

# Unified Broadcast in Sensor Networks

Morten Tranberg Hansen  
Dept. of Computer Science  
Aarhus University  
mth@cs.au.dk

Raja Jurdak  
Autonomous System Lab.  
CSIRO ICT Center  
Raja.Jurdak@csiro.au

Branislav Kusy  
Autonomous System Lab.  
CSIRO ICT Center  
Brano.Kusy@csiro.au

## ABSTRACT

Complex sensor network applications include multiple services such as collection, dissemination, time synchronization, and failure detection protocols. Many of these protocols require local state maintenance through periodic broadcasts which leads to high control overhead. Recent attempts to consolidate these broadcasts focus on piggybacking information into existing services but such tight coupling between protocols limits code reuse and interoperability of applications.

We present Unified Broadcast (UB) which combines broadcasts from multiple protocols while maintaining a modular architecture of the network stack. UB is implemented as a transparent layer between the link and network layers, where it delays, schedules, and combines broadcasts from upper layer protocols before transmission on the wireless channel. Our empirical results in simulation and on a testbed show that UB can decrease the overall packet transmissions in the network by more than 60%, corresponding to more than 40% energy savings, without requiring new interfaces or affecting the correctness of the upper layer protocols.

## Categories and Subject Descriptors

D.2.1 [Computer-Communication Networks]: Network Architecture and Design

## General Terms

Design, Experimentation, Performance, Standardization

## Keywords

broadcast, unifying abstraction, link protocol, network protocol, wireless sensor network

## 1. INTRODUCTION

Recent advances in wireless and sensor technologies have enabled wireless sensor networks applications that go beyond the simple data collection paradigm [10]. Deployed

sensor networks can now provide end-to-end reliability [13], global time synchronization [15], wireless network programming [11], and health monitoring [18]. Each of these features typically relies on a stand-alone protocol that uses broadcast messages for state maintenance. With current sensor network operating systems, each broadcasting protocol schedules its broadcasts, independently of all other broadcast protocols, for immediate transmission, without coordinating with other protocols. This means that the total control overhead is the sum of the control overhead of the protocols. Thus, an increase in the number of broadcast protocols results in an increase in the amount of control traffic and a faster depletion of the node's energy resources.

To meet energy requirements in a multi-protocol application, some protocols restrict themselves from explicitly sending broadcasts and instead piggyback their data into broadcasts from other protocols [9, 18]. Theoretically, one could tailor any specific application to combine as many broadcasts as possible based on a fixed set of protocols used. However, such tight coupling between the protocols limits the code reuse and interoperability of the application [12] and is undesirable in more complex applications.

We present Unified Broadcast (UB), a transparent protocol layer that combines broadcast packets from multiple upper layer protocols while maintaining a modular network stack architecture. The basic premise of UB is to exploit the tradeoff between the energy efficiency of radio communication and the latency of packet delivery. The same information can be transmitted with less overhead when combined in a single packet as compared to multiple fragmented packets. In fact, the difference between broadcasting long and short packets becomes negligible when Low Power Listening (LPL) (e.g., BoXMAC2 [16]) is enabled, as the packet simply gets retransmitted multiple times for the duration of the LPL period.

UB sits between the link and the network layer where it seamlessly intercepts broadcasts from upper layer protocols and automatically combines them in a single broadcast packet on the wireless channel. This provides UB with backward and forward compatibility with any protocol that uses broadcast messages.

Delaying packets of upper layer protocols, however, can adversely impact the correctness of the protocols. UB, therefore, does not delay the packets indefinitely. If the same protocol wants to broadcast another packet, UB pushes out the unified packet even before its content reaches the maximum length. We show through extensive simulations and experiments that this simple approach, which is a tradeoff between

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.

*IPSN'11*, April 12–14, 2011, Chicago, Illinois.

Copyright 2011 ACM 978-1-4503-0512-9/11/04 ...\$10.00.

energy and latency, preserves correctness of the majority of a set of representative wireless sensor network protocols. For those protocols that might be negatively affected by the increased latency, UB bends its goal of transparency and implements a simple extension that allows the upper layer protocols to force immediate transmission of its packets.

We demonstrate the savings that UB enables through analysis, simulations, and testbed experiments. Clearly, sensor network applications that use larger number of protocols can potentially achieve higher savings with UB. We first analyze the relationship between savings and protocols qualitatively through an offline analysis. Our analysis shows that UB saves up to 50% control overhead combining just three sensor network protocols. Encouraged by these results we evaluate our TinyOS implementation of UB quantitatively in simulations and a real sensor node testbed using various combinations of four popular sensor network protocols. We show that UB can achieve up to 60% improvement in the control overhead of sensor network applications without affecting the correctness of the upper layer protocols.

The contributions of this paper are:

- Proposal of a unified broadcast layer to combine scheduled broadcast from multiple protocols within a wireless sensor node.
- Performance evaluation of UB through analysis, simulation, and testbed experiments to quantify its savings in terms of packet transmissions and energy.
- Empirical validation of UB’s functionality and savings on a set of representative network protocols, including periodic, adaptive periodic, many-to-one and one-to-many protocols.

The remainder of the paper is organized as follows. Section 2 provides background on common sensor network protocols that rely on regular broadcasts, and presents related architectural approaches that have taken steps to consolidate protocol broadcasts. Section 3 presents an overview of UB, and Section 4 motivates its use through offline and theoretical analysis of protocol broadcasts. Section 5 focuses on our UB implementation in TinyOS, while Section 6 evaluates UB in simulation and testbed experiments. Section 7 discusses the results and concludes the paper.

## 2. BACKGROUND AND RELATED WORK

This section provides an overview of common sensor network protocols that will benefit from UB, and discusses UB related design coupled and architectural approaches.

### 2.1 Common Protocols

Sensor network applications can include various protocols for managing data collection, time synchronization, code dissemination, and anomaly detection, among others. Each active protocol relies on its own scheduling of packet transmissions, a combination of unicast and broadcast packets, to achieve its desired functionality. Broadcast transmission periods in particular can either be static or can change dynamically during the lifetime of a deployment. Here, we briefly present a representative set of common sensor network protocols, both with static and dynamic broadcast periods, in order to expose the opportunities for unifying their broadcasts.

#### 2.1.1 Static Period Protocols

##### *MultiHopLqi.*

MultiHopLqi (LQI) [2] is a widely used collection tree protocol for sensor networks. It uses periodic broadcasts to distribute a node’s aggregated link quality to the root node. The broadcast period is fixed, but randomly offset at the boot up to avoid packet collisions between neighbors. An LQI node maintains its own cost, its parent’s cost to the root, as well as its last seen broadcast link quality indicator from the parent. Upon reception of a new broadcast from a neighboring non-parent node, a node (re)elects its parent by comparing the cumulative path cost of the new neighbor to its current path cost, and selects the path with the minimum cost.

##### *FTSP.*

The Flooding Time Synchronization Protocol (FTSP) [15] is a widely used global time synchronization protocol for resource-restricted sensor networks. In FTSP one node is dynamically (re)elects as the root of the network which maintains a global clock with which other nodes synchronize. The synchronization is achieved by per node periodic broadcasts, where the radio layer performs the actual time stamping to minimize the error. The FTSP broadcast period is fixed, but each node adds a slight randomization to its period upon booting to avoid repeated collisions between neighboring nodes.

##### *Memento.*

Memento [18] is a health monitoring system for wireless sensor networks. It consists of two parts: a protocol to deliver state summaries to a root node and a set of failure detection mechanisms. Its failure detection mechanisms are based on the number of periodic heartbeats heard from neighboring nodes within a defined sweep period. The default mechanism, called Variance-Bound, takes a target false positive rate as input to adapt to packet losses. Variance-Bound estimates the maximum number of consecutively missed heartbeats from a neighbor within a sweep period. Memento’s standard configuration sets the sweep period to three times the heartbeat rate. Memento uses a tree-based state collection mechanism where a node keeps an aggregated state of its own local state and that of its children. For energy efficiency a node only propagates its state up through the collection tree when it changes. This enables the root to have a global view of the network state. For improved failure detection Memento suggest to expand the set of monitored nodes from the node’s subtree to also include its neighbors.

#### 2.1.2 Dynamic Period Protocols

##### *Trickle.*

One of the most popular techniques for dynamic period broadcasts in the sensor networks community is the Trickle timer, which was originally designed as a protocol for maintaining code updates in sensor networks [14]. A Trickle timer uses dynamic adjustment of the broadcast timer and beacon suppression to reduce the control overhead of keeping neighboring nodes up-to-date. When neighbors are up-to-date, the broadcast period exponentially increases from a fixed minimum broadcast period to a fixed maximum broad-

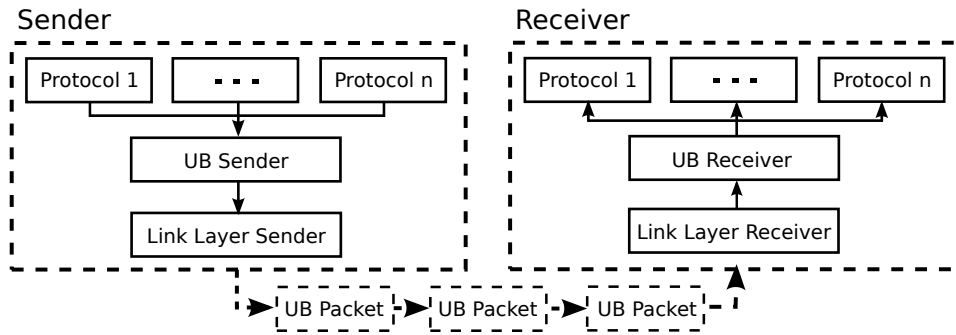


Figure 1: Overview of UB showing the sending and the receiving side.

cast period. Furthermore, a node suppresses its own broadcast if it overhears a minimum number of similar broadcasts from neighbors. When neighbors are not up-to-date, they reset the broadcast period to the minimum broadcast period. To avoid broadcast period alignment among neighbors, each node randomizes its actual broadcast time within the broadcast period. Many network layer protocols adopt a variation of the Trickle timer as an alternative to the non-adaptive periodic timer.

### CTP.

The Collection Tree Protocol (CTP) [10] is a well-tested tree routing protocol that supports any-cast data collection to a set of root nodes in a static sensor networks. In CTP, sensor nodes distribute their expected number of transmissions (ETX) to a root node through regular broadcasts. A sensor node then chooses its parent based on the minimum ETX to a root node. To limit the amount of broadcasts, CTP uses adaptive broadcasts based on the Trickle timer without suppression. Hence, the broadcast period exponentially increases from a minimum broadcast period to a maximum broadcast period until the period is reset. CTP resets its period in three cases: when a pull bit of an incoming control packet is set; a routing loop is detected; or if the ETX estimate drops significantly. CTP uses a pull bit on control traffic whenever it is disconnected from the routing tree i.e. does not have a parent node. Loops are detected based on data path validation by prepending all data packets with the ETX of the sender. A loop is then assumed at the receiver if this ETX is less than or equal to its own ETX. CTP transmits its broadcast packets at a random time during the second half of its Trickle timer period.

### Deluge.

Deluge [11] is a wireless network re-programming protocol for sensor networks. In Deluge, a sensor node uses a standard Trickle timer with suppression to periodically advertise its code version in an energy efficient way. Once code inconsistency is detected, Deluge uses a unicast stream of packets to update the outdated node. The suppression factor in Deluge is set to 1, which means that a node will suppress its own broadcast if it hears a similar broadcast from any neighboring node.

## 2.2 Design-Coupled Approaches

Several proposals have recognized the need for combining broadcasts for multiple sensor network protocols for im-

proved energy efficiency. Memento [18] embeds its heartbeat messages within the underlying routing protocol's periodic broadcasts and the four bit link estimator [9] appends its link estimate to CTP beacons. However, these approaches tightly couple their interfaces which limits modularity. In general, requiring higher layer components to change their existing interfaces in order to use broadcast combining features is less versatile and risks backward incompatibility.

More recently, a UB related announcement layer has been proposed for broadcast coordination in sensor networks [5]. The announcement layer is implemented as a service, defining a fixed interface where protocols can register to get their data broadcasted at a minimum rate. Thus, existing protocols would have to be modified in order to use this announcement layer. Alternatively, UB is implemented as a transparent protocol layer that does not require modification of upper layer protocols, and is evaluated both based on savings and on how it affects upper layer protocol performances. The latter is especially important with respect to the announcement layer which, opposed to UB, does not guarantee an upper bound on the number of times a certain broadcast can be sent.

## 2.3 Architectural Approaches

Other approaches for reducing protocol redundancy in sensor networks follow a more architectural approach. The Sensor Protocol (SP) [17] proposes a layer 2.5 unified link abstraction to consolidate all link layer communication, as well as a single neighbor table. Research has built on this and proposed a modular network layer [8] for better code reuse in composing new network protocols, providing typical network layer functions, such as (1) packet forwarding, queueing, scheduling, and fragmentation, (2) route discovery and maintenance, and (3) addressing. Chameleon [6] focuses on separation of packet formats from protocols and applications. This is done through the Rime stack, which provides communication primitives to higher layer protocols, and the Chameleon component, which maps packet attributes to packet formats for the MAC protocols in use. All three architectural proposals focus on streamlining the interfaces to and between layers 2 and 3 for more efficient composition and functionality of complex sensor network applications. UB shares the goals and spirit of these approaches, and it complements them through its introduction of a unified broadcast layer that seamlessly combines broadcasts from higher layer protocols and relays the resulting packets to the MAC layer.

	CTP	LQI	FTSP	Deluge	Memento
$\tau_{min}$	64ms	32s	30s	2s	30s
$\tau_{max}$	3600s	32s	30s	60s	30s

**Table 1: Protocol configuration parameters taken from papers proposing the protocols.**

### 3. DESIGN OVERVIEW

The general principle of Unified Broadcast (UB) is simple: combine as many broadcasts as possible without affecting the upper layer protocols. UB’s approach takes advantage of the asynchronous best-effort nature of broadcasts that do not require synchronous acknowledgements.

An overview of UB is shown in Figure 1. At the sending side, UB delays broadcasts from upper layer protocols, in order to embed multiple of such broadcasts into one UB packet. Upper layer protocols might rely on protocol specific sequence numbers [10] or physical layer per packet parameters [2]. Thus, to minimize the influence on upper layer packets UB never combines two packets from the same protocol and guarantees that all packets from upper layer protocols are sent. To maximize savings UB delays a packet as long as possible until either an optional fixed maximum allowed delay is exceeded, or a protocol with an already delayed packet sends another one. Once a UB broadcast is to be sent, all currently delayed packets are embedded in a single UB broadcast packet and transmitted on the channel. At the receiving side, UB packets are unpacked and delivered to the upper layers.

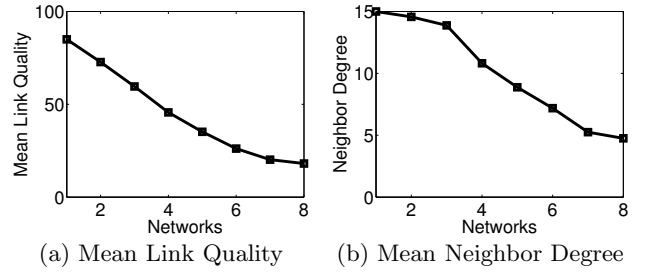
Specifically, we consider the following set of representative upper layer protocols: CTP and LQI for data collection, FTSP for global time synchronization, Deluge for network reprogramming, and Memento for failure detection. These protocols cover most of the design choices that network protocols adopt: static and dynamic broadcast periods, proactive and reactive operation, and packet suppression. Table 1 provides an overview of the protocol configurations we use in this paper. These settings match values in the original papers that proposed these protocols. However, in the few cases (LQI and Memento) where the papers did not have a clear indication for these settings, we adopt the default TinyOS source code configuration (LQI) or the most reasonable one based on the papers evaluation (Memento).

### 4. POTENTIAL SAVINGS

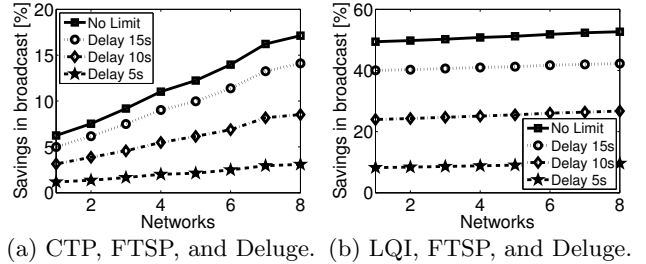
Before we implement and evaluate UB in real sensor networks, we estimate the potential savings of UB under idealized conditions.

#### 4.1 Offline Analysis

To estimate the potential savings of UB, we built a few simple TinyOS applications and recorded network traces during their operation. The applications include CTP/LQI, FTSP, and Deluge protocols configured with the parameters from Table 1. We simulated these applications in TOSSIM, on eight different 16-node topologies. The TOSSIM networks were 4x4 grid networks with increasing inter-node distances where links are generated with the TinyOS link-layer model for static and low dynamic networks [4]. The collection root is placed in a corner of all the TOSSIM networks. We ran 5 simulation experiments lasting 6 hours for each network. Figure 2 highlights the topological differences



**Figure 2: Mean node neighbor degree and mean link quality of all links with  $PRR > 0$  in 8 TOSSIM networks. All networks are 16 node grid networks.**



**Figure 3: Mean expected broadcast savings of UB using offline analysis of broadcasts sent in different networks with two different set of protocols.**

between the networks. Figure 2(a) shows that networks 1 through 8 have a progressively degrading mean link quality, while Figure 2(b) illustrates that the degrading link qualities naturally lead to smaller neighbor degrees (a less-connected network). We log the protocol and timestamps of all broadcasts sent and received from all nodes during each simulation run. Based on this log, we performed an offline analysis of the possible savings using unified broadcast by counting the number of broadcasts that could be combined without combining two broadcasts from the same protocol or delaying a broadcast more than a fixed maximum delay.

Figure 3 shows the potential savings in broadcasts using UB in the 8 TOSSIM networks with decreasing connectivity. Both Figures 3(a) and 3(b) show that the savings increase with the maximum time a broadcast is allowed to be delayed. Thus, for maximum savings using UB one should not limit the maximum delay of a broadcast, and hence only send broadcasts whenever a protocol with an already delayed packet sends another one. Furthermore, Figure 3(a) suggests that the savings for this specific combination of protocols increase with decreasing network connectivity. This can be caused by CTP scheduling more broadcasts due to sudden changes in route costs or detection of loops, or Deluge suppression being less effective in a weakly connected network. To analyze the exact causes, the next subsection studies the influence of these parameters on potential savings in detail.

#### 4.2 Maximizing the Savings

If broadcasts from two protocols are perfectly aligned with the same periods, all broadcasts can be combined. Thus, the maximum savings would be 50%. Likewise, if the broadcasts from three protocols were perfectly aligned, one could

send the broadcasts from two of the protocols with the third one, and hence the maximum savings in broadcasts would be 66%. In general, the maximum savings in broadcasts can be expressed as:

$$savings = \frac{\#protocols - 1}{\#protocols}$$

However, as we saw in the previous section network dynamics and advanced protocol behavior reduce the savings and their predictability. In this section, we simplify the notion of network dynamics and analyze the maximum savings possible with any combination of protocols.

We assume all protocols can be modeled by a Trickle timer (or simplification hereof). A special case of a Trickle timer is a periodic broadcast timer which is a Trickle timer without suppression and the maximum broadcast period set equal to the minimum broadcast period. Consequently, we define a protocol  $p$  as a four tuple  $\langle \tau_{min}^p, \tau_{max}^p, \theta^p, \rho^p \rangle$  where  $\tau_{min}^p$  is the minimum broadcast period,  $\tau_{max}^p$  is the maximum broadcast period,  $\theta^p$  is a fixed reset period after which the broadcast period is reset to  $\tau_{min}^p$ , and  $\rho^p$  is the probability of suppressing a broadcast. It is important to note that the fixed reset period is a simplification of the sporadic resets that would happen in a real deployment. Furthermore, the probability of suppression is a deployment specific parameter, which depends on the average node degree and the number of packets a node has to overhear before suppressing its packet.

For a protocol  $p$  we can derive the number of broadcasts required for the broadcast period to reach its maximum  $\tau_{max}^p$  without any resets as:

$$b_{max}^p = \log_2 \frac{\tau_{max}^p}{\tau_{min}^p} + 1, \quad (1)$$

Any broadcast after this will occur with a period of  $\tau_{max}^p$ . Based on the broadcasts,  $b_{max}^p$ , we can derive the time it takes for a protocol  $p$  to send  $b$  broadcasts without any resets as:

$$t_b^p = \begin{cases} \tau_{min}^p (2^b - 1) & \text{if } b \leq b_{max}^p, \\ \tau_{min}^p (2^{b_{max}^p} - 1) + \tau_{max}^p (b - b_{max}^p) & \text{if } b > b_{max}^p, \end{cases} \quad (2)$$

By substituting  $b_{max}^p$  from Equation 1 for  $b$  in Equation 2 we define  $t_{b_{max}^p}^p$  to be the time it takes to reach the maximum broadcast period. From this, we can derive the number of broadcasts a protocol  $p$  can broadcast within time  $t$  without any resets as:

$$b_t^p = \begin{cases} \lfloor \log_2(\frac{t}{\tau_{min}^p} + 1) \rfloor & \text{if } t \leq t_{b_{max}^p}^p, \\ \lfloor \log_2(\frac{t - t_{b_{max}^p}^p}{\tau_{min}^p} + 1) \rfloor + \lfloor \frac{t - t_{b_{max}^p}^p}{\tau_{max}^p} \rfloor & \text{if } t > t_{b_{max}^p}^p. \end{cases} \quad (3)$$

The number of broadcasts a protocol can send before being reset by its reset period  $\theta^p$  is  $b_{\theta^p}^p$  and the number of times a protocol has been reset up to a certain time  $t$  is  $\frac{t}{\theta^p}$ . Thus, we define  $send_t^p$  to be the number of accumulative broadcasts a protocol  $p$  wants to send up to and including time  $t$  without considering suppressions as:

$$send_t^p = \lfloor \frac{t}{\theta^p} \rfloor b_{\theta^p}^p + b_t^p \pmod{\theta^p}. \quad (4)$$

Let  $x_1, \dots, x_{t_{max}}$  be an identically distributed random variable sample from  $U(0, 1)$  and  $broadcast_t^p$  be a binary indicator of whether or not protocol  $p$  broadcasts at time  $t$ , defined

as:

$$broadcast_t^p = \begin{cases} send_t^p & \text{if } t = 1, \\ send_t^p - send_{t-1}^p & \text{if } t > 1 \wedge x_t > \rho^p, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Note that  $broadcast_t^p$ , as opposed to  $send_t^p$ , considers suppression of broadcasts.

We assume that time is divided into a set of discrete units according to the minimum granularity of any of the protocols considered, and that any simultaneous broadcasts that happen at time  $t$  will be processed at the same time by UB. Let  $pending_t^p$  indicate whether or not protocol  $p$  has a pending transmission at time  $t$ . We then define  $ub_t$  to indicate whether or not a unified broadcast is to be sent a time  $t$ . In this subsection we do not consider a fixed maximum delay of broadcasts, so a unified broadcast is sent at time  $t$  if any protocol  $p$  wants to broadcast at time  $t$  while already having a pending broadcast:

$$ub_t = \exists p : broadcast_t^p \wedge pending_{t-1}^p \quad (6)$$

A protocol has a pending broadcast at time  $t$  if:

1. no unified broadcast is sent and it has a broadcast at  $t$ ,
2. no unified broadcast is sent and it has a pending broadcast from  $t - 1$ , or
3. a unified broadcast is sent with its pending broadcast from  $t - 1$  and it has a broadcast at  $t$

We define  $pending_t^p$  as:

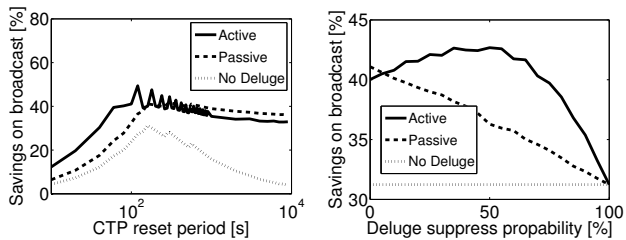
$$pending_t^p = \begin{cases} broadcast_t^p \vee pending_{t-1}^p & \text{if } ub_t = 0, \\ broadcast_t^p \wedge pending_{t-1}^p & \text{if } ub_t = 1. \end{cases} \quad (7)$$

Assume a unifying broadcast is sent at time  $t$  and that a protocol without any pending broadcast from time  $t - 1$  wants to broadcast at time  $t$ . Due to our assumption that all broadcasts at time  $t$  are processed simultaneously, this broadcast will be sent with the unified broadcast and hence the protocols would have  $pending_t^p = 0$ . Using Equation 5 and 6, the savings in broadcasts of using UB can be defined as:

$$savings = \frac{\sum_p \sum_t broadcast_t^p - \sum_t ub_t}{\sum_p \sum_t broadcast_t^p} \quad (8)$$

Based on Table 1 we define the following four protocols in our model:  $CTP = \langle 64ms, 3600s, \theta^{CTP}, 0 \rangle$  with a varying reset period  $\theta^{CTP}$ ,  $FTSP = \langle 30s, 30s, 30s, 0 \rangle$ , and two Deluge protocols with varying suppression probability and different reset periods depending in the frequency of code updates as  $Deluge_{Passive} = \langle 2s, 60s, \infty, \rho^{Deluge} \rangle$  and  $Deluge_{Active} = \langle 2s, 60s, 10min, \rho^{Deluge} \rangle$ .

Figure 4 shows the broadcast savings of three applications using CTP, FTSP and optionally a Deluge version, based on Equation 8 with a maximum time  $t$  of 50000s which is five times as much as the highest CTP reset period tested. Figure 4(a) shows the broadcast savings without any Deluge suppressions,  $\rho^{Deluge} = 0$ , and an increasing CTP reset period  $\theta^{CTP}$ . The maximum savings possible using only CTP and FTSP are for a reset period around 160s, and the savings increase when adding either active or passive *Deluge* to the application. As hinted from our offline analysis in the last section, the maximum savings are hard to achieve in well connected networks where the CTP timer is rarely reset. Figure 4(b) shows the broadcast savings with a fixed



(a) Deluge without suppression and a varying CTP reset period  
 (b) CTP reset period set to 160s and a varying Deluge suppression probability.

**Figure 4: The maximum savings in broadcasts, according to our model, when varying the CTP reset period and the Deluge suppression .**

CTP reset period,  $\theta^{CTP} = 160s$ , and an increasing Deluge probability of suppression,  $\rho^{Deluge}$ . The figure shows that the maximum savings in active *Deluge*, where Deluge contributes to a significant amount of the control traffic, depend on the suppression probability. As the suppression probability increases from 0 to 50%, the savings increase, because Deluge broadcasts that are sent can be combined. After a certain point, where more Deluge broadcasts that could be combined are not sent, the savings start to decrease again. The same tendency, at a smaller scale, is seen when using passive *Deluge* where Deluge’s contribution to the overall control traffic is limited.

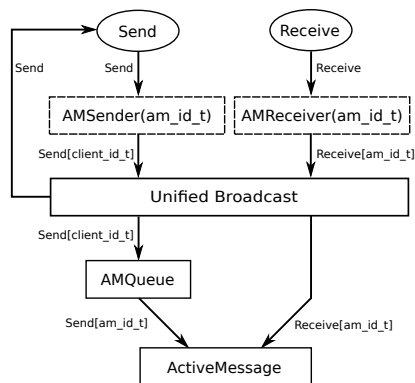
### 4.3 Guidelines

The previous subsections showed that the potential savings that UB achieves can be significant and that the best savings are achieved with a maximum delay of broadcasts. In general, broadcast savings are best in scenarios where the protocols equally contribute to the amount of control traffic. However, the savings do not only depend on the protocols used, but also on the actual deployment, where changes in the network state, such as link quality or topology changes, can impact broadcast savings as shown in Figure 4. As the exact characteristics of a deployment are rarely predictable or known a priori we recommend that the savings of using UB are experimentally verified, before deployment, when using low control overhead protocols such as CTP.

## 5. IMPLEMENTATION

UB is implemented in TinyOS together with its Active Message (AM) layer. The AM layer is a dispatch layer between the link and the network layer, that adds a one byte AM identifier to all outgoing packets which can be used on the receiving side to dispatch incoming packets to the appropriate upper layer protocol.

Following the TinyOS design principles, we had two alternative choices for how to implement UB: (1) it could be implemented as a separate service through which protocols could send broadcasts; or (2) it could be integrated into the AM layer. For (1), the UB service would use the AM layer to send special UB packets which would be handled accordingly at the receiving side. This approach would create a clear boundary between whether or not a protocol sends a packet through UB, but would require modification of all existing protocol implementations in order to enable them to use UB, and furthermore create the need for future protocol



**Figure 5: Overview of how UB is integrated into the TinyOS AM stack. Ovals represents interfaces to be used by upper level components and rectangles represent components. A dashed components represent a generic component from which multiple instances can be created. An interface can be parameterized which means that the providing component provides multiple instances of the same interface.**

developers to be aware of UB. Instead, we took the more transparent approach in which we integrated UB into the current AM stack so that all broadcasts by default, transparent to the actual protocols, are sent using UB. Our implementation takes about 2.6kB of ROM and 210B bytes of RAM.

Figure 5 shows how UB is integrated into the TinyOS AM stack. The top level components of the AM stack are the generic *AMSender* and generic *AMReceiver* which provide single instances of the *AMSend* and *Receive* interfaces, respectively. On the sending side without UB the *AMSender* sets some AM specific parameters and passes the packet on to the *AMQueue* which employ a first come first served queue on packets to be sent over the radio through the *ActiveMessage*. Note how the *AMQueue* provides the *Send* interface parameterized by the number of clients (number of *AMSender*’s) whereas the *ActiveMessage* only provides the *Send* interface parameterized by the AM identification. The *ActiveMessage* is a components provided by the specific radio and hence is the lowest layer of this hardware independent stack. The receiving side without UB is less complicated as the *AMReceiver* wires directly to the *Receive* interface with the specified AM identifier provided by the *ActiveMessage*.

UB sits as a layer directly below the *AMSender* and the *AMReceiver*. On the sending side broadcast packets are delayed until either a protocol with an already delayed packet in the queue sends another one or the aggregate size of delayed packets exceeds a single UB packet. As the analysis in Section 4.1 has shown, UB yields the maximum savings without a fixed maximum bound on delay, so our implementation does not apply this.

UB embeds broadcast packets into an UB packet by prepending the data of the packet by a one byte length field and the packet’s one byte AM identifier. Thus, each broadcast packet using UB incurs a two byte packet overhead. The embedded UB packet is sent from UB using a dedicated *AMSender* with a special UB AM identifier. The re-use of the *AMSender* for sending the UB packet enables it to be fairly scheduled for transmission together with other (e.g. uni-

	CF	CFD	CFDM	LF	LFD	LDFM
CTP	x	x	x			
LQI				x	x	x
FTSP	x	x	x	x	x	x
Deluge		x	x		x	x
Memento			x			x

**Table 2: The six different sensor network applications considered. A “x” means that the protocol is included in the application.**

cast) packets through the `AMQueue`. Furthermore, the use of a special UB AM identifier, instead of simply assuming all broadcasts are UB broadcasts, enable UB to send broadcast packets that are too big to be embedded as a non-embedded normal broadcast packet.

UB has to handle packets requiring time synchronization as a special case. The current best practice for packet time synchronization in TinyOS is to modify the event time after packet transmission has started. Thus, the event time needs to be placed at the end of the packet which might not be the case if we naively embed broadcast packets into a UB packet. Fortunately, TinyOS currently sends all packets requiring time synchronization with a special allocated time synchronization AM identifier. Thus, we can easily handle time synchronization packets by placing the timestamp field at the end of the UB packet, while keeping the packet content embedded as with any other packet. Similarly, on the receiving side, if a time synchronization packet is embedded, the event time is fetched from the end of the UB packet and concatenated with the packet content before being relayed to the upper layers.

## 6. EVALUATION

We evaluate our UB implementation in simulation and on real sensor node hardware by comparing sensor node applications with and without the use of UB. The evaluation is twofold: first we evaluate the savings achieved by using UB and then we evaluate how these savings affect the upper layer routing, time synchronization, dissemination, and failure detection protocols used.

### 6.1 Methodology

We evaluate our UB implementation with six different sensor network applications deployed on the eight 16 node TOSSIM networks introduced in Section 4.1 and on a 15-node local sensor node testbed. The local testbed is a 5x3 TMote Sky [3] grid network deployed on a flat surface with an inter-node distance of approximately 1 meter. Again, the collection root is placed in a corner and multi-hop paths to the root is ensured by setting the transmission power of the CC2420 radio to a minimum. All testbed experiments were done using software-initiated IEEE 802.15.4 acknowledgements for reliability, hardware address recognition enabled, and on channel 26 where we infrequently detected natural interference from neighboring sensor networks.

The six different sensor network applications consist of either CTP or LQI for data collection, FTSP for time synchronizations, Deluge for wireless network re-programming, and Memento for node failure detection. Section 2 provided an overview of the protocols and Table 2 an overview of the applications. The implementation of CTP, LQI, and FTSP

are taken from the latest TinyOS tree. As packet timestamping, and hence FTSP, is not supported in TOSSIM we used a stripped down version of FTSP leaving out the timestamping and global time calculations for the TOSSIM simulations. Similarly, Deluge is not supported (or needed) in TOSSIM so we only run its code consistency maintenance protocol (DRIP [19] in its TinyOS implementation) which handles all communication in Deluge when its in a consistent state. The code for Memento is not public available so we implemented a distributed version of its Variance-Bound failure detection mechanism based solely on neighborhood monitoring. Hence we neglect any efforts done by Memento to efficiently unicast state summaries to a root node and only focus on the broadcast related part in form of heartbeats and failure detection ability. The detection of failures are reported together with any other protocol statistics over the USB back-channel. As in Section 4, the above protocols are configured based on Table 1 leaving only the data generation period as a tunable parameter.

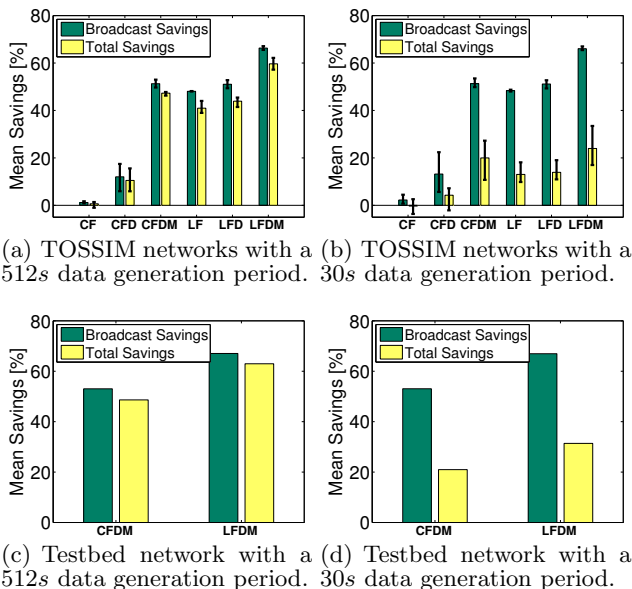
UB saves on broadcasts by combining broadcasts from multiple protocols into one. If the size of a broadcasts from a protocol equals the maximum MAC Service Data Unit (MSDU) of the link layer, UB will not be able to save on the broadcasts, and hence will send it as a normal broadcast. To exploit the full potential of UB we increased the TinyOS MSDU from the default 28 bytes to 60 bytes.

All results shown in the following subsections are averaged over 5 identical runs with the same application and data generation period for both the TOSSIM networks and the testbed.

### 6.2 Packet Transmission Savings

Figure 6 shows the broadcast and total savings in terms of packet transmissions for the six different sensor network applications with two different per node data generation periods deployed on the TOSSIM networks and on the testbed.

Figures 6(a) and 6(b) show the savings for the six application in the 8 TOSSIM networks with a per node data generation period of 512 and 30 seconds, respectively. The figures show how the savings increase with the number of protocols used, especially with CTP, where the broadcast savings increase from a neglectable  $\sim 1\%$  to more than 50% with both data generation periods. The error bars show the changes in savings over the different networks which is quite significant in the CFD application. Comparing the CFD broadcast savings from Figure 6(a) to the savings in our offline analysis of broadcast savings from Figure 3(a) with no fixed delay limit, we see a similar variation. As shown in Section 4.2, this can be attributed to how the network dynamics of the specific deployment affect CTP and Deluge broadcasts. Furthermore, the fact that the broadcast savings of the offline analysis, which does not consider how UB influences upper level protocols, and the results shown in Figure 6(a) are similar indicates that UB has minimal influence on upper level protocol performance with regard to broadcasts. A detailed discussion of this is the topic of the next subsections. As expected from our offline analysis the savings of using UB are greater with LQI than CTP. Of particular interest is the broadcast savings of the LF application, which shows broadcast savings close to 50% for only two protocols with both data generation periods. This approaches the theoretical limit of the broadcast savings possible with two protocols. In general the total savings with a per node data generation



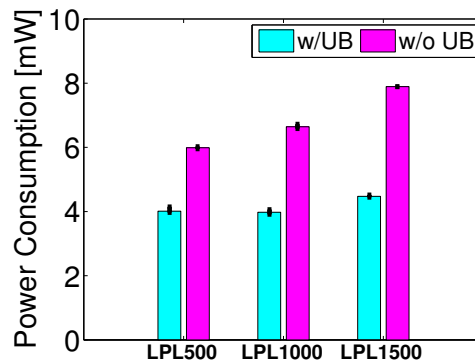
**Figure 6: Broadcast and total packet transmission savings in TOSSIM and in the testbed. TOSSIM graphs are averaged over 8 different networks, while we only tested one network topology in the testbed. The error bars in TOSSIM graphs show variation in savings across different networks. The graphs are averaged over 5 identical 6 hour runs for each application in each network.**

period of 512 seconds comes close to the broadcast savings whereas the total savings with a per node data generation period of 30 seconds is less due to increased amount of data traffic. From the figures we see that the total savings using LQI seems to vary more than the broadcast savings over the networks. A detailed analysis of this variation show that the total savings decrease with decreasing network connectivity, due to a decrease in delivery rate, which causes nodes to send more data packets.

Figures 6(c) and 6(d) show the savings of the CFDM and the LFDm application in the testbed with a per node data generation period of 512 and 30 seconds, respectively. Comparing Figure 6(c) to 6(a) and Figure 6(d) to 6(b) we see that the savings found in the testbed are similar to the savings found in TOSSIM. In fact, the savings achieved by using UB seems to be slightly better when running the protocols on real sensor node hardware, which we attribute to topology differences between the simulation and testbed networks. Note that error bars showing variations over the different runs are not shown in Figure 6(c) and 6(d) for consistency with Figure 6(a) and 6(b) where the error bars show variations over different networks. However, the error bars would, similarly to the ones shown in Figure 7, not add much information to the figure.

### 6.3 Energy Savings

The UB savings in packet transmissions do not directly translate to total savings in energy, which also depends on the time spent in reception and idle states. As a case study, to see how the use of UB can affect the total energy consumption, we tested it with the widely used TinyOS CC2420 Low



**Figure 7: Power consumption of the CC2420 radio with and without UB in the testbed with LPL intervals of 500ms, 1000ms, and 1500ms. The graphs are averaged over 5 identical 6 hour runs with LPL where the error bars show the variation of the runs.**

Power Listening (LPL) layer (a BoXMAC2 implementation [16]) in our testbed. We used different LPL wake up intervals and a delay to turn off the radio after packet reception of 10ms (which is used as a mechanism in TinyOS to allow a transmitter to send a number of packets without a receiver turning off its radio). We limited this test to the CFDM application and ran 5 identical 6 hour runs for each LPL interval. We chose CFDM in favor of LFDm which have better savings due to CTP being more widely-used than LQI.

To estimate the energy consumption of the sensor nodes we used software-based on-line energy estimation [7] which multiplies the on time of peripherals with their current draw and voltage to get a measure of their energy consumption. We implemented our own variant in TinyOS which tracks not only the on and off states of the peripherals but also their internal states, which can have varying current draws. Our implementation uses a 32KHz timer to accurately measure the time each peripheral spend in a state and add this to an accumulated energy consumption state variable at every state change. In our experiments we neglect the energy consumed by the MSP430 micro-controller due to it being orders of magnitude lower than the energy consumed by the CC2420 radio. The TinyOS CC2420 drivers always keep the radio off, in receive mode, or in transmit mode. When the radio is off, it does not consume any energy, so to measure the radios energy consumption we only had to modified the drivers to track whenever it was in receive or transmit mode. We multiplied the times spent in receive and transmit mode by the current consumptions of 19.7mA and 17.4mA, respectively, taken from the CC2420 data-sheet [1].

Figure 7 shows the estimated power consumption of the CC2420 radio, with and without the use of UB, in our LPL experiments. In general the figure confirms that the significant total packet savings achieved with UB also translates to significant energy savings. However, due to the fact that the savings in packet transmissions will only cause similar savings in packet receptions and not in idle listening, the energy savings will always be less.

Idle listening is only affected by savings in packet transmissions to the extent that one will not do idle listening while sending or receiving. The ratio of this happening, compared to the actual number of idle listens, is low and hence the energy consumed idle listening is majorly determined by

		CTP500		LQI500		CTP30		LQI30	
		w/ UB	w/o UB	w/ UB	w/o UB	w/ UB	w/o UB	w/ UB	w/o UB
Delivery [%]	min	99.83	99.83	37.20	37.15	99.76	99.87	39.10	39.76
	mean	99.93	99.94	58.19	58.89	99.92	99.94	58.92	60.53
	max	100.00	100.00	98.43	99.45	99.99	99.99	99.43	99.70
Data Cost [pkt]	min	1.64	1.70	3.00	2.92	1.88	1.65	2.98	2.94
	mean	3.56	3.61	12.88	13.12	3.76	3.71	11.92	11.76
	max	5.73	6.14	28.59	28.46	6.28	6.18	20.31	22.29
Control Cost [pkt]	min	<b>0.70</b>	<b>0.63</b>	16.64	16.48	<b>0.03</b>	<b>0.02</b>	0.94	0.94
	mean	<b>0.77</b>	<b>0.72</b>	32.45	33.03	<b>0.08</b>	<b>0.07</b>	1.83	1.84
	max	<b>0.89</b>	<b>0.83</b>	50.32	51.01	<b>0.18</b>	<b>0.17</b>	2.61	2.86
Churn [chg]	min	1.24	1.16	64.80	70.63	1.20	1.21	67.97	71.97
	mean	1.59	1.64	156.61	157.73	6.21	6.14	162.32	163.19
	max	1.95	1.89	224.50	222.53	15.11	16.04	233.47	223.69
Path Length [hops]	min	1.29	1.36	1.08	1.08	1.53	1.36	1.04	1.02
	mean	2.05	2.09	1.70	1.68	2.37	2.34	1.65	1.62
	max	2.92	3.06	2.50	2.44	3.26	3.23	2.47	2.44

**Table 3: TOSSIM routing performance with and without UB for CTP and LQI using a data generation period of 512 and 30 seconds. The results are averaged over 5 identical 6 hour runs for each application in each network.**

	CTP500		LQI500		CTP30		LQI30	
	w/ UB	w/o UB	w/ UB	w/o UB	w/ UB	w/o UB	w/ UB	w/o UB
Delivery [%]	99.74	99.92	92.02	95.71	99.97	99.97	96.65	93.89
Data Cost [pkt]	3.93	4.00	4.50	4.13	3.88	3.94	4.01	4.29
Control Cost [pkt]	<b>0.48</b>	<b>0.53</b>	19.27	17.83	<b>0.03</b>	<b>0.03</b>	0.98	1.03
Churn [chg]	0.72	0.85	10.54	6.89	0.58	0.72	10.77	8.42
Path Length [hops]	3.60	3.64	3.18	3.32	3.61	3.66	3.36	3.25

**Table 4: Testbed routing performance with and without UB for CTP and LQI using a data generation period of one packet per node each 30s and 500s. The results are averaged over 5 identical 6 hour runs for each application.**

the LPL interval and the time it takes to sample the channel. Thus, the energy consumed in idle listening decreases with an increasing LPL interval, which is why the energy savings in Figure 7 increase from 33%, to 40% and 43% when increasing the LPL interval from 500ms to 1000ms and 1500ms, respectively.

Similarly, one could increase the energy savings achieved by using UB by decreasing the time it takes to sample the channel for activity. Using the default TinyOS CC2420 stack we empirically measured this time to be 14ms which is set this high (almost an order of magnitude higher than the time it takes to transmit an IEEE 802.15.4 packet on the channel) due to the stack using an acknowledgement timeout of almost 8ms. If the CC2420 stack was optimized and this time decreased, the energy savings of UB will increase. However, it can never be more than the total packet savings which was around 50% for the CFDM application in the testbed.

Of special interest in Figure 7 is the trends of the total power consumptions when increasing the LPL interval. Without UB the total power consumption increases from 5.99mW, to 6.64mW and 7.89mW when increasing the LPL interval due to the increased cost of transmissions being more than the increased idle savings. Whereas with UB the total power consumption is almost stable around 4mW (4.01mW and 3.98mW) when increasing the LPL interval from 500ms to 1000ms while it increases again to 4.47mW when increasing the LPL interval further to 1500ms. We

attribute the slight decrease in (or unchanged) power consumption to the savings in idle energy being larger than (or the same as) the extra cost of transmission when increasing the LPL interval, and vice versa for the increase in power consumption. Note that this trade-off between idle and transmission energy is a common trade-off, not caused by the use of UB, when dealing with LPL—it just happens to be that with UB in our testbed the equilibrium is in the chosen LPL range.

## 6.4 Routing

The previous section showed the savings achieved by allowing UB to delay and combine broadcasts from protocols in a number of scenarios. However, the delay of broadcasts from a routing protocol might affect its functioning and hence decrease its ability to correctly estimate link qualities or eliminate routing loops. In this section we compare the routing performance of CTP and LQI with and without the use of UB.

We focus on five performance metrics: delivery, data cost, control cost, churn, and path length. We compute the network delivery rate as the average of individual node delivery rates. Individual delivery rates are simply measured as the portion of packets for each node that are correctly received at the root of the routing tree. We also measure the data and control cost as the average of the per node total number of data/control packets sent over the number of packets originated from the node. Churn is the average of the per

	w/ UB	w/o UB
Error Mean	1.2	1.3
Error Std. Dev.	0.3	0.4

**Table 5: Mean absolute deviation of FTSP global time in 32kHz clock units. Mean and standard deviation is shown for all recorded reference broadcasts.**

node number of parent changes throughout the experiment and path length is the total number of successfully sent data packets in the network over the number of originated data packets in the network.

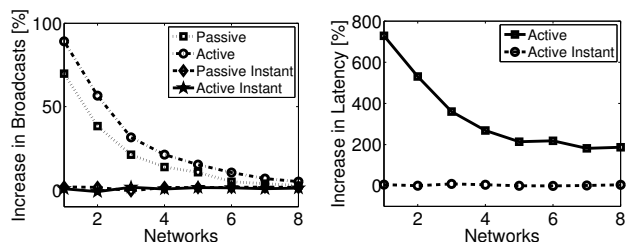
Table 3 compares the routing performance of the CFDM and the LFD application with and without the use of UB using a per node data generation period of 512 and 30 seconds in our 8 TOSSIM networks. The table compares the minimum, mean, and maximum of all five performance metrics for the two application in the 8 networks. In general, the protocols perform similar with and without the use of UB. Similarly, Table 4 compares the routing performance in the testbed of the two applications with and without the use of UB using a per node data generation period of 512 and 30 seconds. Of special interest in these tables are how the use of UB affects the control cost of the routing protocol. LQI is a periodic beaconing protocol, so UB was not expected to affect the control cost of this protocol. But, as discussed earlier, CTP makes use of adaptive broadcasts which might be reset more times if the UB delays e.g. causes routing loops not handled in a timely manner. The TOSSIM evaluation from Table 3 seems to indicate that the control cost is slightly higher when using UB whereas the testbed evaluation in Table 4 indicate the opposite. We believe that these small fluctuations in control cost are due to the ability of UB to adapt its delays to changing periods—UB never delays a packet longer than the last period from the same protocol.

## 6.5 Time Synchronization

FTSP is a fixed period protocol and with the use of UB the average number of broadcast sent throughout a networks lifetime is not going to change. However, the use of UB might delay an FTSP packet up to one period and similarly send two FTSP packet right after each other. In this section we explore how such randomization to the FTSP period affects accuracy of the time synchronization.

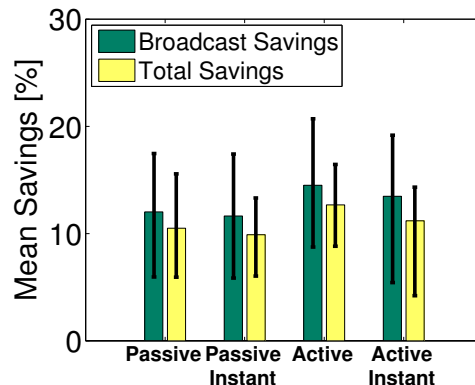
To test the accuracy of FTSP we made use of a reference broadcaster during our testbed experiments which periodically, every 5 minutes, sends a special reference broadcast on the wireless channel. On reception of such a broadcast the nodes log their estimated global time together with the reference’s sequence number and send this, together with the other protocol statistics, over the available USB back-channel.

Table 5 shows the mean and the standard deviation of the global time mean absolute deviation (MAD) for all reference broadcast recorded in five 6 hour runs with the CFDM application. In order to let FTSP synchronize we consider the first 20 minutes of each experiment an initialization period and do not consider reference broadcasts logged in these intervals. The table shows that the randomization caused by UB slightly improves the global time accuracy of the FTSP protocol. We consider the difference shown in Table 5 to be negligible with respect to the clock resolution.



(a) Deluge Broadcasts.

(b) Deluge Latencies.



(c) Dissemination Savings.

**Figure 8: Dissemination performance and UB savings of the CFD application with passive and active Deluge over the 8 TOSSIM networks with and without delays. The graphs are averaged over five identical 6 hour runs.**

## 6.6 Dissemination

Deluge uses a Trickle timer with suppression for nodes to advertise their code version in order to insure code consistency throughout the network. With the use of UB these code advertisement can be delayed, and hence a node might receive and old version from a node that is already up to date. Furthermore, the use of suppression in Deluge can be affected by the delays incurred by UB. In this section we explore how the use of UB affects the working of Deluge which is the only protocol considered that makes use of suppression.

In order to focus on the Deluge performance, we limit ourselves to the CFD application (the CFDM application gives better savings but Deluge’s influence on savings is more clear with the CFD application) with a per node data generation period of 512s. We consider two versions of Deluge: a passive and an active version, which represent the two possible states of the Deluge protocol. The majority of the time Deluge will be in a passive state, where the code is functional and hence does not need updates. However, occasionally Deluge will be used to re-program the network either due to feature updates or for debugging purposes. We let the passive version be Deluge without any code updates and the active version be Deluge where the networks is re-programmed every 10 minutes.

Figure 8(a) first shows how the use of UB with delays increases the number Deluge broadcasts being sent, both with passive and active Deluge (the non-instant lines). A comparison of these results to Deluge running without suppression show that the increase in number of broadcast being sent

with passive Deluge is entirely caused by UB interfering with the suppression mechanism of Deluge. This also explains why the increase in broadcasts decreases with a decreasing network connectivity where suppression is less likely to happen. Similarly, the use of UB interferes with the suppression mechanism of active Deluge, and contributes to the majority of the increase in broadcast being sent in this case. However, comparing these results to an active Deluge running without suppression we found that the lack of suppression is not the sole cause of the increase in broadcasts. We attribute the remaining increase in broadcasts with active Deluge to cases where a sensor node broadcasts and old version, due to the UB delay, even though it is already up to date. Thus, causing the neighbors to think that it is outdated.

The use of UB does not only increase the number of broadcast but also the code inconsistency detection time which we refer to as Deluge latency. Figure 8(b) first show how the latency in detection for the Deluge active (the non-instant line) goes from a striking 700% for well connected networks to about 200% for medium and bad connected networks. As with the number of broadcast sent by Deluge, we compared these results to Deluge running without suppression, which showed that a fraction of the delay can be attributed to limited suppression, but most of it is simply due to the delay of Deluge broadcasts.

The increase in latency with the use of UB can be unacceptable, so we extended our UB implementation with the ability to disable delays for certain packets. We do this per AM client and hence let each `AMSender` provide a `UnifiedBroadcast` interface with a command for enabling and disabling delays for packets send through that specific `AMSender`. Note that disabling delays for a protocol does mean that the protocols packets are sent without UB and hence prevent it from being combined with other packets. Instead, it merely forces the UB layer to send the current accumulated UB packet instantly after processing such a packet.

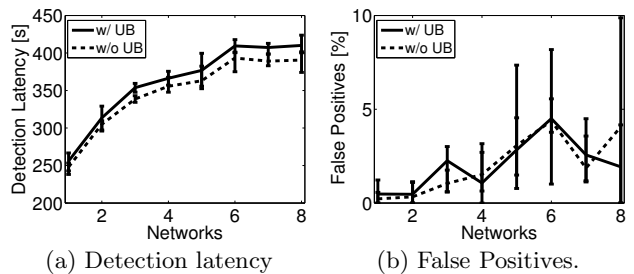
Figures 8(a) and 8(b) show how the increase in broadcast sent by Deluge and the increase in detection latency is 0% with Deluge broadcast sent instantly. Thus, Deluge performs no different with the use of UB in this case.

Figure 8(c) shows how the savings achieved by using UB is affected by disabling delays for Deluge packets. Similar to the figures and discussion in Section 6.2, the error bars show the changes in savings over the different networks which is quite significant in the used CFD application. Due to the fact that Deluge packet without delays can still be combined with other broadcasts, the figure shows that the average savings with delays for Deluge packets disabled is only slightly less than the savings with delays for Deluge packets enabled. Also, we see that the savings achieved when using UB with active Deluge is higher than when using UB with passive Deluge due to the increase in Deluge broadcasts.

## 6.7 Failure Detection

Memento detects node failures based on the number of heartbeat messages it hears from neighboring nodes in a sweep period. If these heartbeats are delayed the number of heartbeats per sweep period can vary independent of the link quality and hence influence the Memento failure detection. In this section we explore exactly how by running the Memento Variance-Bound failure detection with a target false positive rate of 1% with and without the use of UB.

We limited these experiments to the CFDM application



**Figure 9: Detection latency and false positive rate when running Memento's Variance-Bound failure detection mechanism with and without UB. The graphs are averaged over five identical 6 hour runs and the error bars show the absolute variation over these runs.**

with a per node data generation period of 512 seconds and run five 6 hour runs in the 8 TOSSIM networks with forced node failures. We let the failure schedule be identical for all runs and turn off a node for 10 minutes every 20th minute throughout the simulation.

Figure 9(a) shows the average node failure detection latency of the Variance-Bound failure detection mechanism with and without the use of UB. The figure shows that the detection latency when using UB is slightly higher than without which is due to the variance in number of heartbeats per sweep period caused by the UB delays. The Variance-Bound failure detection mechanism cannot tell the difference between this variance and the variance in heartbeats caused by changing link qualities, and hence accounts for this variance in its estimate. Thus, the number of missed heartbeats before a failure is concluded increases and thereby the detection latency.

Figure 9(b) shows the average number of false positive failures reported by the Variance-Bound failure detection mechanism with and without UB. Even though the variance over the runs is quite large, which is due to the limited number of samples available in a 6 hour run where a node only fails every 20 minutes, the figure shows similar results with and without UB. Considering that the target false positive rate, taken as input to the Variance-Bound failure detection mechanism, was 1%, the technique performs poorly in the low quality networks. We believe this is due to even less recorded detection in these networks, but leave a detailed exploration of the cause to future work as it is not the topic of this paper.

## 7. CONCLUSION

We presented Unified Broadcast as a modular solution for seamlessly combining broadcasts from many higher layer sensor network protocols. We analyzed the benefits of UB through extensive offline analysis, which motivated the design and implementation of UB in TinyOS. We then explored UB in TOSSIM simulations and testbed experiments for more comprehensive results. Our main observation is that the benefits of UB increase with the number of concurrent broadcast protocols. Our analysis and experiments explored two routing protocols, CTP and LQI, running alongside time synchronization (FTSP), dissemination (Deluge), and health monitoring (Memento) protocols. We have shown both an-

alytically and empirically that the total packet savings of UB depends on the number of upper layer protocols, and range from close to 0% for a certain two protocol case up to 60% for 4 protocols running within the same application. Our performance evaluation has also shown that a total packet savings of around 50% with UB translate to about 43% energy savings when using LPL with a wakeup interval of 1500ms. The reduced number of broadcasts with UB means that nodes spend more time idle listening and hence the energy savings can never be as much as the packet savings. However, through our energy analysis with different LPL intervals, we have shown that higher LPL intervals achieve higher energy savings because of the reduced impact of idle listening on the overall node energy profile.

One of the key questions for UB is its impact on the correct operation of the upper layer protocols. We have investigated this question for each of the protocols used. For protocols that use a static broadcast period, such as LQI, FTSP, and Memento, the use of UB did not exhibit any significant decrease in the protocol performance metrics. Dynamic period broadcast protocols had more distinctions when running with and without UB. More specifically, the CTP simulation experiments with UB yielded a 5-7% increase in control traffic, while the testbed experiments for UB showed a decrease of up to 8% for the same metric. We attribute this difference to topological differences between the simulation and testbed. In the case of Deluge, the basic implementation of UB yielded significant increases in latency and broadcasts, which prompted us to bend the UB design goal of transparency to add an optional feature to disable delays for certain packets. We show that UB still achieves considerable savings even if some protocols force immediate transmission of their broadcast packets.

UB is a layer 2.5 protocol which can fit into existing architectures, such as SP and Chameleon, that aim at providing a layer 2.5 abstraction for all upper layer services and protocols. An interesting direction for future research is to explore how UB can be integrated as a component into these architectures.

## 8. REFERENCES

- [1] CC2420: 2.4 GHz ieee 802.15.4/ZigBee RF transceiver. <http://www.ti.com/lit/gpn/cc2420>.
- [2] Multihoplqi. <http://www.tinyos.net/tinyos-1.x/tos/lib/MultiHopLQI>.
- [3] Tmote sky low power wireless sensor module. <http://sentilla.com/files/pdf/eol/tmote-sky-datasheet.pdf>.
- [4] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin. Temporal properties of low power wireless links: modeling and implications on multi-hop routing. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 414–425, 2005.
- [5] A. Dunkels, L. Mottola, N. Tsiftes, F. Österlind, J. Eriksson, and N. Finne. The Announcement Layer: Beacon Coordination for the Sensornet Stack. In *Wireless Sensor Networks*, Lecture Notes in Computer Science. 2011.
- [6] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 335–349, 2007.
- [7] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32, 2007.
- [8] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A modular network layer for sensorsets. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 249–262, 2006.
- [9] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. Technical report, Stanford, 2007.
- [10] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys '09: Proceedings of the 7th ACM conference on Embedded network sensor systems*, 2009.
- [11] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, 2004.
- [12] R. Jurdak. *Wireless Ad Hoc and Sensor Networks: A Cross-Layer Design Perspective (Signals and Communication Technology)*. Springer-Verlag New York, Inc., 2007.
- [13] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 351–365, 2007.
- [14] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, 2004.
- [15] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, 2004.
- [16] D. Moss and P. Levis. Box-macs: Exploiting physical and link layer boundaries in low-power networking. Technical report, Stanford, 2008.
- [17] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, 2005.
- [18] S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, volume 2, pages 575–584, September 2006.
- [19] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Wireless Sensor Networks, 2005. Proceedings of the Second European Workshop on*, pages 121–132, February 2005.