layer using only local information to enable multiple protocols to function with good performance at high data rates. As long as the rate rates required by the protocols remains in the capacity region, our greedy approach is able to use the time previously lost to initial CSMA back-off periods, and successfully schedule communication in that period. The Greedy Queues scheduler shows the fastest time to disseminate new Deluge code images. It also maintains a data delivery rate equal to or better than those of the other schedulers, even beyond the network communication capacity region.

4 MULTI-HOP EVALUATION

The Greedy Queue scheduler was designed to optimise local, single hop communication. We wanted to see its effect on multi-hop communication scenarios. For these experiments we decided to only evaluate Greedy Queues against the default TinyOS round robin scheduler. This is because the Fair Waiting Protocol only adds delays to reduce contention, and in a multi-hop environment we found that the delays compounded to make the system performance very slow. The Unified Broadcast network layer did work very well, but it worked by combining broadcast messages and increasing the default message payload size from 28 bytes to 60 bytes to aim message combination and reduce network traffic. This method is not a scheduling approach, and therefore we do not compare our scheduler against it.

4.1 In Laboratory Testbed

Our first multi-hop experiments used the same experimental setup as for the one hop networks with 16 nodes. The radio transmission power was reduced to power level 5 on the CC2420 radio used by the MicaZ. This gave us about -20dBm of output power and created a network with a maximum hop depth of 4 hops to the collection base station.

We used CTP to do data collection, and the nodes sent at the rate of one data message every eighth of a second. This rate was chosen by experimentation to try and be as close to the edge of the capacity region of the network as possible. A separate Deluge base-station was used to inject and disseminate new code images into the network because it is difficult to have one base-station handle both dissemination and collection. Each code image was 38 pages, and each page was 1024 bytes. The time was measured from the time at which the deluge command was issued, to the time at which the last node began its reboot. The results are shown in Table 1.

We can see from this result that the average time to disseminate is 43% faster with the Greedy Queues scheduler than with the default TinyOS round robin scheduler. Data collection is also improved by four percent. In all cases the FTSP synchronisation protocol functioned properly. This experiment shows that although the Greedy Queue scheduler only uses local information to determine the protocol to send and the backoff delay, it still provides benefits over the standard scheduler in a multi-hop network.

4.2 Remote Testbed

Our next set of experiments deployed the same set of protocols on the remote wireless sensor testbed Indriya (Doddavenkatappa et al., 2012). The testbed contains 138 functioning nodes upon which these experiments were run. This facility allows us to evaluate our protocol in a multi-hop environment, where the population of the nodes is fixed. To vary load we increased the sending rate of CTP, and then periodically disseminated large amounts of data using a Polite broadcast dissemination protocol similar to Deluge. It was not possible to run the full Deluge protocol on the testbed because it does not allow the use of the tosboot boot-loader. The boot-loader is used to reboot a node with a new code image, and is an integral part of Deluge.

4.3 Multi-hop Evaluation

We can see in Figure 5 that as the rate of data sent by each node (represented in Figure 5 as the number of messages sent per second) decreases, the percentage received by the base-station diminishes. When CTP is on its own, not sharing the network with other protocols, its performance begins to degrade at the rate of a data message sent every two seconds. At a message every second, the base-station receives less than 45% of the data messages sent.

The Greedy Queues and the round robin TinyOS scheduler both have very similar performance to CTP on its own up to the rate of one message every five seconds. At a message every three seconds, both scheduling policies lose the same percentage of data messages. At higher data rates, the default Round Robin scheduler loses on average 5% more messages that the Greedy Queue scheduler. The greedy approach taken by our scheduler means that sensor

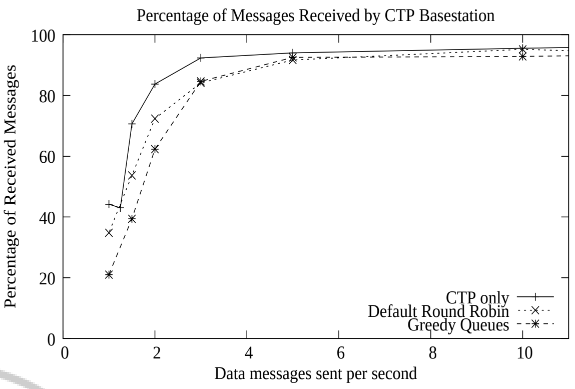

Figure 5: Percentage of data packets received by CTP per data sending rate.

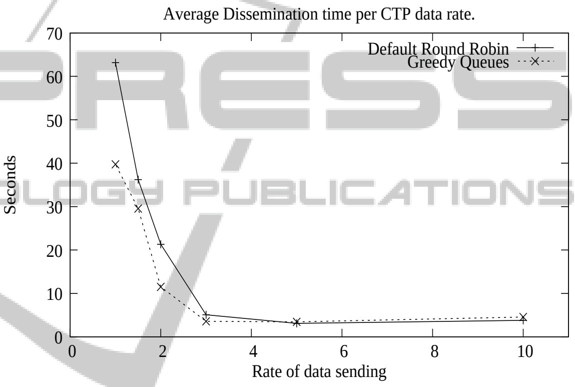

Figure 6: Average data dissemination times using Polite Flooding.

nodes with the highest weight transmit with no delay, and with a very low chance of interference. The removal of the default CSMA back-off time allows us to use the channel as efficiently as possible.

The data dissemination results in Figure 6 show that under a heavy data communication load, the Greedy Queue scheduler reduces dissemination time by a minimum of 10%. The times are equal at lower CTP data rates. At one message every three seconds the dissemination messages start to get delayed. One data message every two seconds shows that the Greedy Queue scheduler has approximately 48% less delay than the default Round Robin scheduler. By the CTP data rate of one message every second, the Greedy Queue scheduler is 35% percent faster than the default Round Robin scheduler.

The synchronisation results are similar. The percentage of time 90% of network is in synchronised state was higher for the Greedy Queue scheduler across all data rates.

The Multi-hop evaluation results are similar to those which we saw in the single-hop evaluation. Once again the percentage of data packets received

Table 1: The average percentage of data packets received by CTP.

	dissemination time	avg packets collected
Standard	431.5 sec.	80%
Greedy Queues	242.67 sec.	84%

at high data rates is better with the Greedy Queue scheduler. Under the same conditions, the average dissemination times are lower than for the default Round Robin scheduler. These results show us that as the network communication traffic increases, the Greedy Queue scheduler can provide the network requirements of each protocol better than the default Round Robin scheduler currently in use.

5 CONCLUSIONS

Current environmental WSN systems are made by combining several different protocols to provide the services required by the application. This naive combination can cause disruption or failure to one or more of the protocols. Here, we propose a combined protocol and radio access scheduler which uses local information about protocol queue lengths and link capacities to enable multiple protocols to co-exist and operate optimally within the network communication capacity region.

We prove that our Greedy Queue scheduling scheme is throughput optimal as long as the data rates required by the protocols are within the capacity region of the network. Through evaluation we show that it outperforms the current state of the art in a single-hop network. We also demonstrate the same performance gains in a multi-hop network. For future work we would like to examine the affects of fairness on the performance of the Greedy Queue scheduler.

REFERENCES

Cardell-Oliver, R., Kranz, M., Smettem, K., and Mayer, K. (2005). A Reactive Soil Moisture Sensor Network: Design and Field Evaluation. *International Journal of Distributed Sensor Networks*, 1(2):149–162.

Choi, J., Lee, J., Chen, Z., and Levis, P. (2007). Fair waiting protocol: achieving isolation in wireless sensornets. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 411–412. ACM.

Doddavenkatappa, M., Chan, M., and Ananda, A. (2012). Indriya: A low-cost, 3d wireless sensor network testbed. *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 302–316.

Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and Levis, P. (2009). Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM.

Hansen, M., Jurdak, R., and Kusy, B. (2011). Unified broadcast in sensor networks.

Hui, J. and Culler, D. (2004). The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94. ACM.

Langendoen, K., Baggio, A., and Visser, O. (2006). Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8–pp. IEEE.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. (2005). TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence*, pages 115–148.

Maróti, M., Kusy, B., Simon, G., and Lédeczi, Á. (2004). The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM.

Martinez, K., Padhy, P., Elsaify, A., Zou, G., Riddoch, A., Hart, J., and Ong, H. (2006). Deploying a Sensor Network in an Extreme Environment. *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, 1.

Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., and Welsh, M. Monitoring volcanic eruptions with a wireless sensor network. *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 108–120.