The Announcement Layer: Beacon Coordination for the Sensornet Stack

Adam Dunkels, Luca Mottola, Nicolas Tsiftes,
Fredrik Österlind, Joakim Eriksson, and Niclas Finne

Swedish Institute of Computer Science
{adam,luca,nvt,fros,joakime,nfi}@sics.se

Abstract. Sensornet protocols periodically broadcast beacons for neighborhood information advertisement, but beacon transmissions are costly when power-saving radio duty cycling mechanisms are used. We show that piggybacking multiple beacons in a single transmission significantly reduces transmission costs and argue that this shows the need for a new layer in the sensornet stack—an announcement layer—that coordinates beacons across upper layer protocols. An announcement layer piggybacks beacons and coordinates their transmission so that the total number of transmissions is reduced. With an announcement layer, new or mobile nodes can quickly gather announcement information from all neighbors and all protocols by issuing an announcement pull operation. Likewise, protocols can quickly disseminate new announcement information to all neighbors by issuing an announcement push operation. We have implemented an announcement layer in the Contiki operating system and three data collection and dissemination protocols on top of the announcement layer. We show that beacon coordination both improves protocol performance and reduces power consumption.

1 Introduction

Sensor network protocols use periodic beacons to advertise information to neighbors. Examples include routing cost gradients in data collection protocols [11,23,24], version or sequence numbers in data dissemination protocols [12,15,16], and presence information in neighbor discovery protocols [8,14]. Beacons are transmitted both periodically and when protocols detect potential inconsistencies. For example, a node in a collection protocol that detects a loop repairs the network by asking its neighbors for the latest routing cost gradient [11], and a node in a dissemination protocol that has an older version than its neighbors achieves consistency by asking its neighbors for the latest version [15].

Beacons are transmitted as broadcasts so that they reach all nodes in the neighborhood of the transmitter. Broadcast are, however, costly in terms of power since low-power networks duty cycle their radios. Many protocols therefore attempt to reduce the amount of beacons they transmit. For example, the CTP data collection protocol uses adaptive beaconing [11] and the Trickle and the

P.J. Marrón and K. Whitehouse (Eds.): EWSN 2011, LNCS 6567, pp. 211–226, 2011.

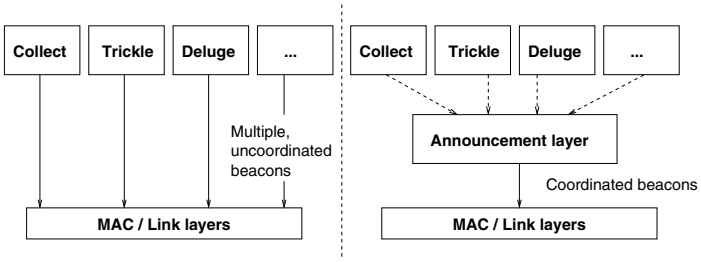


Fig. 1. The announcement layer coordinates beacons from multiple protocols. This allows the number of beacon transmissions to be reduced and announcement operations to be coordinated across protocols.

RPL protocols uses beacon suppression [15,23]. These solutions only work for one individual protocol, however. When multiple protocols are used concurrently, each protocol will transmit their beacons independently of each other, increasing the power consumption.

We argue that there are significant power savings to be made through coordinating and piggybacking beacons from multiple protocols. We present the announcement layer, a beacon coordination layer for the sensornet stack that transmission of periodic beacons from multiple protocols, piggyback multiple beacons into each transmission to reduce the total amount of beacon transmissions, and provide operations for pushing announcements to the neighborhood and for requesting announcements from neighbors. Figure 1 shows how the announcement layer fit into the network stack.

The announcement layer defines two operations: *push* and *pull*. Push quickly transmit an announcement to the neighborhood and pull requests announcements from all neighbors. The push operation is used e.g. when a collection protocol finds a better route and the pull operation is used e.g. when a collection protocol node detects a routing loop and the latest routing gradients are needed.

We argue that an announcement layer provide at least three benefits. *Reduction of bandwidth usage and network congestion* since multiple beacons are collected in a single broadcast transmission (Section 5.1). *Reduction of power consumption* since the marginal cost of sending larger packets is low (Section 5). *Easier inter-protocol coordination* since announcements from multiple concurrent protocols are collected at a single point in the network stack, push and pull operations can be made across protocols (Section 5.2).

We make our case as follows. We demonstrate that beacon transmissions are costly that multiple transmissions even more so (Section 2). This motivates the need for an announcement layer (Section 3). We have implemented an announcement layer in Contiki (Section 4) and demonstrate that the use of an announcement layer reduces the number of beacon transmissions and the power consumption in a series of simulations and testbed experiments (Section 5) and discuss how an announcement layer differs from existing approaches (Section 6).

2 Motivation: Beacon Transmissions Are Costly

The background to the announcements programming abstraction stems from the periodic broadcast beacons used by sensor network protocols to do one-hop neighborhood information advertisement, and the observation that radio duty cycling makes broadcast expensive.

2.1 Sensornet Protocols Use Beacons

Sensor network protocols use periodic beacon transmissions to advertise information to the one-hop neighborhood. Examples include route metrics in data collection protocols [11] and version numbers in data dissemination protocols [12].

Information advertisement within the physical neighborhood of sensor nodes may also serve functionality reaching up to the application level, e.g., to coordinate sensors and actuators in a control application [5]. Programming systems for such application-level information sharing exist, such as Hood [22] and TeenyLime [4].

Beacons are typically transmitted periodically, but the period often changes over time. Many protocols exponentially increase their beacon rate when the information in the beacons has been transmitted several times and is no longer new. Examples include the Trickle single-packet data dissemination protocol [15], the multi-packet data dissemination protocol Deluge [12], the CTP data collection protocol [11], and the RPL low-power IPv6 routing protocol [23].

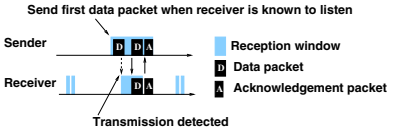
2.2 Duty Cycling Makes Beacon Transmissions Costly

Beacons need to reach all nodes in the neighborhood of the transmitting node. Beacons are therefore sent as broadcast messages, but since radio duty cycling must be used to maintain a low power consumption, broadcast messages become comparatively expensive in terms of power consumption.

To send a broadcast transmission, the sender must make sure that all its neighbors are awake to receive the broadcast transmission. Ensuring that all neighbors are awake to receive the transmission can be done in two ways: either by having all nodes agree on scheduled rendez-vous when all nodes are simultaneously awake, or by having the sender explicitly wake up all its neighbors. Scheduled rendez-vous are costly since nodes must wake up for every rendez-vous, even if no data is to be transmitted. Explicit wake-ups are expensive since the sender must make sure that all nodes receive the message, thus typically needs to transmit the message multiple times. In contrast, for a unicast transmission, it is enough that only one node—the receiver—is awake to receive the message. Thus unicast transmissions are fundamentally less expensive.

We perform a set of experiments to quantify the power cost of broadcast transmissions. We use Contiki 2.5 with the ContikiMAC duty cycling protocol, the default duty cycling protocol in Contiki 2.5. ContikiMAC is a low-power-listening MAC protocol that builds on mechanisms from many existing state-of-the-art duty cycling protocols [2,10,17,19] but adds a very power-efficient channel

(a) Low-power listening with ContikiMAC.



(b) Transmission phase-lock.

Fig. 2. After a successful transmission, the sender has learned the channel sampling phase of the receiver, and subsequently needs to send only two transmissions

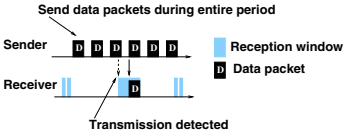


Fig. 3. A broadcast transmission must wake up all neighbors. The sender therefore extends the packet train for a full channel sampling period.

sampling mechanism. From B-MAC [19], ContikiMAC lends the basic idea of low-power listening. From X-MAC [2], ContikiMAC uses the idea of a packetized preamble. From WiseMAC [10], ContikiMAC uses the phase-lock mechanism, which we describe below. From BoX-MAC [17], ContikiMAC uses the idea of using the data packet as the wake-up signal.

Figure 2 shows the basic operation of ContikiMAC. Nodes wake up periodically to sample the radio medium for transmissions. This is performed in a power-efficient way: a node turns the radio on for only 192 microseconds to measure the received signal strength. If this indicates a transmission from a neighbor, the node keeps the radio on. To avoid missing transmissions, the node samples the radio medium twice within 0.5 ms. A sender triggers a transmission by sending a train of data packets, until one packet finds the receiver's radio on. Upon receiving a packet, the receiver answers with a link-layer acknowledgment and the sender stops transmitting the packet train. In ContikiMAC, unicast transmissions are power-efficient because senders phase-lock to the wake-up interval of its neighbors [10]. A sender synchronizes to the wake-up phase after the successful transmission, as shown in Fig. 2. For broadcasts, the sender needs to send its packet train for a full channel sampling period to ensure that all neighbors have heard the transmission, as shown in Fig. 3.

To quantify the relative cost of broadcast and unicast, we perform an experiment using two TMote Sky motes running Contiki and ContikiMAC. We use a channel sampling rate of 16 Hz. We run two experiments, one where we send broadcast traffic and one where we send unicast traffic, and vary the send rate. We measure the radio duty cycle using Contiki's built-in software-based power profiler [6]. Figure 4 shows the results. We observe that the cost of broadcast is significantly higher than that of unicast, and that the marginal increase in power

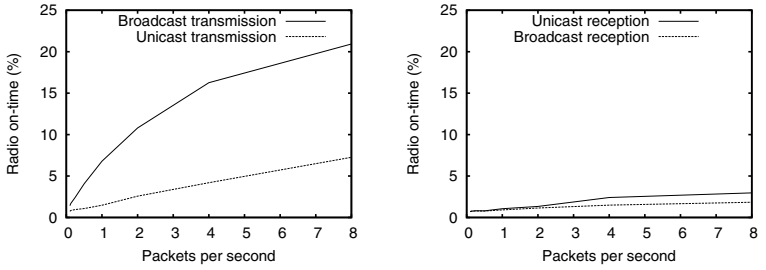


Fig. 4. The power consumption of broadcast is significantly higher than for unicast because broadcast transmissions (Fig. 3) are longer than unicast transmissions (Fig. 2)

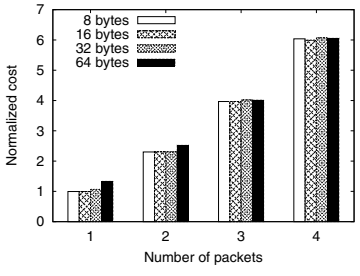


Fig. 5. Multiple, small broadcast transmissions are significantly more costly than a single broadcast transmission, for the same amount of data

consumption with increasing send rate is higher for broadcast. Therefore, there is much larger room for improvement in optimizing broadcast transmissions rather than unicast transmissions.

2.3 Multiple Transmissions Are More Costly

We perform another experiment, using the setup as above, where we transmit a fixed amount of data split into one, two, three, or four broadcast transmissions. The total amount of data is the same in all four cases, and we vary the amount of data across experiments. The purpose is to study the power consumption of multiple transmissions versus that of a single transmission, containing the same amount of data.

The result is shown in Fig. 5 and shows that multiple transmissions are significantly more costly than a single transmission, with the same amount of data. Also, the marginal cost of transmitting additional bytes in a single broadcast is significantly lower than the cost of transmitting the data as multiple transmissions. This demonstrates that there is a strong incentive to reduce the number of broadcast transmissions by collecting multiple beacons into a single, larger packet.

3 The Announcement Layer

The announcement layer provides beacon coordination for upper layer protocols. Protocols register announcements with the announcement layer and the announcement layer takes care of the periodic transmission of the announcements. An announcement is the information that a protocol would otherwise periodically transmit as beacons.

The announcement layer piggybacks announcements from multiple protocols in each beacon transmission and coordinates the transmissions so that the total amount of transmissions is reduced. Since the information sent in each announcement is typically small [11,15,16], several announcements often fit in a single beacon.

In addition to beacon coordination, the announcement layer provide a small but powerful set of operations that give protocols the ability to push announcements to neighbors and to pull announcements from neighbors.

An announcement is a key-value pair. The key is an integer that uniquely identifies the announcement. The value is a data array. The semantics of the value in an announcement is application-specific and opaque to the announcement mechanism.

Each announcement has a minimum rate for its periodic transmissions. The rate is set by the protocol that registered the announcement, and can be different for different announcements. From the minimum rates set for each announcement, the announcement layer computes a schedule that ensures that one and only one beacon is transmitted for every beacon interval. Since multiple announcements are consolidated into each beacon transmission, it is enough to send one beacon for each interval, thus reducing the total amount of beacon transmissions. This also means that an announcement may be transmitted more often than its minimum rate, which is allowed by the semantics of the announcements layer because protocols specify only the minimum transmission rate, not the maximum rate.

Each announcement has a scope that is either node scope or network scope. Node-scope announcements have a value that is bound to the node, whereas network-scope announcements have a value that is shared across the network. An example of a node-scope announcement is a hops-to-sink metric in a data collection protocol, which is specific to the node that registered the announcement. An example of a network-scope announcement is a global version number in a data dissemination protocol, which is the same for all nodes in the network.

3.1 Beacon Coordination

The announcement layer coordinates beacon transmissions for all registered announcements. The values from multiple beacons is collected into a single broadcast transmission. The beacon intervals from all protocols are coordinated to reduce the total amount of beacon transmissions.

Each protocol registers its maximal beacon interval with the announcement layer. The announcement layer ensures that at least one beacon is transmitted

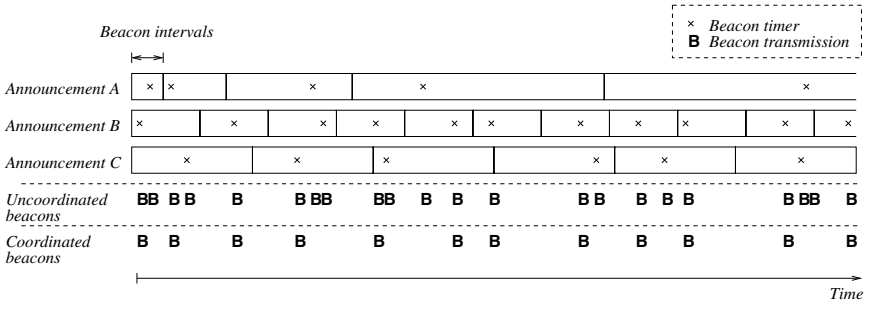


Fig. 6. Beacon coordination: Protocols A, B, and C have registered announcements A, B, and C. Data from all announcements are consolidated into each beacon transmission. With beacon coordination, only one beacon per announcement interval is transmitted. In this example, beacon coordination reduces the number of beacon transmissions from 22 to 12.

within this time interval. If two or more protocols need to transmit a beacon within a given time interval, beacon coordination will see to it that only the first beacon is sent. Since each beacon contains all announcements, the first transmission is enough to ensure that both announcements are transmitted within their respective intervals.

The beacon coordination concept is illustrated in Fig. 6. In the illustration, three protocols have registered three announcements. Each announcement has a different beacon interval. A beacon is to be sent randomly within each interval. Without beacon coordination and beacon consolidation, each protocol would send their own beacon messages without coordinating with the other protocols and the system in Fig. 6 would transmit 22 beacons. By contrast, with beacon coordination and beacon consolidation, the system sends only 12 beacons.

The beacon coordination algorithm is simple. For each new announcement, a timer fires randomly within each interval. When the timer fires, it checks if a beacon has already been transmitted within its interval. Unless the protocol has updated the value of its announcement since the last beacon transmission, there is no need to send a new beacon and the transmission is consequently cancelled.

3.2 Announcement Operations

The announcement layer defines two primary operations, *push* and *pull*. In addition, the announcements layer provides functions for registering announcement, to set the value of an announcement, and to set the minimum rate for the periodic transmission of announcements. Figure 7 shows the operations and functions.

Protocols that use announcements first register the announcement with the *register* function. The registration is a simple procedure that binds a key to the announcement and sets its scope. After registering the announcement, the protocol will begin receiving notifications when neighbors transmit their announcements. This is handled via a callback function that is given as an argument to

`push(key)`	Pushes an announcement to neighbors.
`pull(key)`	Pulls an announcement from neighbors.
`register(key, scope, callback)`	Registers an announcement and set its scope.
`setValue(key, value)`	Sets the value of an announcement.
`setMinRate(key, rate)`	Sets the min periodic transmission rate.

Fig. 7. Announcement layer operations

the *register* function. The protocol gives the announcement its value with the *setValue* function. The minimum transmission rate of the announcement is set with the *setMinRate* function.

The *push* and *pull* operations are used for pushing announcements to the neighborhood and to request announcements from neighbors, respectively. A *push* causes the transmission of an announcement to all neighbors. Protocols can use this operation to send new information to neighbors quickly. For example, a node in a data collection protocol that learns a significantly better route may want to quickly let its neighbors know of this new route. The node would set a new value for its announcement with the *setValue* function, and then call the *push* operation to push the changed routing metric to its neighbors.

The *pull* operation requests announcements from neighbors. The operation causes one or more neighbors to send their announcements to the node that issued the pull. This operation is used by protocols that are able to discover when a node needs information from its neighbors. For example, when a mobile node moves into a new environment it needs to gather information about its network environment, such as routing metrics. It then issues a *pull*, which causes its neighbors to send their announcements to the requesting node.

Although both the push and pull operations are defined to operate only on a single announcement, the beacon consolidation will collect all a node's announcements the beacon transmission. This means that a single push operation will push all announcements to neighbors. Likewise, a single pull operation will pull all announcements from neighbors.

The push and pull operations are based on the observed behavior of existing protocols:

Data collection with CTP. The CTP data collection protocol [11] defines an implicit push operation and an explicit pull operation. When the routing metric of a node changes significantly, CTP quickly transmits a beacon message with the new routing metric, which constitutes an implicit push operation. This makes the new information propagate faster than with the usual periodic dissemination. If a node detects that its routing metric is out of date, e.g., if a loop is detected or if the node has just started, CTP sends a beacon with a pull-bit set, to which neighbors respond by sending a beacon. This is an explicit pull operation.

Single-packet data dissemination with Trickle. The Trickle single-packet dissemination protocol [15] defines implicit push and pull operations. When Trickle starts sending a new version of the data to be disseminated, Trickle nodes issue

an implicit push to rapidly propagate the new version through the network. Also, if a node notices that a neighbor has an older version than the current global version, the node performs an implicit push by directly broadcasting a beacon with the latest version. When a node boots up, it performs an implicit pull operation by sending a beacon with version number zero, which causes neighbors to broadcast the latest version.

Low-power IPv6 routing with RPL. The RPL IPv6 routing protocol defines explicit push and pull operations [21,23]. Nodes periodically transmit DODAG Information Objects (DIO) messages to let nodes in their one-hop neighborhood discover and maintain routes. The DIOs are transmitted at adaptive intervals and RPL also uses suppression to reduce the amount of transmissions. The combined effect of adaptive intervals and suppression may cause a node that has just started or moved into a new neighborhood to wait for a long time before getting a DIO. An RPL node may therefore send a DODAG Information Solicitation (DIS) message, to which neighbors reply with DIO messages. This constitutes an explicit pull.

Neighbor discovery. Many neighbor discovery protocols for mobile sensor networks [8,9,13,25] define both push and pull operations, which may be triggered by physical mobility. Nodes transmit beacons to announce their presence, constituting an explicit push, and may request information from their neighborhood to gather the identity of surrounding devices when the connectivity may change because of a changing physical location.

3.3 Protocol Implementations with an Announcement Layer

To demonstrate the feasibility of the announcement layer as a programming primitive for network protocols, we have rewritten three of the most common sensor network protocols to use announcements: data collection, single-packet data dissemination, and multi-packet data dissemination. All three protocol implementations are based on the original implementations in Contiki [7]. The data collection protocol is Contiki collect, an address-free, tree-based collection protocol similar to the TinyOS Collection Tree Protocol [11]. The single-packet data dissemination protocol is based on the Trickle protocol by Levis et al [15]. The multi-packet data dissemination protocol is Deluge [12]. We describe the implementation of the data collection protocol in detail below.

The starting point for our announcements-based data collection protocol is the Contiki collect protocol. Contiki collect builds a tree, rooted at the sink, by letting each node estimate the expected number of transmissions (ETX) to reach the sink. Nodes outside the neighborhood of the sink select the neighbor with the lowest ETX routing cost as their parent in the tree. Each node announces its ETX value to its neighbors through periodic beacons. The beacons are transmitted with an increasing interval, but when a node finds a significantly better parent, the beacon interval is reset to a low value.

The original Contiki collect module [7] uses periodic beacons to advertise routing cost. In our rewritten variant, the protocol instead uses announcements to advertise its routing cost. An excerpt of the rewritten version is shown as

```
collectInit() {
  register(COLLECT_KEY, NODE_SCOPE)
  pull(COLLECT_KEY)
}
receivedAnnouncement(fromAddress, etx) {
  addNeighbor(fromAddress, etx)
  updateLocalETX()
  setValue(COLLECT_KEY, localETX)
  if (newParent) {
    push(COLLECT_KEY)
    setMinRate(COLLECT_KEY, lowestRate)
  }
}
sendDataPacket() {
  if (parent == nil) {
    pull(COLLECT_KEY)
  } else {
    sendto(parent)
  }
}
```

Fig. 8. The relevant parts of the data collection protocol with announcements, in pseudo code

pseudo code in Fig. 8. When the collection protocol is initiated, it registers an announcement with a pre-defined key and with the node scope. This announcement is used for advertising route metrics. When the node starts, it does not have any route information and therefore issues a *pull* to get route information from neighbors. Likewise, when a node has no parent, it performs a *pull* to obtain one. When a new parent has been found, the node does a *push* to let others know about its new route.

4 Implementation

We have implemented an announcement layer in the Contiki operating system and the Rime network stack [7]. The announcement layer is implemented as a separate Rime module that uses a Rime broadcast channel to send and receive its beacon messages.

Beacon coordination consolidates all announcements into each beacon packet, but technology-specific limitations on radio packet size may restrict the amount of announcements that can be consolidated into each packet. For example, the popular 802.15.4-2006 standard defines a maximum packet size of 127 bytes[1]. Our announcement layer implementation handles this by breaking up large beacons into multiple broadcast transmissions.

To avoid instantaneous congestion caused by network synchronization, our implementations of the *push* and *pull* operations incur a random wait period before the beacons are transmitted. In our implementation, we set the waiting period to a random time between 0 and 8 seconds.

[1] Upcoming versions of the standard increase the maximum packet size to 2047 bytes.

5 Evaluation

We evaluate three aspects of the announcement layer. First, we quantify the beacon coordination mechanism and the resulting reduction in power consumption. Second, we quantify the cost of the announcement operation in terms of power consumption and the number of packet transmissions. Third, we use a testbed experiment to study a sensor network with concurrent protocols.

We use both simulation and testbed experiments. All simulations and experiments are carried out with Tmote Sky motes. For our simulations, we use the Contiki Cooja network simulator and the MSPsim Tmote Sky emulator [20]. Cooja and MSPsim provides a cycle-level accurate emulation of the MSP430 microcontroller and a bit-level accurate emulation of the CC2420 radio transceiver, which makes it possible to correctly emulate low-level protocols such as radio duty cycling mechanisms. For our testbed experiments, we use a 24-node Tmote Sky testbed in an office environment with a wired backchannel through which we obtain logging information. Throughout our experiments, we use Contiki 2.5 and the ContikiMAC low-power listening duty cycling mechanism with a channel check rate of 8 Hz, which results in an idle duty cycle of 0.5%.

We use the radio duty cycle as a proxy for energy consumption because the radio transceiver is the most power consuming component. We use Contiki's power profiler to measure the radio duty cycle [6], both the amount of time that the radio spends in listen mode and in transmit mode.

5.1 Beacon Coordination

The purpose of beacon coordination is to reduce the number of beacon transmissions by consolidating all announcements into every beacon and by suppressing the transmission of redundant beacons.

To evaluate the effectiveness of the beacon coordination mechanism, we set up a system with a variable number of announcements and set a fixed minimum rate of ten seconds for each announcement. We vary the number of announcements and measure the number of beacons that get transmitted as well as the total power consumption of the system. We run the system both with and without beacon coordination.

Figure 9 shows the result. We see that without beacon coordination, the number of beacons per interval increases with the increasing number of announcements. With beacon coordination, however, the number of beacons remains at one per interval. Similarly, without beacon coordination, the power consumption grows with the number of announcements, but with beacon coordination the power consumption stays almost constant, even though there is a slight increase in power consumption due to the additional size of each beacon.

5.2 The Cost of Announcement Operations

The push and pull announcement operations involve the transmission and reception of network traffic, incurring power consumption in the involved nodes. We quantify the effect of these operations on the power consumption by conducting

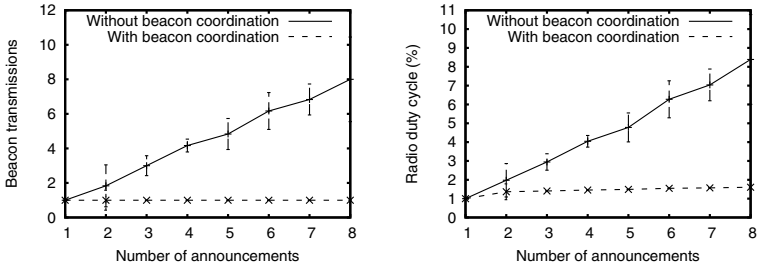
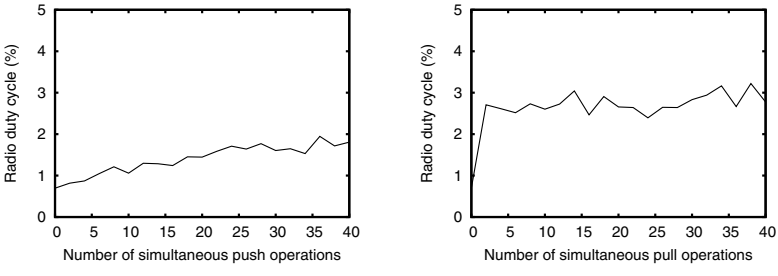


Fig. 9. The number of beacon transmissions with and without beacon coordination (left). The power consumption with and without beacon coordination (right).



(a) The cost of push (b) The cost of pull

Fig. 10. The cost of the push and pull operations in a 40 node network where all nodes are in range of each other

three experiments. First, we measure the effect on power consumption caused by the push operation in a dense network with a varying number of nodes that issue a push operation. Second, we measure the effect of the pull operation in the same situation. Third, we quantify the marginal cost of an increasing number of announcements being pushed in a single push operation.

To quantify the cost of the push operation, we set up a simulated network with 40 nodes. A push operation results in a broadcast transmission, which reaches all nodes in range. To create a situation in which the push operation was as expensive as possible, we set up our network so that all nodes are transmission range of each other. The nodes issue a push every ten seconds and we vary the amount of nodes that issue a push from one to all nodes. We measure the radio duty cycle of the nodes over the ten seconds between each push operation. The result is shown in the left graph in Fig. 10. As expected, we see that the cost grows linearly with the amount of nodes issuing a push.

The right graph in Fig. 10 shows the result of the same experiment, but with nodes issuing pull operations instead of push operations. We see that the cost is higher than for the push operation, but that it is relatively constant regardless of the number of nodes that issue a pull operation. This is due to the delay between the reception of a pull request and the corresponding push response: with many

pull requests, nodes will receive several requests before eventually responding with a push. Thus the resulting power consumption is not significantly affected by the number of simultaneous pull operations.

5.3 Case Study: Collection and Dissemination

To study the aggregate effects of announcements on a real-world scenario, we perform a data collection testbed experiment. We use the Contiki shell to collect sensor data from a 24 node office testbed. The Contiki shell has one command for setting up a sink node, `collect`, which forms a collection tree with the Contiki collect protocol, and one command for sending data through the collection tree, `send`. To start the commands on the nodes in the network, the Contiki shell provides a mechanism for starting commands on other nodes in the network, `netcmd`. The `netcmd` command uses reliable data dissemination with Trickle to disseminate the commands through the network. Both the data collection protocol and the data dissemination protocols use beacons and we expect to see a reduction in the number of beacons in the network.

We run two versions of the experiment, one with announcement-based implementations of the protocols and one without. In both experiments we use ContikiMAC with a channel check rate of 8 Hz. We run the network for one hour for each experiment. With both experiments, we receive an average of 54 packets per node. Two nodes have poor connectivity and only reported 1 and 3 packets respectively in one experiment, and 1 and 7 packets in the other experiment, whereas the others reported 60 packets. The longest path was 5 hops long.

We measure the power consumption per Rime channel using Contiki's power profiler [6]. We see the resulting breakdown in Fig. 11. The boxes show the amount of transmission and reception power spent on beacons and data packets, respectively. The results show that the announcement-based implementation is

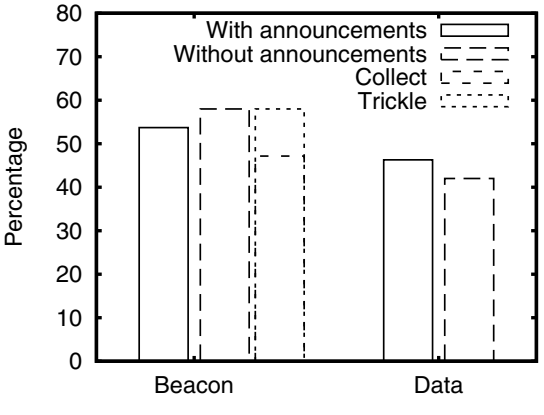


Fig. 11. Activity breakdown of the data collection and dissemination testbed experiment

able to reduce the number of beacons. The reduction is due to the suppression of data dissemination beacons, which account for 9% of the total number of beacons in the non-announcement-based implementation.

6 Related Work

The idea of inserting a new layer in the network stack to coordinate data from multiple upper-layer protocols has been used in many contexts. Balakrishnan et al. [1] introduced an explicit congestion management layer for Internet hosts. Choi et al. [3] add an isolation layer that shields different sensor network protocols from each other. The announcement layer is different because it focuses on a specific traffic type: broadcast beacons. Furthermore, since the announcement layer do not shield protocols from each other, there is no performance penalty as for the isolation layer by Choi et al [3].

There are many examples where information from multiple packets are combined into a single transmission to improve performance. Lin and Levis [16] observe that packing multiple pieces of information into the same physical packet aids in reducing the performance penalty due to broadcast transmissions. However, their scope is limited to information belonging to a single protocol (DIP), and comes hardwired with the protocol implementation itself. By contrast, the announcement layer provide a re-usable, generic mechanism that can be used across different protocols.

The push and pull operations of the announcement layer are similar to the operations used in sensor network neighborhood abstractions [18,22]. However, the latter aim at redefining the notion of physical neighborhood mostly based on application-level requirements. Announcements, instead, target network-level functionality that typically leverage communication in the physical neighborhood. In addition, some of the aforementioned systems [18,22] inherently provide a push-only communication paradigm, whereas announcements also provide a pull operation.

7 Conclusions

We present the announcement layer that piggybacks announcements from multiple protocols and coordinates their transmission to reduce the total amount of beacons. The background to the announcement layer is the observation that beacon transmissions are costly, and multiple transmissions even more so. In addition to beacon coordination, the announcement layer provides inter-protocol coordination through two operations: push and pull. We have implemented an announcement layer in Contiki and rewritten three staple sensornet protocols on top of it: data collection, single-packet data dissemination, and multi-packet data dissemination. We demonstrate that beacon coordination reduces the amount of beacons and that the cost of the push and pull operations is low.

Acknowledgments. This work was funded by the SSF, the Swedish Foundation for Strategic Research through the Promos and Supple projects, and VINNOVA, the Swedish Agency for Innovation Systems through the ReSense project.

References

1. Balakrishnan, H., Rahul, H., Seshan, S.: An integrated congestion management architecture for internet hosts. SIGCOMM Comput. Commun. Rev. 29(4), 175–187 (1999)
2. Buettner, M., Yee, G.V., Anderson, E., Han, R.: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), Boulder, Colorado, USA, pp. 307–320 (2006)
3. Choi, J., Kazandjieva, M., Jain, M., Levis, P.: The case for a network protocol isolation layer. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), Berkeley, CA, USA (2009)
4. Costa, P., Mottola, L., Murphy, A.L., Picco, G.P.: Programming wireless sensor networks with the TeenyLIME middleware. In: Cerqueira, R., Pasquale, F. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 429–449. Springer, Heidelberg (2007)
5. Deshpande, A., Guestrin, C., Madden, S.: Resource-aware wireless sensor-actuator networks. IEEE Data Engineering 28(1) (2005)
6. Dunkels, A., Österlind, F., Tsiftes, N., He, Z.: Software-based on-line energy estimation for sensor nodes. In: Proceedings of the IEEE Workshop on Embedded Networked Sensor Systems (IEEE Emnets), Cork, Ireland (June 2007)
7. Dunkels, A., Österlind, F., He, Z.: An adaptive communication architecture for wireless sensor networks. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), Sydney, Australia (November 2007)
8. Dutta, P., Culler, D.: Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), Raleigh, North Carolina, USA (2008)
9. Dyo, V., Mascolo, C.: Efficient node discovery in mobile wireless sensor networks. In: Nikoletseas, S.E., Chlebus, B.S., Johnson, D.B., Krishnamachari, B. (eds.) DCOSS 2008. LNCS, vol. 5067, pp. 478–485. Springer, Heidelberg (2008)
10. El-Hoiydi, A., Decotignie, J.D., Enz, C.C., Roux, E.L.: Wisemac, an ultra low power mac protocol for the wisenet wireless sensor network. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), pp. 302–303 (2003)
11. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), Berkeley, CA, USA (2009)
12. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), Baltimore, Maryland, USA (November 2004)
13. Jiang, J., Tseng, Y., Hsu, C., Lai, T.: Quorum-based asynchronous power-saving protocols for IEEE 802.11 ad hoc networks. Mobile Networks Applications 10(1-2) (2005)

14. Kandhalu, A., Lakshmanan, K., Rajkumar, R.: U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In: Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN), Stockholm, Sweden, pp. 350–361 (2010)
15. Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: Proceedings of the USENIX Symposium on Networked Systems Design & Implementation (NSDI 2004) (March 2004)
16. Lin, K., Levis, P.: Data discovery and dissemination with dip. In: Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN), St. Louis, MO, USA (2008)
17. Moss, D., Levis, P.: BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Tech. Rep. SING-08-00, Stanford University (2008)
18. Mottola, L., Picco, G.: Logical neighborhoods: A programming abstraction for wireless sensor networks. In: IEEE International Conference on Distributed Computing in Sensor Systems (2006)
19. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), pp. 95–107. ACM Press, Baltimore (2004)
20. Tsiftes, N., Eriksson, J., Finne, N., Österlind, F., Höglund, J., Dunkels, A.: A Framework for Low-Power IPv6 Routing Simulation, Experimentation, and Evaluation. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM), Demo Session, New Delhi, India (August 2010)
21. Vasseur, J., Dunkels, A.: Interconnecting Smart Objects with IP: The Next Internet. Morgan Kaufmann, San Francisco (2010)
22. Whitehouse, K., Sharp, C., Brewer, E., Culler, D.: Hood: a neighborhood abstraction for sensor networks. In: Proceedings of The International Conference on Mobile Systems, Applications, and Services (MobiSys), Boston, MA, USA (June 2004)
23. Winter, T., Thubert, P.(eds.) : RPL Author Team: RPL: IPv6 Routing Protocol for Low power and Lossy Networks, internet Draft draft-ietf-roll-rpl-11 (work in progress)
24. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys), pp. 14–27. ACM Press, Los Angeles (2003), `http://citeseer.ist.psu.edu/woo03taming.html`
25. Zheng, R., Hou, J., Sha, L.: Asynchronous wakeup for ad hoc networks. In: Proceedings of the 4th ACM International Symposium on Mobile ad hoc Networking & Computing (2003)