TeaCP: a Toolkit for Evaluation and Analysis of Collection Protocols in Wireless Sensor Networks

Wei Si*, Morteza Hashemi*, Idan Warsawski*, Moshe Laifenfeld†
David Starobinski*, and Ari Trachtenberg*
*Dept. of Electrical and Computer Engineering, Boston University, USA
{weisi, mhashemi, idan, staro, trachten}@bu.edu
†Wireless Enablers Group, GM Advanced Technical Center, Israel
moshe.laifenfeld@gm.com

***Abstract*—Several collection protocols have been developed to achieve efficient gathering of data in Wireless Sensor Networks (WSN) including intra-car WSN. Though there exist WSN tools capable of controlling, monitoring, and displaying sensor data, there is still a need for a *general* benchmarking tool capable of visualizing, evaluating, and comparing the *network layer* performance of these protocols. In an effort to fill this gap, we present TeaCP, a prototype Toolkit for the evaluation and analysis of Collection Protocols in both simulation and experimental environments. Through simulation of an intra-car WSN and real lab experiments, we demonstrate the functionality of TeaCP for comparing the performance of two prominent collection protocols, the Collection Tree Protocol (CTP) and the Backpressure Collection Protocol (BCP).**

I. Introduction

Data collection is intrinsic to numerous applications in Wireless Sensor Networks (WSN), ranging from intra-car monitoring [8] to environmental data gathering [1] and military surveillance [2]. In a data collection network, sensor readings of wireless motes are routed towards a root[1] (sink). Though WSN routing protocols can accomplish the goal of delivering data to the root, another set of protocols, known as *collection protocols*, has emerged to support the specific needs of data collection. Typical requirements include:

- *reliability:* a high fraction (e.g., 90% or above) of the packets generated by the sources should be delivered to the root;
- *Quality of Service:* high throughput and low packet latency should be achieved;
- *robustness:* high reliability and QoS should be maintained even under stress conditions such as dynamic links.

Different applications may have different requirements. For instance, intra-car monitoring may emphasize robustness to wireless interferences (WiFi, Bluetooth, etc.), while military applications may also strive for low packet latency. Thus, in order to select the most suitable collection protocol for a given application, one should be able to evaluate the performance of different collection protocols in the same operating environment. In addition, the ability to visualize a network, including its topology and routes used by packets, is essential for understanding and troubleshooting the behavior of collection protocols, especially under stress.

Background: Many collection protocols have been proposed in the literature [3, 4, 6, 7, 12] and several of them have also been implemented. However, a common platform to visualize, analyze and compare their performance is still needed. Such a toolkit should not only enable the evaluation of multiple collection protocols in intra-car WSN, but also help to visualize the behavior of the collection protocols.

Though there already exist several visualization tools for WSN, most of them concentrate on controlling, monitoring and displaying sensor data rather than on analyzing the underlying network protocol. Some tools such as Mote-View [13] show packet loss statistics but these statistics are collected through the underlying collection protocol itself, possibly perturbing the ongoing test. Some of the tools are tailored for a specific collection protocol, in which case testers need to understand the tool and possibly write a significant amount of new code in order to test a different collection protocol. Besides, existing evaluations of collection protocols (Octopus [9] and SpyGlass [5]) sometimes resort to approximate calculations or only include partial aspects of the performance for sake of practical implementation. Though evaluation and analysis of collection protocols have been a common practice in this research area, none of the analysis tools have been published or made publicly available to the best of our knowledge.

It is within this context that we have designed and implemented TeaCP, an open-source benchmarking Toolkit for the evaluation and analysis of Collection Protocols in wireless sensor networks. TeaCP provides generic configurations for testing protocols, including packet generation rate and transmission power, and functions for post-test analysis. TeaCP can be used for both experiments with real data and simulation in TOSSIM [11]. For experiments, TeaCP utilizes out-of-band communication for logging data at all nodes, so that network events and packet information are captured regardless of network conditions. For simulations, TeaCP provides the convenience of testing the performance of collection protocols over a wide range of conditions. The post-experiment analysis functionalities allow evaluation of standard metrics, including reliability, throughput, and delay (which have been used for evaluating protocols in most previous works). TeaCP also permits visualization of the dynamics of the network topology and packet routes, illustrating the network layer behavior of

[1]Though data collection can also be used in multi-root scenarios, we frame our discussion for the single root case for sake of simplicity.

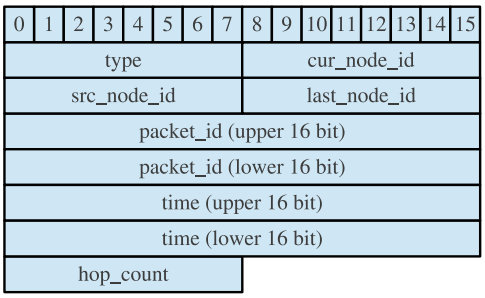

Fig. 1. Format of log message.

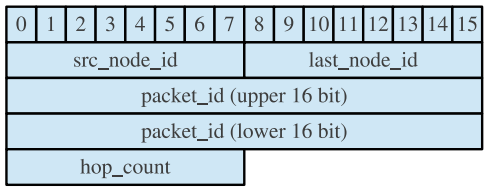

Fig. 2. Format of data message.

collection protocols over time.

As a case in point, TeaCP can be used to conveniently evaluate and compare intra-car collection protocols, using the TOSSIM simulator. To emulate a real vehicular environment, TeaCP is run using RSSI traces collected from actual measurements in a car. One can then evaluate the performance of protocols under different network configurations in a time-efficient manner.

In summary, the main features of TeaCP are the following:

- *Generality:* TeaCP is a general toolkit optimized to conveniently plug in various collection protocols and analyze their performance through post-experiment analysis.
- *Configurability:* With TeaCP, one can easily configure tests for evaluating a data collection protocol, run the tests and analyze the underlying performance (e.g., reliability, throughput, and delay) via both real experiments and simulations.
- *Visualization:* TeaCP provides visualization of packet routes, network topology and other statistics, that can be used for understanding, analyzing and diagnosing the behavior of collection protocols.

The rest of this paper is organized as follows. In Section II, we describe the design and implementation of TeaCP. Thereafter, in Section III we demonstrate how TeaCP is used to analyze, compare, and troubleshoot collection protocols. Finally Section IV concludes the paper.

II. TeaCP Implementation

In this section, we present the design and implementation of our TeaCP toolkit. We then highlight the convenience of using TeaCP to test different data collection protocols.

A. TeaCP Structure

The structure of TeaCP is as follows. Configurable test parameters include packet generation interval, radio power and radio channel of the network. Radio power and channel settings are applied to the radio component of hardware. The packet generation interval is applied to a periodic timer, whose firing signal invokes the packet generator to generate a new packet, drive the log generator to output a log message, and transfer the packet to the network layer, which moves it to the root using a collection protocol. Since the collection protocol may use multihop routing to accomplish the delivery, the packet will be received by intermediate nodes and the root. No matter whether the node is a client or the root, the network layer would notify the packet reception handler when it receives a packet from other sensors. The packet reception handler prepares the information needed by the log generator, which outputs log message to storage through serial communication. Finally, as described before, post-experiment analysis is done based on data saved in storage.

The only interactions between the application layer and the network layer are two necessary processes: (1) the packet generator transfers the packet to the network layer; (2) when a packet is received, the network layer notifies the packet reception handler on the application layer. The two processes correspond to two types of network event: (1) packet arrival at the source node; (2) packet arrival at the destination node or an intermediate node. Log messages related to these events are sufficient for computation of packet routes and other statistics.

B. Message formats

A log message is generated whenever a network event happens[2]. Hence, for each event, there is an associated packet and a recording log message.

As shown in Fig 1, the log message contains 7 fields: `type`, `cur_node_id`, `src_node_id`, `last_node_id`, `packet_id`, `time` and `hop_count`. The field `type` denotes whether this log message is generated at packet generation or packet reception, and `cur_node_id` is the ID of the current node where the log message is generated. The field `time` records the time when the event happened and `src_node_id` is the ID of source node that generates the packet. The field `packet_id` equals to the value of packet counter at the source node when it is generated. The field `last_node_id` denotes the node ID of the packet's last hop. If the log message is generated upon packet reception, `last_node_id` is the ID of the node that sends the packet to the current node. If the log message is upon packet generation, then `last_node_id` is not important and set to be `src_node_id`. The field `hop_count` denotes the number of hops that the packet has experienced.

The data message shown in Fig. 2 has four fields: `src_node_id`, `last_node_id`, `packet_id` and `hop_count`, which have the same meanings with the log message fields.

TeaCP manipulates data messages and generates log messages as follows:

[2]In the nesC code, packet arrival at source node, the root and intermediate nodes correspond to `command Send.send()`, `event Receive.receive()` and `event Intercept.forward()`, respectively.

- When a sensor node generates a packet, it sets the data message's `src_node_id` to its `TOS_NODE_ID`. `last_node_id` is set to its ID as well. `packet_id` is set to the node-unique counter of how many packets have been sent. `hop_count` is initialized to 0. All the fields of the data message are copied to corresponding fields of log message.
- When a sensor node receives another sensor node's packet, first the data message's field values are copied to the generated log message. Then the data message's `last_node_id` is updated to the current node ID and `hop_count` is incremented.
- When the root node receives a packet, it increments the packet's `hop_count` and outputs the associating log message.

C. Post-experiment analysis

Post-experiment analysis functions of TeaCP include visualization of packet routes and network evolution over time. TeaCP also evaluates reliability, throughput and delay performance of the network. Here we provide details of these functionalities.

For a packet with specific source node ID and packet ID, TeaCP calculates the route of this packet based on collected data. `last_node_id` and `cur_node_id` of an event related to the packet represents a directed edge on the packet route. If the packet has only one copy in the network, then TeaCP links these directed edges together and generates the packet route:

```
routes[src_node_id][packet_id]= [ID of hop 0,
ID of hop 1, ID of hop 2, ...]
```

If the packet has duplicates that go through different routes, TeaCP detects this scenario and generates all the route branches:

```
routes[src_node_id][packet_id]= {
duplicate1:[ID of hop 0, ID of hop 1, ...],
duplicate2:[ID of hop 0, ID of hop 1, ...],
...}
```

To visualize the network topology, first all packets in the network are divided into equal-size windows according to their packet IDs. In a window, based on the routes of the packets, TeaCP calculates the number of packets that traverse along each edge and presents a directed graph representing the network topology. For each window of packets, TeaCP generates a network topology with edge weights. Then the generated time-windowed topologies are combined together and converted into a movie to demonstrate the topology evolution over time. We will show some sample graphs of the visual topology in Section III.

For evaluation purposes, we use the most widely used performance metrics, namely delivery rate, throughput, goodput, and delay.

D. TeaCP working with CTP and BCP

CTP and BCP are two main collection protocols implemented in TinyOS. Here we explain how TeaCP works with these two protocols. Some protocols are developed as variants of CTP and BCP so they can be directly used with TeaCP.

Our only requirements for collection protocol implementations are: (1) the collection protocol should provide function for the TeaCP to inject a packet into the network layer; (2) the collection protocol should notify TeaCP when the network layer receives a packet. In the nesC code, TeaCP is implemented as a module named `TestBenchC`, which uses the interfaces `Send`, `Receive` and `Intercept`. Commands and events of these interfaces (refer to [10] for nesC basics) are defined in TinyOS. The interface `Send` is used for injecting packets into the network layer. The interfaces `Receive` and `Intercept` signal events to the interface user when a packet is received. The difference between the two is that `Receive` is specified for packet reception by the destination while `Intercept` is for packet reception by intermediate nodes. Since TeaCP uses `Send`, `Receive` and `Intercept`, collection protocol implementations are required to provide these three interfaces if they wish to interact with our toolkit.

The CTP implementation satisfies the requirements. Hence TeaCP could be used directly for CTP. The BCP implementation, however, only provides interfaces `Send` and `Receive`. Thus to use TeaCP with BCP, we added the interface `Intercept` in some components and signaling of event `Intercept.forward(message_t *msg)` in the forwarding engine to BCP. Another difference of BCP implementation from CTP is that it uses interface `BcpDebugIF` for protocol debugging and then in `TestBenchC` we just provided the interface `BcpDebugIF` and implemented its commands with empty functions.

After these changes, TeaCP can test, analyze and evaluate BCP. In total, we modified about 13 lines of code in order to adapt TeaCP to the BCP implementation (around 2500 lines of code). The adaptation of TeaCP for BCP also suggests that for other protocols, only small amount of code writing is needed to make TeaCP work.

III. Performance Evaluation

This section illustrates functionalities of TeaCP, through evaluation and analysis of CTP and BCP which serve as representatives of two different categories of protocols.

A. TeaCP evaluation of intra-car collection protocol on TOSSIM

TeaCP can evaluate collection protocols through simulations, which is convenient and fast. For the simulations on TOSSIM, we used real RSSI traces of intra-car wireless sensor network. The RSSI traces were measured in real intra-car experiments, recording the RSSI (in dBm) between different motes at different time. The network consists of 15 nodes, in which the root is on the driver seat, three sensors are placed in the engine compartment, four sensors are respectively attached to the four wheels, three sensors are placed on passenger seats and the rest placed on the chassis. In the simulation, these sensors periodically generate packets and forward them towards the root. The sensor model used in the simulation is MICAz. After running on TOSSIM, TeaCP outputs the network performance statistics such as delivery rate, throughput/goodput, and latency.

Figures 3, 4 and 5 respectively compare the delivery rate, average delay, and throughput/goodput performance of CTP and BCP (both FIFO and LIFO implementations) based on the real RSSI traces. From these simulation results, we observe that CTP outperforms BCP-FIFO in terms of delay, but BCP

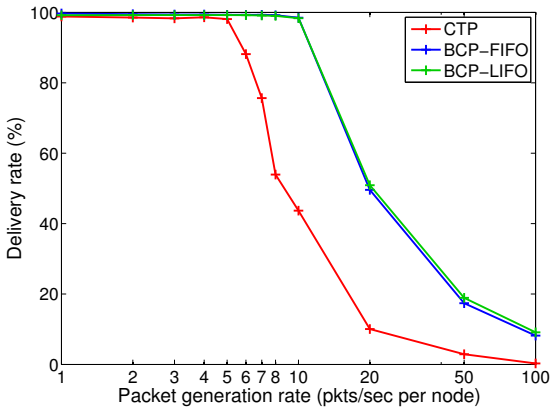

Fig. 3. CTP and BCP delivery rate vs. packet generation rate.

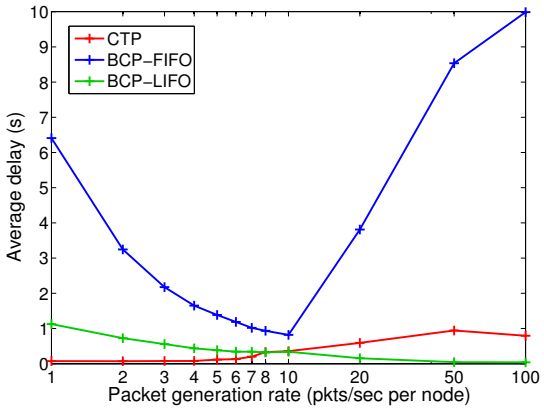

Fig. 4. CTP and BCP average delay vs. packet generation rate.

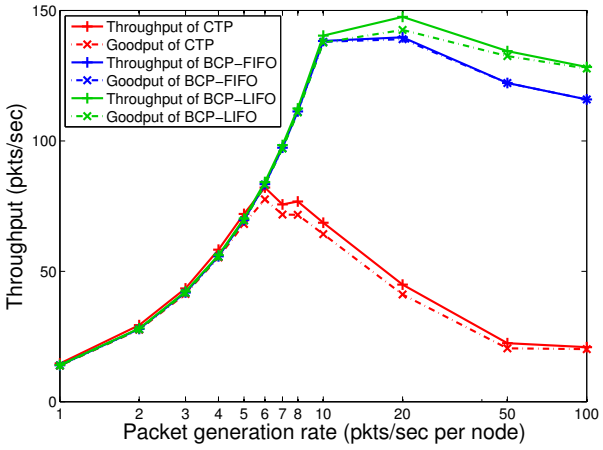

Fig. 5. CTP and BCP throughput/goodput vs. packet generation rate.

improves the throughput/goodput and delivery rate, especially under high load conditions. Furthermore, BCP-LIFO achieves much better (lower) delay performance than BCP-FIFO. We also note that the difference between throughput and goodput is negligible for these protocols. The root receives a duplicate packet when the root has already received the packet but the

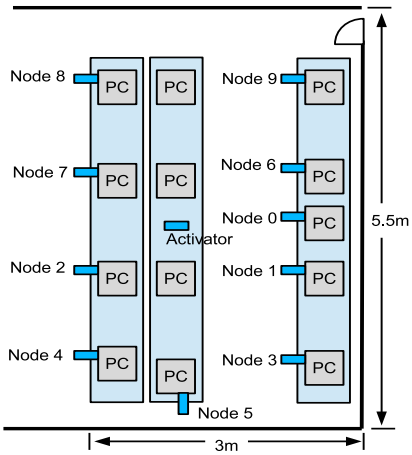

Fig. 6. Placement of the 9 sensor nodes, the root (Node 0) and the activator in the experiment.

sender has not successfully received the acknowledgement. The difference between throughput and goodput is small because duplicates are mostly removed due to the duplicate suppression mechanism in the protocols and only duplicates at the last hop are considered in throughput.

These results obtained by TeaCP depict a wholistic picture for CTP and BCP, from which one can compare these two protocols and analyze their advantages and disadvantages. For TOSSIM simulation (running one out of 8 cores of Intel Core i7-2600 CPU@3.40GHz) of one root and 39 sensor motes, with each sensor mote generating 5000 packets at rate of 1 pkts/sec, the total simulation time is around 10 minutes.

B. TeaCP performance in real experiments

TeaCP provides other analysis functions for visualization of the topology evolution over time, per-node statistics, and complete characterization of network edges and packet routes. The outputs of this toolkit include detailed statistics for each node showing latency histogram, packet loss rate, and route branches due to packet duplicates. Additionally, time-sliced network topology and topology evolution over time can be displayed.

In our experiments, we set up a network consisting of 9 sensor nodes periodically generating packets and forwarding them to a single root (node 0). The sensor motes and the root are IEEE 802.15.4-based Tmote Sky working on 2.4 GHz frequency band. The transmission power is configured to -20 dBm for all experiments and the packet generation rate is set to be either 2 or 4 pkts/sec per node. Our devices are located in a computer lab and the positions of the motes are shown in Fig. 6. The motes are connected to PCs through a USB port for the purposes of logging messages and statistics. In the testing environment, there exists WiFi interference and some foot traffic, which may change during a test. Each test is initialized by a central activator node sending a broadcast message containing test configurations. After the initialization step, each experiment runs for 20 to 30 minutes. Our current experimental setup (e.g., small scale network compared with real-world applications) can be viewed as a proof of TeaCP functionality concept in experimental domain for evaluation and diagnosis purposes.

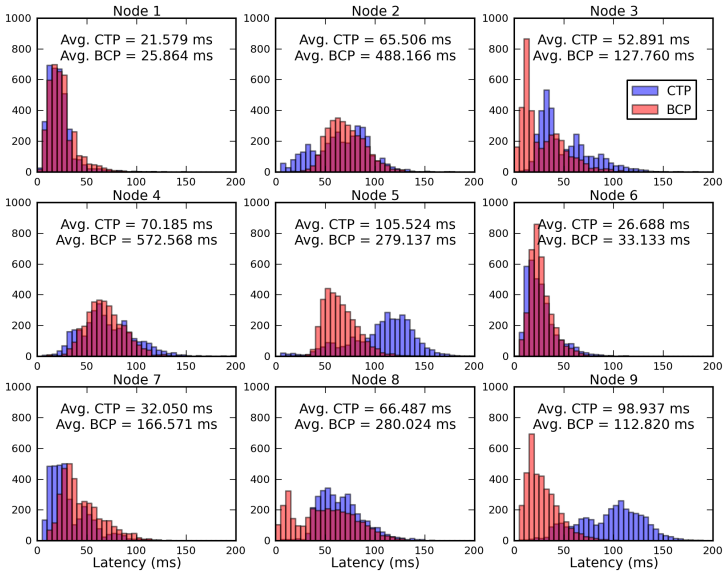

Fig. 7. Latency histograms by node for CTP and BCP. Though the delay distribution of BCP and CTP are similar, average delay of BCP is much higher than CTP.

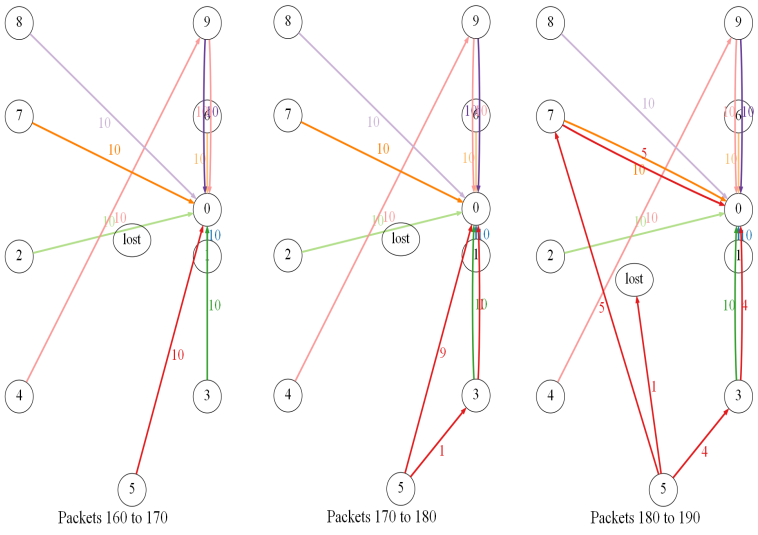

Fig. 8. Topology sample of CTP at 10 packets per image. The topologies are drawn based on packets with ID from 160 to 190 from every source node.

1) *Latency histograms*

CTP and BCP experimental delay performance are shown in Fig. 7. The per-node histograms give detailed statistics of latency of packets generated in the network. Overall, the histograms of two protocols look quite similar. The high average packet delay performance of BCP can be explained by the fact that a few packets are experiencing huge delays, as high as 30 seconds. These results suggest that the average packet delay of BCP could be significantly reduced if this issue were resolved.

2) *Network topology evolution*

TeaCP provides time-sliced topologies and network evolution over time. Fig. 8 shows CTP topology evolution over time with a window size of 10 packets per each caption. In the first caption, with packets ID 160 to 170, node 5 transmits packets directly to the root. In the second caption, node 5 starts to choose node 3 as a relay node. In the last caption, node 5 transmits 5 packets to node 7 among the ten packets it generates. The network topology transition indicates change of quality of links, probably due to walking of testers during the experiment. This showcases that TeaCP can be used to investigate the behavior of collection protocols under dynamic scenarios and their ability to adapt the routes according to environmental conditions.

The topology information can also be potentially used for the selection of the root node. For instance, in one of our tests of CTP (not shown, due to space limitation), we observed that one of sensor nodes carries a large portion of the traffic. In this scenario, it may be worth testing what would happen if this node were to fail or go offline and perform power consumption analysis to determine the battery lifespan of such a high-traffic node.

IV. Conclusion

We have developed TeaCP, a toolkit that focuses on visualization and analysis of network layer collection protocol performance. We showed how our toolkit may be utilized to compare the performance of two dominant protocols, CTP and BCP, though it is also designed for simple evaluation and analysis of new collection protocols following a typical mold. With some straightforward modifications, TeaCP can be also adapted to multiple-root collection protocols and any-to-any routing protocols, and we leave these as future directions.

Acknowledgements

This work was supported in part by NSF under grant CCF-0916892 and by a grant from General Motors.

References

[1] http://greenorbs.org/.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[3] M. H. Alizai, O. Landsiedel, J. A. B. Link, S. Götz, and K. Wehrle. Bursty traffic over bursty links. In *SenSys*, 2009.

[4] M. Alresaini, M. Sathiamoorthy, B. Krishnamachari, and M. J. Neely. Backpressure with Adaptive Redundancy (BWAR). In *IEEE INFOCOM*, 2012.

[5] C. Buschmann, D. Pfisterer, S. Fischer, S. P. Fekete, and A. Kröller. Spyglass: a wireless sensor network visualizer. *SIGBED Rev.*, 2(1):1–6, Jan. 2005.

[6] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. The collection tree protocol (TEP 123). http://www.tinyos.net/tinyos-2.x/doc/.

[7] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys*, 2009.

[8] M. Hashemi, W. Si, M. Laifenfeld, D. Starobinski, and A. Trachtenberg. Intra-car Wireless Sensors Data Aggregation: A Multi-hop Approach. In *VTC*, 2013.

[9] R. Jurdak, A. G. Ruzzelli, A. Barbirato, and S. Boivineau. Octopus: monitoring, visualization, and control of sensor networks. *Wirel. Commun. Mob. Comput.*, 11(8):1073–1091, Aug. 2011.

[10] P. Levis and D. Gay. *TinyOS Programming*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[11] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys*, 2003.

[12] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali. Routing without routes: the backpressure collection protocol. In *IPSN*, 2010.

[13] M. Turon. MOTE-VIEW: A Sensor Network Monitoring and Management Tool. In *EmNetS-II*, 2005.