Leveraging IP for Sensor Network Deployment

Simon Duquennoy, Niklas Wirström, Nicolas Tsiftes, Adam Dunkels
Swedish Institute of Computer Science
{simonduq,niwi,nvt,adam}@sics.se

ABSTRACT

Ease of deployment has always been seen as a major selling point of wireless sensor networks, yet experience has shown deployment to be difficult. We argue that parts of these difficulties have come from the lack of a generic networking layer and of well-tested, generic transport protocols in traditional sensornet deployments. We believe that the use of low-power IPv6 can help by providing node-level addressing, point-to-point routing, and generic well-tested transport protocols. We evaluate the performance of HTTP/TCP and CoAP/UDP over a duty cycled radio layer, showing that with a small modification to the duty cycling layer results in a dramatic improvement in performance at a retained low power consumption. Based on our experiences, we introduce an in-network caching mechanism that significantly improves the performance of software updates in incrementally deployed sensor networks. Our results are the first steps towards a deployment tool for IP-based sensor networks.

1. INTRODUCTION

Although ease of deployment is often touted as a primary advantage of wireless sensor networks, deployment remains difficult [9, 15]. Part of this difficulty stems from the lack of a generic networking layer. Existing sensor network systems are intended to serve a single purpose, where all nodes serve the same purpose. For example, many TinyOS deployments run periodic data collection with the TinyOS Collection Tree Protocol (CTP) [6]. Likewise, many Contiki deployments run periodic data collection with the Contiki collect protocol. In such deployments, nodes that do not run the collection protocol cannot route collection packets.

Moreover, to address individual nodes from the sink node, for example to be able to configure the nodes, the collection protocol cannot be used. Instead, all nodes must run an additional, dedicated protocol in addition to the collection protocol. Similarly, for software updates, the collection protocol cannot be used, so all nodes must run one more, dedicated protocol. This makes networks inflexible and difficult to change after deployment. To make matters worse, if those dedicated protocols are used only rarely, their behavior has often not been tested in the deployed system, which may make them fail silently when they for once are needed, as shown by experiences with network reprogramming in deployments [9, 15]. These are obviously not problems with data collection protocols per se, but with the use of a single, dedicated protocol. With IP, which is a generic network layer, this inflexibility goes away.

In many emerging sensor network applications, the network is heterogeneous. For example, in a building automation system, nodes with light sensors can be placed in windows to determine the available sunlight [11] or near appliances to monitor their usage [13]. Both applications can make efficient use of the same hardware, thereby invoking economics of scale in the manufacturing of the devices, but their application is very different. By being able to dynamically change software at deployment time, one can tailor individual deployments to their specific requirements without the need for special hardware. Current sensor network deployment methods do not support such applications.

Traditionally, the sensor network deployment problem has been explored as the problem of how or where to place individual sensors to achieve good sensor or network coverage [17, 18]. Specialized sensor placement systems also have been developed [10]. We argue that node configuration and software updates are other aspects of the deployment problem that must be considered as well.

We make three contributions with this paper. First, we measure the performance of RPL during incremental deployment, showing that RPL is able to quickly find routes at a low energy cost during network deployment. Second, we show that the performance of HTTP/TCP and CoAP/UDP can be improved significantly by adding a low-power streaming mechanisms at the radio duty cycling layer. To the best of our knowledge, this is the first quantitative comparison between HTTP/TCP and CoAP/UDP. Third, we introduce a novel in-network caching mechanism for bulk data transfer in low-power IP networks. This mechanism alleviates the need for dedicated bulk data transfer protocols.

The remainder of this paper is structured as follows. In Section 2, we discuss how the use of a generic sensor network layer helps deployment. In Section 4 we evaluate the performance of the IETF RPL protocol during deployment. In Section 5 we investigate software installation over IP and in Section 6 we demonstrate that in-network caching in a low-power IP network improves performance of software installation. We discuss related work in Section 7. We lay out the direction for future work in Section 8 and conclude the paper in Section 9.

2. ADDRESSING THE DEPLOYMENT PROBLEM WITH IP

We argue that the use of IP helps mitigate the sensor network deployment problem. IP provides a generic network layer on top of which applications can be built without having to provide low-level network details such as routing. Transport and application protocols atop IP provide established and well-tested mechanisms for data transport for network reprogramming. Unlike dedicated protocols for software updates, which have not been tested for their target environment, generic transport protocols are well-tested.

Should the transport layer not work in a particular deployment, the network fails immediately instead of failing silently when the network needs to be updated.

IP allows individual nodes to be addressed. During deployment, this can be leveraged to configure nodes individually. This is needed to set node-specific configuration parameters, and is particularly useful for heterogeneous deployments. By using an IP routing protocol, nodes can be addressed even without the entire network being deployed. This allows nodes to be assigned roles and given configuration parameters in parallel with the network being deployed.

IP has a generic network layer that is shared by all nodes. Such a layer allows applications to be deployed on only a few nodes, without requiring the other ones to participate in the application. This is in contrast to pure data collection protocols such as CTP and Contiki collect, which require all nodes to participate in the data collection network. Allowing nodes to run different applications, yet collaborate to forward packets, is useful both for heterogeneous deployments and during the deployment phase, when only a subset of all nodes have been programmed with their applications.

IP provides generic transport layers as well. For network reprogramming, the IP architecture provides two alternatives: the TCP and UDP transport protocols. TCP provides a reliable stream service that can be used to transmit binaries to network nodes. But TCP is known to have throughput issues in wireless networks due to path losses being interpreted as congestion. Recently, new dedicated protocols for low-power IP have been developed, such as the CoAP protocol [12]. CoAP provides lightweight RESTful interactions in constrained environment. It runs over UDP, and provides a bulk data transfer mechanism. It performs its own loss detection and retransmission to avoid falling into the same problems as TCP.

3. EXPERIMENTAL SETUP

To support our case, we present the performance of a set of deployment scenarios over low-power IP. We use the Contiki operating system which provides an IPv6 implementation [5]. To be able to perform controlled experiments, we use the Contiki simulation environment for our measurements. The Contiki simulation environment consists of the Cooja network simulator and the MSPsim node-level emulator. Cooja is able to simulate independent random losses. The MSPsim emulator provides a cycle-accurate emulation of the individual nodes, including the radio transceiver, allowing faithful emulation of timing-sensitive radio duty cycling protocols. The motes software used in the simulation is a ready-to-use msp430 binary file including Contiki, the uIPv6 stack and ContikiRPL.

At the network layer, we use the IETF RPL protocol [16, 14]. The RPL protocol builds a directed acyclic graph through which packets can be efficiently routed to sink nodes. Although RPL is designed to primarily be an efficient many-to-one data collection protocol, it also supports any-to-any traffic. From the sink, RPL builds routes to nodes inside the network. These routes are used to distribute software to nodes in the network.

We use the ContikiMAC radio duty cycling protocol [3] because of its energy efficiency: ContikiMAC has an idle duty cycle below 1%. To study the energy consumption, we use Contiki's built-in power profiler [4], which measures the time during which individual components are turned on. We use the radio on-time as a proxy for energy consumption, partly because the radio transceiver is the most power-demanding component in typical motes, and partly because it enables quantitative comparisons of our energy results with those of others who may not use the same hardware platform as we do. In the following sections, all figures show the mean and standard deviation of results gathered over 10 simulations.

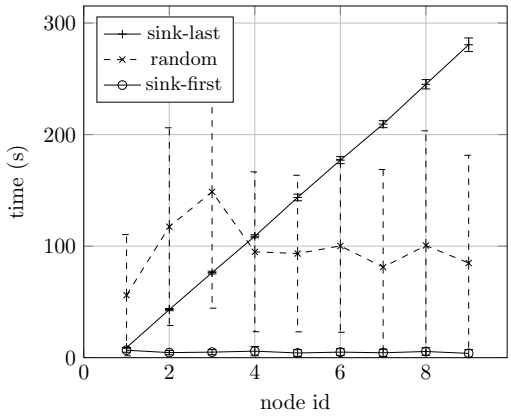


Figure 1: The time between node boots up and until it becomes part of the routing graph. The x-axis shows the number of hops from sink.

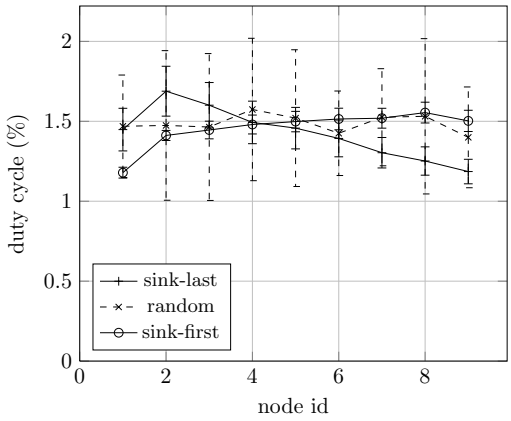


Figure 2: The radio duty cycle of the RPL nodes in Figure 1.

4. INCREMENTAL NETWORK DEPLOYMENT WITH RPL

We first measure the ability for low-power IP to address individual nodes during deployment. To investigate the performance of RPL under different deployment scenarios, we run a set of simulations based on three setups: incremental deployment starting with the sink node (*sink-first*), incremental deployment starting with the node farthest from the sink (*sink-last*), and random deployment (*random*) starting with the sink. In all three cases, nodes are deployed on a line, and each node can hear only its two immediate neighbors. Nodes are deployed at a rate of one node each 30 seconds.

We measure the time passed before new nodes are incorporated in the routing graph, and the energy consumption per node during the first 8 minutes after the first node was deployed. The reason for this long duration is to let the RPL control traffic settle down. Figure 1 shows the results for the three different deployment orders. We use

default values for all RPL parameters, except for parameter *Imin* which is set to approximately 4 s, instead of the default value of 8 ms. *Imin* determines a upper bound for how frequent RPL nodes will broadcast routing information; a value of a few milliseconds is not suitable for use with a duty-cycled radio. Additionally, new nodes send RPL neighbor discovery (DIS) messages each minute until they obtain a parent. In these experiments, we use a 15% link loss rate.

Figure 1 shows the time it takes for each node to be incorporated into the RPL routing graph. The time is measured per node, from the moment the node is deployed and powered on, until it receives a parent in the routing graph. The result is intuitive: the sink-last case shows the highest latency because no nodes will obtain connectivity until the last node has finally been deployed. The sink-first case shows the lowest latency because when a node is deployed, it can be immediately incorporated into the routing graph because one of its neighbors is already connected. The difference in connection times between two neighboring nodes in the sink-last scenario is 33.9 s. Discounting the 30 s deployment interval yields a duration of 3.9 s. In the sink-first scenario, this duration is just 0.4 s because nodes receive routing information from candidate parents immediately after sending out the first DIS message.

Figure 2 shows the measured radio duty cycle for the three different scenarios. We see that the duty cycle is low (around 1.5%) and although the duty cycle is relatively constant across the three scenarios, the duty cycle for the sink-last case exhibits a linear behavior that peaks at node 2. This is due to the increased control traffic resulting from the incremental deployment of nodes with no connection to a routing graph. Such nodes send more frequent DIS messages than nodes that are part of a routing graph, resulting in a slightly higher power consumption. Node 1 has only one neighbor (node 2), and is less affected by such traffic than the others are.

5. SOFTWARE INSTALLATION OVER LOW-POWER IP

We now turn to investigating the performance of software updates in low-power IP networks. In traditional sensor networks, software updates have been distributed by using full-network flooding protocols such as Deluge. In the next section, we look into how such behavior can be achieved in the IP architecture, whereas in this section, we establish a baseline for software updates transmitted in a point-to-point fashion. We set up a system where an application server holds software that the nodes can request.

We use both UDP and TCP, the two main transport layer protocols of the IP protocol stack. Control commands are sent using CoAP, whilst the actual download of the application is done either using CoAP or TCP. In the first case, we use CoAP chunked traffic: the node requests consecutively every single chunk of the file. In the second case, the node opens a TCP connection to the application server and downloads the file. The node advertises a window of a single segment, which allows for a low-footprint TCP implementation and avoids the well-known problems of TCP over wireless links. In both cases, every part of the file (either CoAP chunk or TCP segment) is sent and acknowledged one by one, having each the maximum size that can fit a 802.15.4 frame.

Our duty-cycling protocol, ContikiMAC, is a sender-initiated protocol in which every node periodically checks the channel for traffic (by default every 125 ms). When a node needs to send a packet, it repeatedly sends it until it gets a link-layer acknowledgment from the receiver. To save more energy, the sender remembers the timing of successful transmissions towards each of its neighbors, and uses this information to synchronize future transmissions with the receiver's duty-cycling phase.

To accelerate multi-hop data forwarding, we add a mechanism in which the duty cycling behaves differently in periods identified as *busy*. Every time a node sends or transmits a packet, it triggers a timer and switches to busy mode. The busy mode terminates when the timer expires. We use a default timer duration of 1 second, which means that the node is in a busy period if it handled some traffic in the last second. During busy periods, we propose two optimizations called streaming and snooping. This results in three possible behaviors:

Default Do not adapt to busy periods;

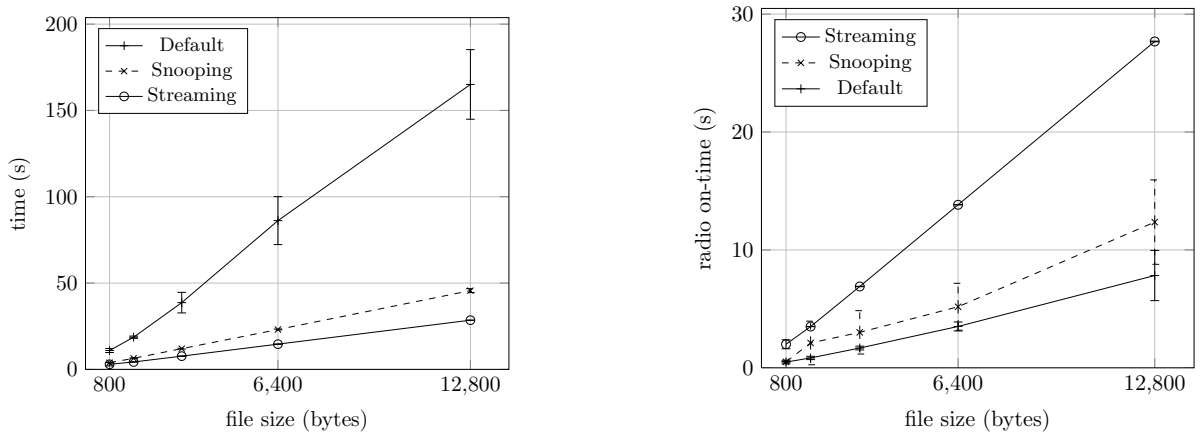
Streaming Keep the radio on during busy periods. The synchronization on the sender is disabled;

Snooping Increase the channel check frequency by a factor of 8 (64 cycles per second). The synchronization on the sender is disabled as well.

Figure 3 shows the time and energy spent when installing an application on a node depending on the size of the executable file. The node is located 4 wireless hops from the application server. The completion time and radio on-time are measured from the user's request to install a new code to the final notification confirming that the process The completion time includes the download and the installation of the application. The overall cost is almost linear since the download of the executable is from far the most demanding operation regarding time and energy. Both snooping and streaming mechanisms allow for a substantial acceleration of the process, from more than 150 s to less than 50 s in the case of 12800 bytes files. Regarding radio on-time, streaming is much more demanding than snooping, which is almost as energy-efficient as default duty cycling.

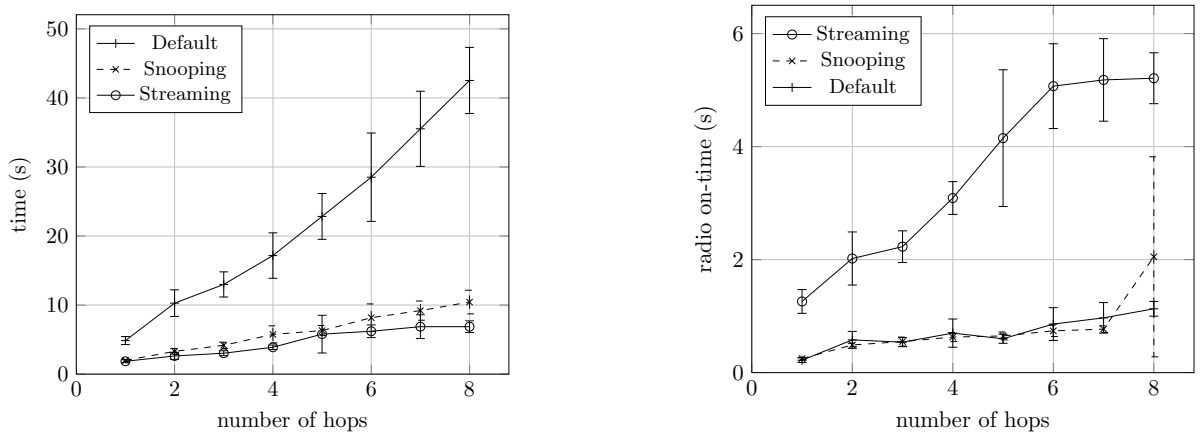
Figure 4 shows time and energy versus the number of wireless hops from the node to the application server, for a 800 bytes executable file (the size of a simple hello-world elf file for Contiki). We use a loss rate of 5 %. Note that every node in the path have no other task than packets forwarding. This operation that can be handled by any IPv6-ready entity, independently of our deployment architecture. Since CoAP chunks mode handles end-to-end retransmissions, all the simulations ended with a successful data download and installation.

Figure 5 compares TCP with CoAP chunks with 4 hops, a 800 bytes file and a loss rate of 5 %. In all configuration, both protocols provide similar results in time and energy. This is because in both cases, bulk data transfer is handled in a chunk-per-chunk manner. Here again, snooping and streaming provide fast operation (around 67 % faster than default), whilst default and snooping are the less energy consuming solutions (81 % more efficient than streaming). In all our experiments, snooping presents an interesting trade-off between performances and energy. This shows that current standard solutions can be used for data transfer over duty-cycled networks, but can be improved significantly by adaptations made at the MAC layer.



(a) Completion time (b) Energy consumption

Figure 3: Time and energy as a function of file size over 4 hops. Both time and energy grow linearly with the file size.



(a) Completion time (b) Energy consumption

Figure 4: Time and energy as a function of the number of hops with 5% per-hop loss. Both CoAP and TCP-based approaches work well in a lossy environment. The default duty cycling mechanism is the slowest while streaming provides the highest energy consumption. Snooping arguably presents the best time-energy trade-off.

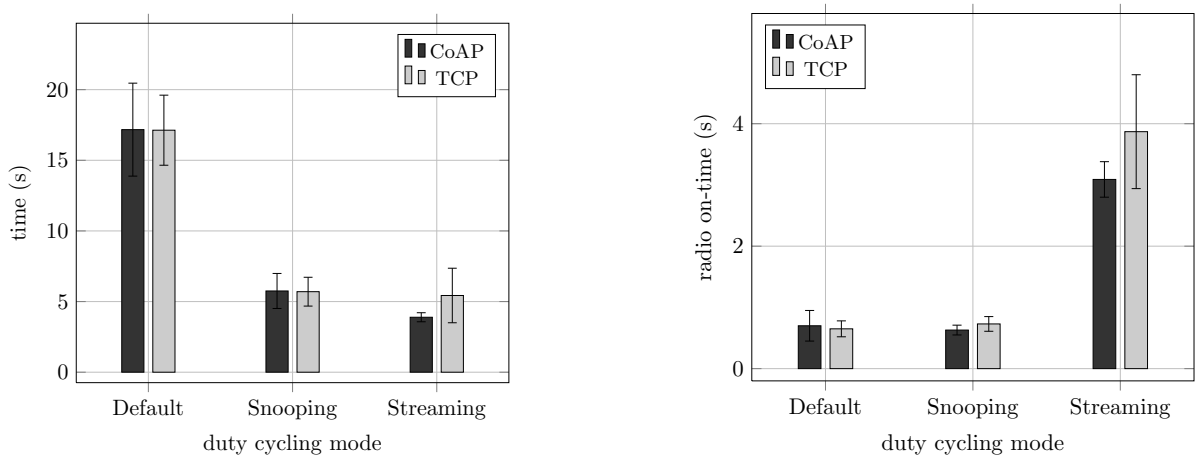
6. EXPLOITING IN-NETWORK DATA CACHING

During network configuration, a substantial amount of data might need to be transmitted before the network becomes fully functioning. Several flooding protocols have been proposed for efficient dissemination of data in a network [8, 7]. The aim of these protocols is to deliver the same information to all nodes in the network. Instead of proposing a new dedicated bulk data flooding protocol, we investigate how existing standardized protocols can be used to improve dissemination efficiency.

We evaluate two different application uploading strategies using CoAP. One strategy lets all nodes download the application only from the sink, as in the experiments in Section 5, while the other lets nodes store the application to secondary storage once it has been downloaded, and set up a CoAP server to let other nodes download the application from it. The sink then sends a message to a newly deployed node, specifying which host it should download the application from. This strategy selects the physically nearest node. Both strategies will initiate an application uploading to a new node only if the previous node has completed. We evaluate both strategies using the sink-first, and sink-last deployment scenarios from Section 4. The size of the application is 800 bytes, and we use a packet loss rate of 15 %.

Figure 6 (a) shows the time it takes for each node to complete the download of an application. The time is measured from the moment the node is powered on, and, as in Section 5, until the server has been notified that the download is complete. In the sink-first case, the two strategies yield similar completion times for nodes of lower number of hops from the sink, but the curves diverge for nodes farther away. This is intuitive because, in the no-caching case, the download times increase at least linear with the number of hops, whereas in the caching case, the download times are constant for all hops (disregarding the upload-initiation and completion-notification messages).

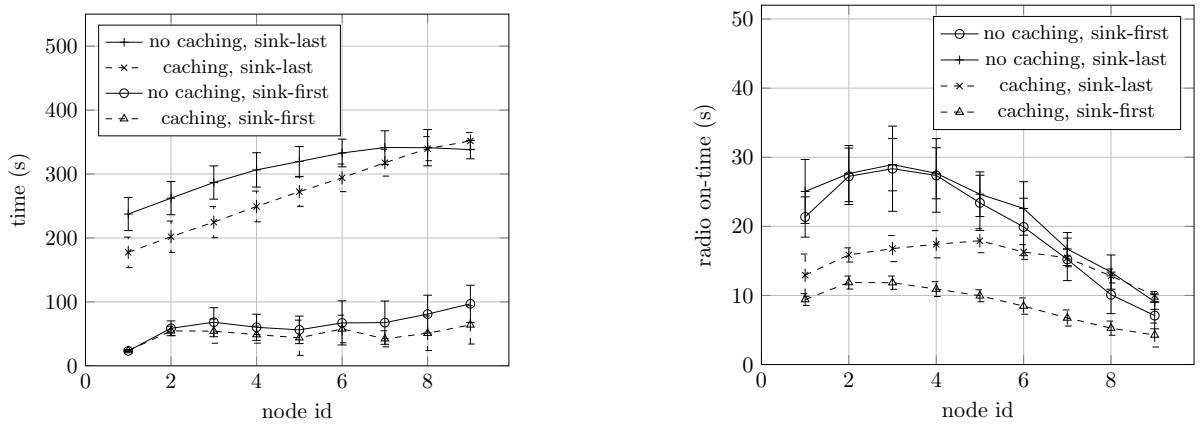
For the sink-last case, completion times are similar for nodes far away from the sink, and converge for nodes closer to the sink. This



(a) Completion time

(b) Energy consumption

Figure 5: Comparing TCP with CoAP chunks. Both protocols provide similar results.



(a) Completion time

(b) Energy consumption

Figure 6: In-network data caching reduces both installation time and energy consumption.

is a result of that nodes do not start a download until all earlier deployed nodes have completed their downloads, and because no node has connectivity to the sink until the last node is deployed. In this case, it is clear that one could much faster obtain a partially working network using a smarter algorithm for selecting the order in which nodes should start downloading.

Figure 6 (b) shows the energy consumption of the experiments. The plots for the two no-caching scenarios are similar. The plot for the sink-last case is shifted upwards in relation to that of the sink-first case, with an amount corresponding to the extra cost of waiting until the last node is deployed before starting the download. Similarly, in the caching case, comparing the consumption of the one hop nodes in the sink-first and sink-last cases, an similar shift is found. In the sink-first case, the average radio-on time is 8.7 s for the caching case, and 20.0 s no-caching case. In other words, in-network caching uses only 43.5% of the energy needed for the no-caching case. In the sink-last case, caching needs approximately 70% of the energy (average radio-on times are 21.7 s and 15.1 s, respectively).

7. RELATED WORK

Traditionally, the sensor network deployment problem has been explored as the problem of how to place sensors to achieve a good sensor coverage [17], to achieve a good network coverage [18], or as being tailored to a specific application [10]. Similarly, many methods to reprogram a network of deployed nodes exist [2, 8, 7].

Beutel et al. [1] propose the use of a secondary network, a so-called deployment support network, to solve parts of the deployment problem. The deployment support network enables statistics gathering, and is intended to be used in the prototyping phase. In the prototype by Beutel et al., Bluetooth was used as the secondary network. By contrast., we aim at assisting deployments of the final deployment, not only during the prototyping phase.

The problem of software updates has been thoroughly studied. Protocols such as Deluge [8] allow binaries to be transported across the network. On the nodes, different ways to install the binary has been investigated [2]. We present a viewpoint that is opposed to much of this work: we argue that software updates should not be done using dedicated and specialized protocols but that generic protocols

provide improved stability.

8. FUTURE WORK

The results presented in this paper have been obtained through simulation only. While simulation is a powerful tool that allows us to study the performance of protocols in a controlled way, the quality of the results depends heavily on the assumptions made by the simulator. Bad assumptions can easily result in results that do not match those obtained in an experimental setting. In our case, one of the biggest sources of deviation from experimental results is the loss model we have used, which assumes independent and identically distributed link losses. In reality, however, link losses follow bursty patterns, which may significantly change the behavior of HTTP/TCP and CoAP/UDP that we have observed in simulation. Thus the natural next step for us is to move our experiments into a testbed environment. Our system already runs in a testbed environment, but we have not yet made enough controlled experiments for us to draw conclusions that can be presented here.

Our results suggest that IP indeed is a useful tool to use during deployment. Based on our experiences, we are building a deployment tool for IP-based sensor networks. The tool can be used to configure and reprogram individual nodes based on their IP addresses. As we've seen, in low-power IP networks based on RPL, nodes can be addressed even during deployment, as long as there is a route to the sink. We see this tool as a way to reduce the deployment cost of IP-based sensor networks.

9. CONCLUSIONS

Deployment in wireless sensor networks has always been difficult. We envision networks in which nodes are individually given distinct capabilities and applications during deployment, leveraging IP as a generic network layer. This is opposed to traditional approaches where a dedicated protocol is used to update all the nodes of a network. We evaluate the feasibility of an IP-based deployment solution for duty-cycled sensor networks and show that the RPL protocol is able to quickly find routes during network deployment. We also show that a simple optimization in the duty-cycling layer can substantially improve the performance of both TCP and UDP, both in terms of throughput and energy consumption. Moreover, we show that the performance of bulk data dissemination using standard protocols can be significantly improved using in-network caching, without resorting to dedicated or custom protocols. These results point towards a future where sensor network deployment can be made radically simpler by leveraging the mechanisms provided by low-power IP.

Acknowledgments

This work was funded by the Swedish Strategic Research Foundation through the Promos and Supple projects and the EU Commission through the GINSENG project and NOBEL projects.

10. REFERENCES

[1] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable topology control for deployment-support networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, Los Angeles, California, 2005.

[2] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Boulder, USA, November 2006.

[3] A. Dunkels, L. Mottola, N. Tsiftes, F. Österlind, J. Eriksson, and N. Finne. The announcement layer: Beacon coordination for the sensornet stack. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2011.

[4] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. 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.

[5] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Raleigh, North Carolina, USA, November 2008.

[6] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Berkeley, CA, USA, 2009.

[7] S. Guo, Y. Gu, B. Jiang, and T. He. Opportunistic Flooding in Low-Duty-Cycle Wireless Sensor Networks with Unreliable Links. In *Proc. of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom '09)*, September 2009.

[8] J. W. Hui and D. Culler. 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.

[9] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, April 2006. IEEE.

[10] H. Liu, J. Li, Z. Xie, S. Lin, K. Whitehouse, J. Stankovic, and D. Siu. Automatic and robust breadcrumb system deployment for indoor firefighter applications. In *Proceedings of The International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 21–34, 2010.

[11] J. Lu, D. Birru, and K. Whitehouse. Using simple light sensors to achieve smart daylight harvesting. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, Zurich, Switzerland, 2010.

[12] Z Shelby and CoAP Author Team. Constrained Application Protocol (CoAP) draft-ietf-core-coap-04. Draft 4, January 2011.

[13] Z. Taysi, M. Guvensan, and T. Melodia. Tinyears: spying on house appliances with audio sensor nodes. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, Zurich, Switzerland, 2010.

[14] J.P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[15] M. Welsh. Sensor networks for the sciences. *Communications of the ACM*, 53:36–39, November 2010.

[16] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Internet Draft draft-ietf-roll-rpl-18, work in progress.

[17] S. Yang, M. Li, and J. Wu. Smart: A scan-based movement-assisted sensor deployment method in wireless sensor networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 2313–2324, 2005.

[18] B. Yener, M. Magdon-Ismail, and F. Sivrikaya. Joint problem of power optimal connectivity and coverage in wireless sensor networks. *Wirel. Netw.*, 13:537–550, August 2007.