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

Wei Si, Morteza Hashemi, Liangxiao Xin, David Starobinski, and Ari Trachtenberg

***Abstract*—We present TeaCP, a prototype toolkit for the evaluation and analysis of collection protocols in both simulation and experimental environments running on TinyOS. Our toolkit consists of a testing system, which runs a collection protocol of choice, and an optional SD card-based logging system, which stores the logs generated by the testing system. The SD card datalogger allows a wireless sensor network (WSN) to be deployed flexibly in various environments, especially where wired transfer of data is difficult. Using the saved logs, TeaCP evaluates a wide range of performance metrics, such as reliability, throughput, and delay. TeaCP further allows visualization of packet routes and the topology evolution of the network, under both static and dynamic conditions, even in the face of transient disconnections. 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). We also present the usage of TeaCP as a high level diagnosis tool, through which an inconsistency of the BCP implementation for the CC2420 radio chips is identified and resolved.**

***Index Terms*—Network and systems monitoring, wireless sensor networks, collection protocol, performance evaluation and visualization, opensource toolkit.**

I. Introduction

Data collection is intrinsic to numerous applications in wireless sensor networks (WSN), ranging from intra-car monitoring [2] to environmental data gathering [3] and military surveillance [4]. 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, *collection protocols* have 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 delay 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, home monitoring applications may emphasize robustness to wireless interferences (WiFi, Bluetooth, etc.), while military applications may also strive for low packet delay. 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 under 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 conditions.

Many collection protocols have been proposed in the literature [5–9] and several of them have also been implemented. However, a common publicly available platform to visualize, analyze and compare their performance is needed. Such a toolkit should not only enable the evaluation of multiple collection protocols, but also help to visualize the behavior of the collection protocols.

Though there already exist several visualization tools for WSN [10–13], most of them concentrate on controlling, monitoring and displaying sensor data rather than on analyzing the underlying network protocol. As explained in Section II, some tools 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, as discussed in the sequel, existing evaluations of collection protocols 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 runs on TinyOS and TOSSIM, and assumes that the PHY and MAC layers follow the IEEE 802.15.4 standard. 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. 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. More specifically, TeaCP incorporates an optional *SD card datalogger*, which provides data storage space. The SD card datalogger also enables us to evaluate collection protocols in environments where wired transfer of logs is difficult (e.g., intra-car WSNs). For simulations, TeaCP provides the convenience of testing the performance of collection protocols over a wide range of conditions. The post-experiment analysis

W. Si, M. Hashemi, L. Xin, D. Starobinski, and A. Trachtenberg are with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 USA (e-mail: weisi@bu.edu; mhashemi@bu.edu; xlx@bu.edu; staro@bu.edu; trachten@bu.edu).

An earlier and shorter version of this paper appeared in the proceedings of the IEEE COMCAS 2013 Conference [1].

[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.

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 collection protocols over time.

Since TeaCP is built at the application layer, a system architect that has some protocol options at hand can utilize this toolkit to test these protocols and select the one that is most suitable. TeaCP is also potentially useful for protocol developers to validate accurate protocol operations based on the obtained testing statistics. We stress, however, that the main benefit of TeaCP lies in its generality, as it enables evaluation and comparison between various collection protocols based on high-level performance metrics. Instructions for using and downloading the TeaCP toolkit can be found at http://nislab.bu.edu/?page_id=355.

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

- *Generality:* TeaCP is a general toolkit designed to 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.
- *Flexibility:* The SD card datalogger provides the convenience to test the collection protocol in various environments.

The rest of this paper is organized as follows. In Section II we present the related work on visualization tools for WSN and data collection protocols. Thereafter, in Section III, we describe the design and implementation of TeaCP. We also present our implementation of an SD card datalogger on the TelosB mote, a widely used wireless sensor model in the research community. In Section IV, we demonstrate how TeaCP is used to analyze, compare, and troubleshoot collection protocols. Finally, Section V discusses potential areas for extension and Section VI concludes the paper.

II. Related Work

A. Visualization for WSN

Many of the existing tools for sensor networks focus on logging and visualizing sensor network readings, rather than on analyzing the performance of the underlying network layer. Tools such as SpyGlass [10] and Octopus [11] help to visualize the topology of multi-hop networks, but typically limit their link quality data to a binary good/bad flag depending on the timestamp and frequency of the last packet received. Compared with these tools, TeaCP provides a much more detailed evaluation of the collection protocols' performance. In particular, Octopus only provides inter-arrival times of packets at the root while TeaCP provides precise measurements of end-to-end delays as well as delays on individual links. TeaCP can also be used to evaluate the performance of a data collection protocol on TOSSIM over a wide range of network conditions.

As a plugin to TOSSIM, TinyViz [14] can visualize sensor network and display TOSSIM events. OMNeT++ [15] also provides comprehensive visualization functions for evaluation and analysis of network protocols in WSN. TeaCP differs from TinyViz and OMNeT++ in that TeaCP incorporates evaluation and analysis of high-level performance metrics for *both* real experiments and simulation.

Other tools, such as Mote-View [13], have the capabilities of monitoring the underlying network layer, such as providing statistics on link quality. These tools are not designed for benchmarking and analyzing the underlying network layer, because they rely on that same network layer to reliably send statistics to the root. Trying to send packet loss information on the same network link that is experiencing packet loss can create a feedback loop, thus causing more packet loss that needs to be reported. Another scenario where these visualization tools might fail is when a subnetwork gets disconnected from the rest of the network. In this situation, visualization tools relying on the underlying network might not be able to provide information about the subnetwork because of their inability to fetch link information.

Existing testbeds such as Tutornet [16], MoteLab [17] and the Guildford Facility of the SmartSantander project [18] have provided the ability to log information of sensor motes through out-of-band communication. Leveraging the same idea, TeaCP logs network events at each mote to storage through serial communication, which does not rely on the tested collection protocol. Isolating the logging system from the network layer enables TeaCP to concentrate on accurately reporting network layer performance. Besides, TeaCP enables one to deploy a WSN testbed on his own targeting environment.

Packet sniffers [19] are an alternative platform that has been widely used to analyze network performance. Sniffers suffer from several limitations in our context, however. First, a sniffer may not be able to extract packet information when two or more nodes transmit packets simultaneously or when the wireless channel experiences deep fading. Second, a packet sniffer can only provide timing information related to events occurring on the wireless channel, such as the start of a packet transmission. However, it may not be able to measure events occurring at higher levels, such as the time elapsing from the generation of the packet by the application layer till its transmission, which is necessary for computing end-to-end delay. Third, in the scenario of large networks, several sniffers are needed to cover all the sensors in the network.

TeaCP provides a reliable logging system by attaching cheap and lightweight data storage to each sensor. It provides accurate delay analysis by recording the time of packet generation and packet delivery at the application layer. We emphasize, nevertheless, that sniffers can still serve as useful, complementary platforms to record network events.

B. Data collection protocols

The Collection Tree Protocol (CTP) [7, 8] is the *de facto* standard for data aggregation and has been supported in TinyOS 1.x and 2.x. Due to its prevalence, we use it to demonstrate the usefulness of our proposed toolkit. Although the TinyOS implementation of CTP itself has provided a logging layer to report events such as packet reception, parent change, etc., such a logging system is more specifically designed for debugging the protocol. TeaCP differs from the CTP logging implementation in that it can evaluate additional metrics, such as reliability and delay performance.

The Backpressure Collection Protocol (BCP) [9] is another data collection protocol that has recently gained interest, as it is the first implementation of a backpressure-based routing algorithm in the context of sensor networks. BCP has been shown to outperform CTP in terms of robustness and throughput, two important metrics for collection protocols, although it has been observed that BCP suffers from higher packet delay than CTP. We thus use BCP as another testing protocol to verify the functionality and usefulness of our toolkit.

Several newer collection protocols are built upon CTP and BCP. For instance, [5] proposes a new routing scheme based on CTP, called Bursty Routing Extensions (BRE). Elsewhere, Backpressure with Adaptive Redundancy (BWAR) [6] has been proposed to improve the delay performance of BCP through the injection of redundant packets into the network.

The IPv6 routing protocol for low power and lossy networks (RPL) [20], whose design was informed by CTP, can also be used for data collection by having the nodes in the Destination Oriented Directed Acyclic Graph (DODAG) send packets to the DODAG root.

The development of these data collection protocols motivates us to design a general open-source toolkit to analyze and compare their performance. Although many of the aforementioned protocols have their own experiment evaluation software, our toolkit TeaCP provides a more comprehensive set of evaluation functions on delay and reliability performance. For example, the BCP implementation calculates packet delay by accumulating MAC delays and queuing delays. Queuing delays are measured through local timers while the MAC delay is approximated by a constant 10 ms. TeaCP, on the other hand, calculates the packet delivery delay by recording generation time and delivery time of each packet and does not resort to approximations. Our toolkit also helps to visualize the route of the packet in the network and the evolution of the network topology. This feature helps to understand how a collection protocol behaves in response to link and traffic variations. We believe that TeaCP can not only be used for evaluating different protocols but also through visualization and analysis to provide insight for improving existing protocols.

A previous and abbreviated version of this work appears in [1]. The current paper differs in several aspects. First, it introduces the design and implementation of the SD card datalogger and reports several new experimental results. Second, the TeaCP toolkit is described in much greater detail. Finally, the paper includes detailed discussion of related work and implementation issues.

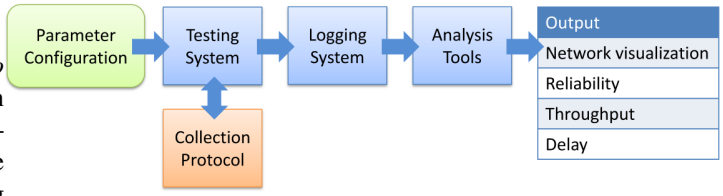


Fig. 1. General structure of an evaluation toolkit. Note that the logging system does not use the collection protocol to send log messages to avoid perturbing its behavior.

III. TeaCP Implementation

In this section, we present the design and implementation of our TeaCP toolkit. First, we define the requirements for designing a toolkit for our purposes, i.e., to visually and quantitatively analyze a collection protocol under various conditions. Thereafter, the implementation of TeaCP is described based on the goal of fulfilling these requirements. Meanwhile, we also describe the implementation of the SD card datalogger on the TelosB mote. Finally, we highlight the convenience of using TeaCP to test different data collection protocols.

A. Design objective and requirements

The development of various collection protocols necessitates a common toolkit that can visually analyze protocol behavior and evaluate performance of these protocols including reliability and delay. Due to needs of different applications, the protocol evaluation and analysis should be carried out under various network conditions such as light traffic and heavy traffic. The general structure of such a toolkit is illustrated in Fig. 1.

The toolkit mainly consists of the testing system, the logging system and the analysis tools. The testing system depends on an underlying collection protocol to deliver packets from sensor nodes to a root, while generating log information about what is happening in the network. The logging system stores log outputs from the testing system for post-experiment analysis, which is accomplished by the analysis tools.

In a typical test case, parameter configurations are first fed to the testing system. Then sensor nodes in the testing system start to generate packets and transfer them to the underlying collection protocol, which is responsible for delivering the packets to the root. At the same time, the testing system outputs log to the logging system whenever an event happens in the network. After the testing is done, the analysis tools process the log files stored in the logging system and generate network visualization and other statistics.

A key difference between this structure and existing visualization tools is that no interface is required between the logging system and the collection protocol, i.e., the logging system does not use the collection protocol to send log messages to the root for post-experiment analysis. Analysis results of the latter might be inaccurate because injected feedback information can perturb the ongoing test, especially under stress conditions (e.g., close to link capacity).

Next, we define the requirements for the testing system and logging systems of the toolkit to accomplish desired functionalities.

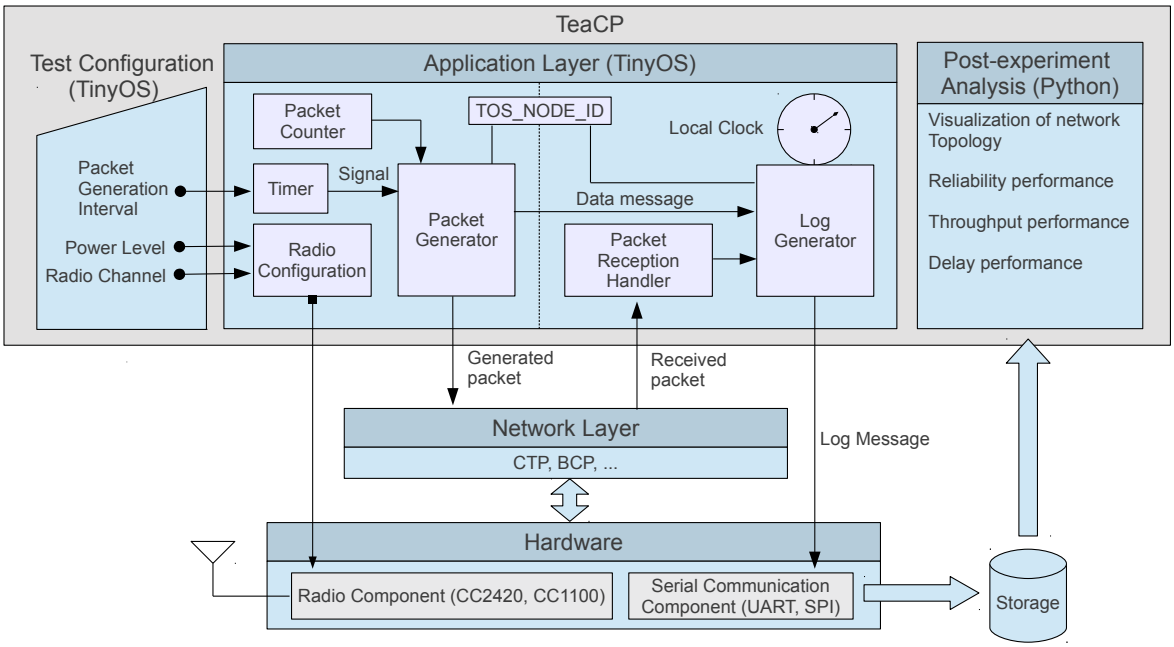


Fig. 2. Structure of TeaCP. TeaCP is built on the application layer, and consists of the test configuration component, the application layer component for generating and handling packets, and the post-experiment analysis component. The test configuration component and the application layer component run on sensor motes. The post-experiment analysis component runs on a PC.

Testing system requirements

We desire that any data collection protocol can be tested on the platform with minimal code reading and modifications, though the interfaces between the testing system and the collection protocol may differ among existing protocol implementations. For example, the standard BCP implementation in TinyOS provides an interface `BcpPacket` for the application layer to extract packet-related information such as the packet delay and the number of transmissions the packet has experienced. In contrast, CTP's provided interface `CtpPacket` does *not* give delay information or the number of transmissions. Thus a toolkit fetching packet delay via `BcpPacket` cannot be directly used with the CTP implementation. Therefore we choose to minimize the set of interactions between the testing system and the collection protocol for achieving the toolkit functionality.

Logging system requirements

The testing system communicates messages to the logging system. Since sensor motes are limited in computation and communication resources, it is desirable to avoid introducing too much overhead to the testing system. Hence a key challenge in building such a toolkit is to design an efficient log message format through which the desired statistics can be analyzed.

A log message should be generated whenever a network event occurs. We are specifically interested in the arrival of a packet at a node (source node, destination node, and intermediate node). When a packet arrival event happens, the log message should contain the identification of the packet (*who*), identification of the node (*where*) and time of this event (*when*). With these event-related information, it should be possible to uniquely identify and trace the route of a packet. The toolkit should also be able to detect route branches of duplicates when they arise in the network.

As for post-experiment analysis, it turns out that the records of arrival events are sufficient for calculating statistics such as delivery rate, throughput and packet delays; the details will be provided later in this section.

B. TeaCP Structure

TeaCP is designed to meet the requirements previously defined. The structure of TeaCP is shown in Fig. 2. TeaCP operates at the application layer, and consists of the test configuration component, the application layer component for generating and handling packets, and the post-experiment analysis component. The test configuration component and the application layer component run on sensor motes. The post-experiment analysis component runs on a PC.

The configurable test parameters include the packet generation interval, the radio power and the channel for the network. The radio power and channel settings are applied to the radio component of the hardware. The packet generation interval is applied to a periodic timer. The firing signal of the timer (i) increments the packet counter by one, (ii) invokes the packet generator to generate a data message (encapsulated in a packet) with packet ID equal to the packet counter, (iii) drives the log generator to output a log message, and (iv) transfers the packet to the network layer, which moves it towards the root using a collection protocol. Since the collection protocol may use multihop routing to accomplish the delivery, the packet may be received by intermediate nodes and the root. No matter whether the node is a relay or the root, the network layer will 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

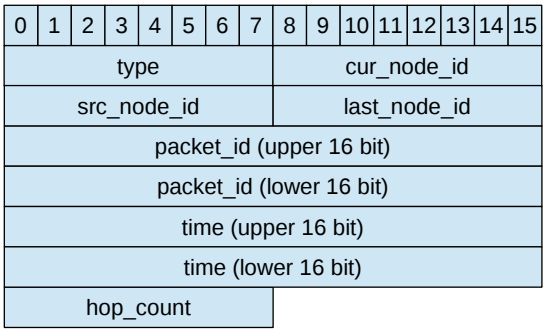


Fig. 3. Format of log message. A log message is generated whenever a network event (i.e., packet generation and reception) happens. The fields `cur_node_id` and `time` denote the location and the time of the event, respectively.

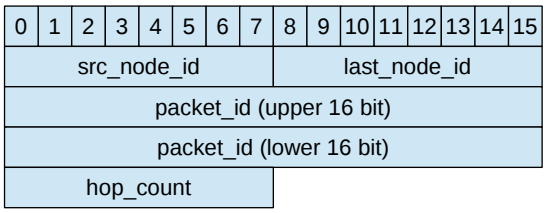


Fig. 4. Format of data message. Data messages are generated by the packet generator and carried as payload of packets by the collection protocols. Each data message is identified by the fields `src_node_id` and `packet_id`.

message to storage through serial communication. Finally, as described before, post-experiment analysis is done based on the data saved in storage (e.g., the SD card datalogger).

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

A TinyOS-based collection protocol can interact with TeaCP as long as it provides the following three interfaces:

- A `Send` interface, that is, the collection protocol can receive a packet from the application layer component;
- A `Receive` interface, that is, the collection protocol can notify the application layer component when receiving a packet if the current node is the root node;
- An `Intercept` interface, that is, the collection protocol can notify the application layer component when receiving a packet if the current node is an intermediate node.

Any collection protocol should include `Send` and `Receive` interfaces in order to interact with applications. However, some collection protocols may not natively provide an `Intercept` interface. As shown in Section III-F, adding such an interface usually entails adding only a few lines of code.

The configurable parameters in the test configuration component are as follows:

- *Packet generation rate* – the number of packets a sensor node generates per second. It is reciprocal of the packet generation interval, which is used to set the firing period of timer.
- *Radio power* – the transmission power when a node transmits a packet or a beacon message. For the CC2420 chip, radio power ranges from -25 dBm to 0 dBm.
- *802.15.4 radio channel* – the radio channel on which the collection network operates. The 2.4GHz ISM band is divided into 16 non-overlapping ZigBee channels, with channel ID from 11 to 26.

Step 1	Compile the code of the collection protocol and of the TeaCP application layer component and install them onto the nodes.
Step 2	Attach a datalogger to each node and deploy the nodes in the testing environment.
Step 3	Compile and install the TeaCP test configuration component onto the activator node.
Step 4	Press the user button on the activator node to send a broadcast signal and start the experiment.
Step 5	When the experiment ends, collect the data from the dataloggers.
Step 6	Use the TeaCP post-experiment analysis component to analyze the collected logs and obtain results.

TABLE I
USING TEACP IN EXPERIMENTS.

Step 1	Compile the code of the TeaCP test configuration and application layer components.
Step 2	Configure the network in TOSSIM, e.g., number of nodes, link gains and noise traces.
Step 3	Run the TOSSIM simulation, which outputs the logs into a log file.
Step 4	Use the TeaCP post-experiment analysis component to analyze the log file and obtain results.

TABLE II
USING TEACP IN SIMULATIONS.

The purpose of the packet generation rate is to allow for testing collection protocols under different network traffic scenarios. Tests of different radio power could compare performance of a low power configuration with a high power configuration. Since power consumption is a natural concern in WSN, low power is always preferred if it could satisfy application requirements. Flexibility in configuring the radio channel permits us to move the network between a low-interference channel and a channel with intensive WiFi signals for interference tests.

Table I and II describe the steps to follow for using TeaCP in experiments and simulations, respectively.

C. Message formats

This section describes the formats of the log message and the data message. We will also detail how to determine the fields of the messages.

A log message is generated whenever a network event hap-

pens.[2] Hence, for each event, there is an associated packet and a recorded log message. As shown in Fig. 3, 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 that generates the log message. The field `src_node_id` is the ID of source node that generates the packet. 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 meaningful and set to `src_node_id`. The field `packet_id` equals the value of the packet counter at the source node when it is generated. The field `time` records the time when the event happened. The field `hop_count` denotes the number of hops that the packet has experienced.

The fields `src_node_id` and `packet_id` are used to identify the packet associated with the log message and the event. The `packet_id` is different from the packet sequence number in the network layer; it is obtained from the 32-bit packet counter when the packet is generated. One reason that we do not use the network layer sequence number is that a 16-bit sequence number may be insufficient in the scenario of long-duration tests. Another reason is that some protocols may not provide sequence number for packet identification.

The fields `cur_node_id` and `time` represent the location and the time of the event. In addition to them, `last_node_id` is also important in determining packet routes. The following example shows that in the case of packet duplicates, packet routes might not be identifiable based on `cur_node_id` and `time` only. Suppose packet A has the route $1 \rightarrow 3 \rightarrow 5$. The arrival times at the three nodes are time 0, 1, 2. Suppose node 1 also generates a duplicate A'. The duplicate follows the route $1 \rightarrow 2 \rightarrow 4$ and the arrival times at node 2 and 4 are 0.5 and 1.5, respectively. Because we are not able to detect duplicates based on the locations and times, we can only get a single route $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, which is incorrect. Given `last_node_id` we are able to draw a directed edge for each event related to the packet on the network graph. Then by collecting these directed edges, TeaCP will be able to find out all the routes of duplicates.

The fields of the log message are determined as follows. The values of `type`, `cur_node_id` and `time` are set directly by the node generating the log message: `type` can be trivially determined because the node knows whether this is packet generation or packet reception; `cur_node_id` is set to the node's `TOS_NOD_ID`, its unique node ID in the network; `time` is obtained from the local clock. Value of other fields (`src_node_id`, `packet_id`, `last_node_id`, `hop_count`) need to be obtained from data messages.

Data messages are generated by the source nodes and

[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.

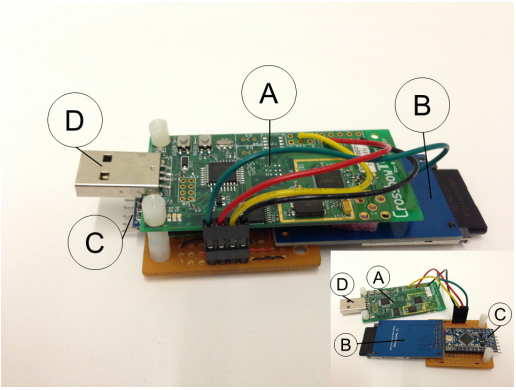


Fig. 5. TelosB with SD card datalogger: A. TelosB; B. SD card slot; C. Arduino; D. Power adaptor (USB).

are carried as payload of packets by collection protocols. As shown in Fig. 4, the data message has four fields: `src_node_id`, `last_node_id`, `packet_id` and `hop_count`. Similar to the log message format, `src_node_id` denotes the ID of the source node, `last_node_id` denotes the node ID of last hop on the path, `packet_id` is the packet ID and `hop_count` records the number of hops that the packet has experienced.

TeaCP manipulates data messages and generates log messages as follows:

- When a sensor node generates a packet, it sets the data message's `src_node_id` to its `TOS_NODE_ID` and `last_node_id` is set to its ID as well. `packet_id` is set to the node-unique packet counter. `hop_count` is initialized to 0. Then all the fields of the data message are copied to the corresponding fields of the generated log message.
- When a sensor node receives a packet from another sensor node, first the fields of the data message are copied to the generated log message. Then the data message's `last_node_id` is updated to the current node ID and the `hop_count` is incremented.
- When the root node receives a packet, it increments the `hop_count` of the data message and outputs the associated log message.

D. SD card datalogger

In TeaCP, the log messages can be saved by an SD card datalogger. This section describes the implementation of the SD card datalogger for the TelosB mote.

In embedded systems, the SD card is typically written and read by the CPU through a Serial Peripheral Interface (SPI), which is not included in the expansion port of the TelosB board. There are several ways to enable the TelosB mote to communicate with an SD card. One way is to emulate a software SPI port using four I/O pins of the TelosB mote [21]. However, significant CPU resources may be consumed by the SD card command interface, especially when there is a large amount of data that needs to be stored by the SD card. An alternative way is to resort to an Arduino board, which can serve as an SPI adapter between the TelosB mote and the

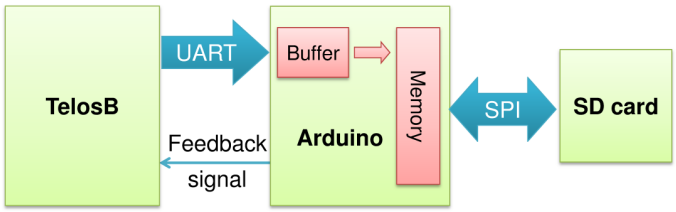


Fig. 6. Hardware schematic of the SD card datalogger. The log messages are first sent by the TelosB mote to the Arduino board via the UART interface, and then the Arduino board writes log messages into the SD card via the SPI.

SD card [22]. The log messages are first sent by the TelosB mote to the Arduino board via a Universal Asynchronous Receiver/Transmitter (UART) interface, and then the Arduino board writes log messages into the SD card via the SPI. Our SD card datalogger leverages the second idea.

In order to transfer the log messages from the TelosB mote to an SD card, we use the Arduino Pro Mini board (5V, ATmega328 CPU@15MHz) as an SPI adapter for the TelosB mote. Arduino Pro Mini is only one third of the size of TelosB and costs 10 dollars. Besides, the integrated development environment (IDE) software of Arduino Pro Mini includes a UART communication library and an SD card command interface library that supports SD card initialization and read/write modules.

The hardware architecture of the TelosB mote with the SD card datalogger is shown in Fig. 5 and 6. When the TelosB mote generates log messages, the messages are transmitted to the Arduino board through UART and shifted into the buffer of the Arduino UART receiver. Then Arduino moves the data from the UART receiver buffer into the memory and send the data to the SD card through SPI. The UART receiver buffer can hold up to 64 bytes while the size of a log message is 25 bytes (including headers). Thus, the buffer can store at most two log messages. If there are three log messages arriving within a short period of time, it is possible that the UART receiver buffer of Arduino will overflow. To avoid the overflow issue, we use a binary feedback signal from the Arduino board to indicate whether it is busy writing data to the SD card.

SD card is a block-addressable storage device with block capacity of 512 bytes, where a block is the minimum quota of write in an SD card [23]. The 25 bytes of log message are converted into 50 hex digits which in turn are converted into 50 ASCII characters (e.g., the byte 0x4e is converted to two ASCII characters “4” and “e”). An escape character byte is appended to the message, and therefore the memory usage for each message is 51 bytes. The Arduino board will write the block of bytes into the SD card when ten log messages are received.

Fig. 7 shows the software flow diagram of the SD card datalogger. First, the Arduino board initializes the SD card and opens the file in which the log messages will be written into. Then the Arduino board waits for log messages from the TelosB mote, with the feedback signal set to 0 (IDLE). When the TelosB mote generates a log message, it first checks whether the feedback signal is IDLE or not. If the feedback signal is IDLE, the TelosB mote sends the log message to the Arduino board. Otherwise, the TelosB mote will hold the data

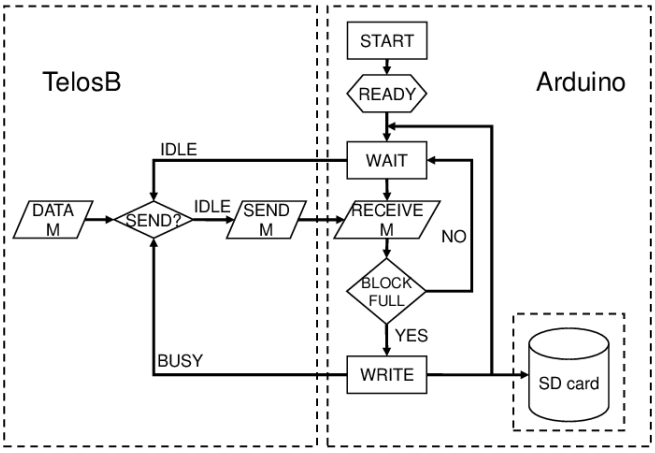


Fig. 7. Software flow diagram of the SD card datalogger. Since SD card is a block-addressable storage device with block capacity of 512 bytes, the Arduino board needs to receive enough log messages to fill in a block and then writes the block of bytes into the SD card.

until the next attempt. Once the Arduino board has received a block of log messages, it sets the feedback signal to 1 (BUSY) and starts to write these data into the SD card. After finishing writing data to the SD card, the Arduino board resets the feedback signal and returns to the WAIT state.

In TeaCP, the timestamps of network events (e.g., packet arrivals at nodes) are recorded by the TelosB motes independently of the SD card datalogger. Thus in terms of timing, the SD card datalogger should not affect the accuracy of recorded timestamps of network events.

E. 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.

1) Packet route and network evolution

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

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. The node locations in the figure are specified by the user in the Python script. Fig. 8 shows an example of such a network topology based on logs of packet ID 210 to 220 from every source node. The label of each edge is the number of packets that go through this link. For

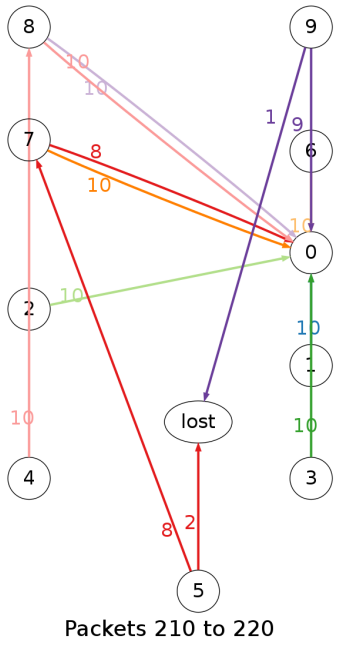


Fig. 8. Network topology visualization using TeaCP. The label of an edge corresponds to the number of packets traversing the edge.

example, for the 10 packets generated by node 5, 8 packets go to node 7 first and then go to the root (node 0). The other 2 packets are lost and the label of edge from node 5 to node "lost" is 2.

For each window of packets, TeaCP generates a network topology with edge labels. Then the generated time-windowed topologies (saved as PNG files) are combined together and converted into a movie (saved as a MP4 file) to show the network topology evolution over time.

2) Reliability and throughput performance

By counting the number of packet generation events, TeaCP obtains the total number of generated packets, which we denote by M. By counting the number of packet reception events at the root, TeaCP gets the total number of received packets by the root denoted by N. After removing duplicate packets from the N packets, the toolkit obtains the number of uniquely delivered packets, N_u. TeaCP computes the delivery rate for the whole network as follows:

$$\text{Delivery rate} = \frac{N_u}{M} \times 100\%. \tag{1}$$

The number of lost packets becomes $M - N_u$ and the number of delivered duplicates will be $N - N_u$. TeaCP also outputs the delivery rate of each source node.

Throughput is defined to be the number of packets received by the root per second and goodput is the number of unique packets delivered to the root per second. Given the total experiment time T, throughput and goodput are computed as follows:

$$\text{Throughput} = \frac{N}{T}, \ \text{goodput} = \frac{N_u}{T}. \tag{2}$$

3) Delay performance

TeaCP utilizes an activator node to send an initial broadcast signal that activates each node and establishes time synchronization. When packet i is generated, the source node records the generation time t_i^g in its clock. When packet i is delivered, the root records its delivery time t_i^d in its clock. Then, the delay of a delivered packet is computed as the difference between the generation time and delivery time (i.e., $t_i^d - t_i^g$). Further, TeaCP generates histograms of packet delivery delays and calculates average delay for each source node. Let $\mathcal{D}$ denote the set of uniquely delivered packets. Then the network-wide average delay is computed as follows:

$$\text{Average delay} = \frac{1}{N_u} \sum_{i \in \mathcal{D}} (t_i^d - t_i^g). \tag{3}$$

The median delay is defined to be the delay quantity in the middle of the delay distribution of the uniquely delivered packets.

It is well known that the clock of a sensor drifts. However, for relatively short experiments (e.g., 30 minutes to an hour) and the same type of sensor nodes, the amount of clock drift is negligible compared to packet delays (around 2 ms for a duration of 30 mins [24, 25]). The issue of time synchronization is further discussed in Section V.

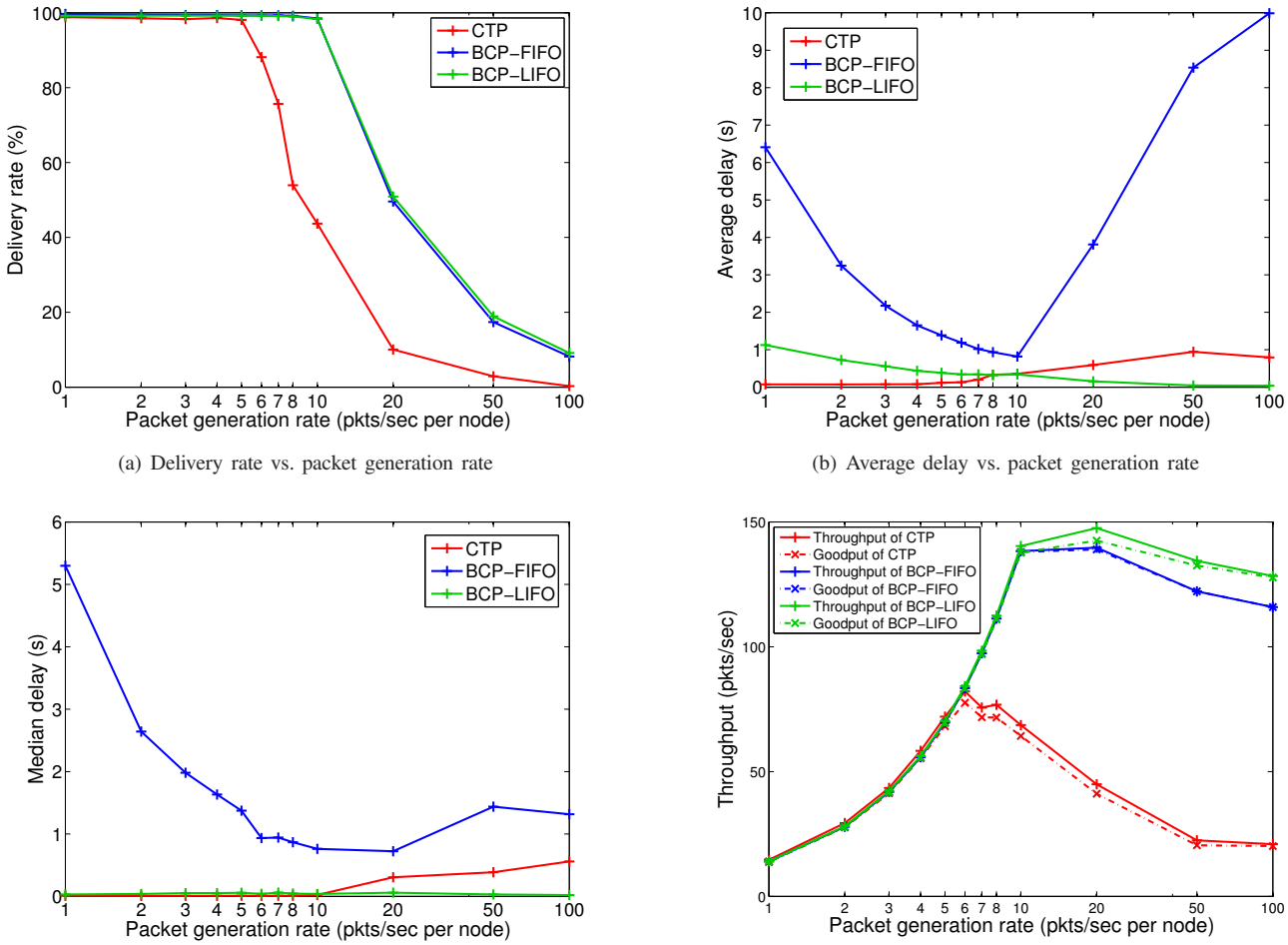
F. TeaCP working with CTP and BCP

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: (i) the collection protocol should provide function for the TeaCP to inject a packet into the network layer; (ii) 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 [26] 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. When used on CTP, TeaCP injects a packet into the network layer by calling the command `Send.send(&packet, sizeof(DataMsg))` provided by CTP. If the mote is a sensor node and receives a packet from other sensor nodes, CTP will notify TeaCP by signaling the event `Intercept.forward(message_t *msg)`. If the mote is the root and receives a packet, CTP signals the event `Receive.receive(message_t *msg)`.

The BCP implementation provides interfaces `Send` and `Receive` but misses the interface `Intercept`. Thus to use TeaCP with BCP, we add the interface `Intercept` in some components and signaling of event `Intercept.forward(message_t *msg)` in the forwarding engine of BCP. Another difference of BCP implemen-



(a) Delivery rate vs. packet generation rate

(b) Average delay vs. packet generation rate

(c) Median delay vs. packet geneartion rate

(d) Throughput/goodput vs. packet generation rate

Fig. 9. Performance evaluation of CTP and BCP in simulations by TeaCP.

tation from CTP is that BCP uses interface `BcpDebugIF` for protocol debugging and then in `TestBenchC` we just provide the interface `BcpDebugIF` and implement its commands with empty functions.

After these changes, TeaCP can test, analyze and evaluate BCP. In total, we modified about 15 lines of code in order to adapt TeaCP to the BCP implementation (around 2500 lines of code).

The IPv6 routing protocol for low power and lossy networks (RPL) [20] has been implemented in TinyOS as TinyRPL. In the IP stack, TinyRPL is the routing engine and the TinyOS component `IPFowardingEngineP` is the forwarding engine. The forwarding engine provides the `Send` and `Receive` interfaces. Similar to the modifications made to BCP, if we add the `Intercept` interface to the forwarding engine `IPFowardingEngineP`, TinyRPL can be used with TeaCP to evaluate its performance.

IV. 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. We showcase the use of TeaCP in three scenarios: (i) TOSSIM simulations; (ii) experiments with logging to the PC; (iii) experiments with logging to the SD card datalogger. We also show how to use TeaCP as a diagnosis tool. Indeed, we identify an inconsistency in the implementation of BCP on the TelosB motes (with radio chip CC2420). This tiny hole causes sensor nodes to always choose the root as the next hop and thus the topology ends up always being a single-hop. We present a simple solution to this problem.

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 (received signal strength indicator) traces of an 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 delay.

Fig. 9 compares the delivery rate, average delay, median 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 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 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 the calculation of throughput.

The results are consistent with the observations made in [9] in two key aspects: (i) BCP provides higher throughput and delivery rate than CTP under high-load conditions, while CTP achieves smaller average delay, and (ii) BCP-LIFO significantly reduces delay of BCP-FIFO while achieving almost the same delivery rate performance.

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 and the memory consumption is around 400 MB.

B. TeaCP performance in real experiments

TeaCP can evaluate collection protocols through experiments. In addition to the reliability, throughput and delay performance, 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 delay histogram, packet loss rate, and route branches due to packet duplicates. Besides, 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 TelosB. The transmission power is configured to -20 dBm for all experiments and the packet generation rate is set to 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. 10. The motes are connected to PCs through a USB port for the purposes of transferring log messages. 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. This broadcast signal is also used to activate each node and establish rudimentary time synchronization. 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 concept of TeaCP functionality in experimental domain for evaluation and diagnosis purposes.

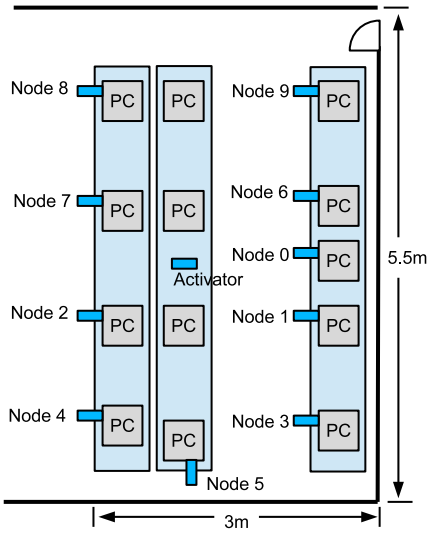


Fig. 10. Placement of the 9 sensor nodes, the root (node 0) and the activator in the experiment.

1) Delay histograms

CTP and BCP(-LIFO) experimental delay performance are shown in Fig. 11. The per-node histograms give detailed statistics of delay of packets generated at each node. Overall, the histograms of the 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. 12 shows CTP topology evolution over time with a window size of 10 packets per each caption. In the first caption, with packet 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 the 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

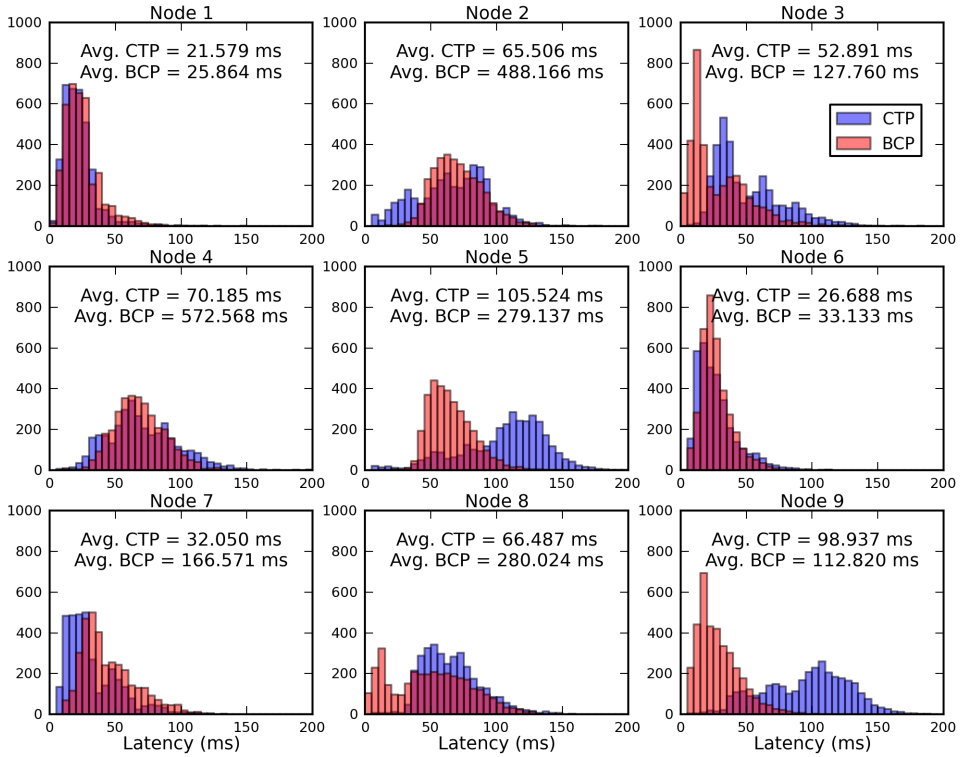


Fig. 11. Delay histograms for CTP and BCP obtained by TeaCP in experiments. Though the delay distribution of BCP and CTP are similar, average delay of BCP is much higher than CTP.

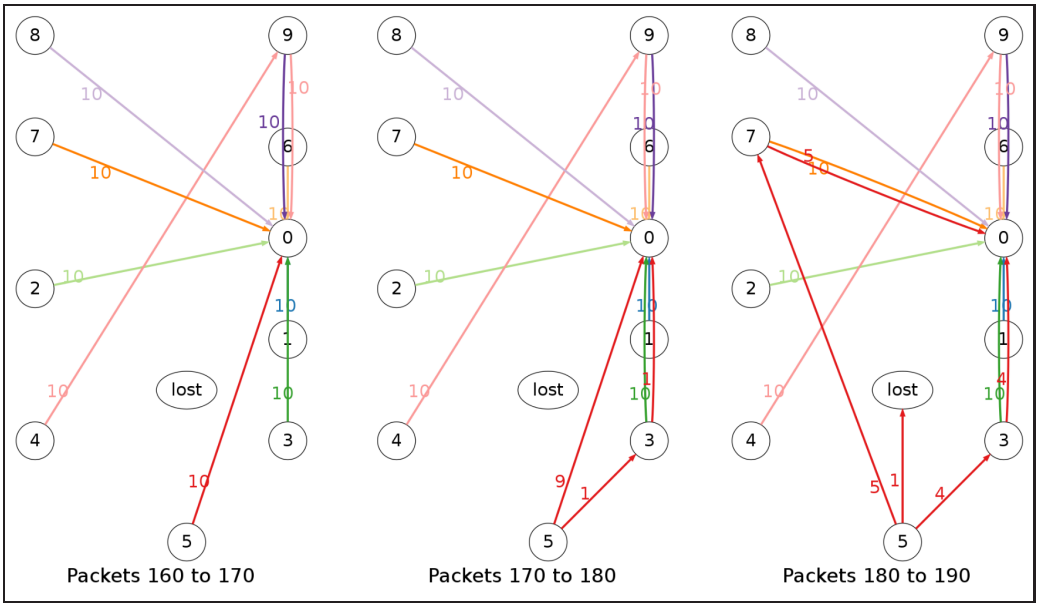
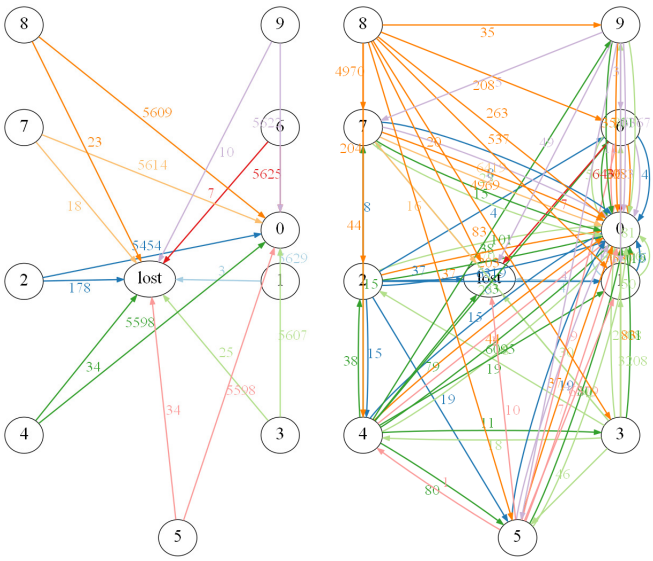


Fig. 12. Topology sample of CTP at 10 packets per image. The topologies are drawn based on packets with ID from 160 to 190 from every source node. The network topology transition indicates change of quality of links. This showcases that TeaCP can be used to investigate the behavior of collection protocols under dynamic scenarios and their ability to adapt the routes according to environmental conditions.

analysis to determine the battery lifespan of such a high-traffic node. It might also be worth considering the possible pros and cons of selecting such a node as the root.

3) System diagnosis

This set of experiments presents the usage of TeaCP as a diagnosis tool. We set up a network consisting of TelosB motes (with radio chip CC2420) running BCP. As we observed, the packet generation rate 2 pkts/sec per node gives reasonable performance results (i.e., BCP protocol collects data with a multi-hop topology and delivery rate of the nodes is higher than 98%). However, when the generation rate increases to 4 pkts/sec per node, TeaCP topology analysis shows that all sensor nodes always choose the root as the next hop. The resultant packet routes are shown in Fig. 13(a).



(a) Single-hop topology of BCP (b) Multi-hop topology of BCP

Fig. 13. BCP network topology obtained by TeaCP. (a) shows the topology (always single-hop) before fixing the inconsistency of BCP implementation with CC2420 (b) shows the multi-hop topology of BCP after making CC2420 consistent with BCP.

The crux of this behavior lies in the following fact. In the TinyOS-implemented BCP, sensor nodes obtain backpressure information (queue length) from neighbors through snooping neighbors' packets when they are transmitting, since the protocol header includes such backpressure information. However, this learning method is not consistent with the default configurations of CC2420. Indeed, CC2420 implements address recognition meaning that the chip filters out the packets that are not destined to the mote and thus does not transfer them to upper layers. Therefore, packet snooping does not work by default for CC2420 chips. However, once we disable the address recognition function of CC2420, BCP supports multi-hop data collection as shown in Fig. 13(b). This example indicates that through the visualization and analysis functions provided by TeaCP, one can diagnose and detect abnormal behaviors of collection protocols.

C. Experimental evaluation of SD card datalogger

1) Benchmarking statistics

We aim to obtain the following benchmarking statistics for the SD card datalogger: (i) recording accuracy; (ii) maximum packet generation rate with close-to-zero packet loss rate; (iii) impact of feedback signal on the recording accuracy. We use the log messages recorded by the PC as a reference for evaluating the accuracy of the SD card datalogger. Thus the recording accuracy is defined to be the ratio of number of log messages recorded by the SD card datalogger to that by the PC.

In this set of experiments, two TelosB motes are connected to both SD card datalogger and PC, which record the log messages simultaneously. Among the two motes, one is the sensor and the other is the root, running CTP. We increase the packet generation rate of the sensor from 10 pkts/sec to 50 pkts/sec. Each experiment runs for 30 minutes.

Packet generation rate (pkts/sec)	Run	#packets recorded by the SD card datalogger		Accuracy (%)		Packet loss (%)
		Sender	Receiver	Sender	Receiver	
10	1	18000	18000	100	100	0
	2	18000	18000	100	100	
	3	18000	18000	100	100	
20	1	36000	36000	100	100	0
	2	36000	36000	100	100	
	3	36000	36000	100	100	
30	1	54000	54000	100	100	1.2×10^{-3}
	2	54000	54000	100	100	
	3	54000	53997	100	100	
40	1	72000	71997	100	100	2.8×10^{-3}
	2	72000	71998	100	100	
	3	72000	71999	100	100	
50	1	90000	89993	100	100	5.6×10^{-3}
	2	90000	89995	100	100	
	3	90000	89997	100	100	

TABLE III
RECORDING ACCURACY OF THE SD CARD DATALOGGER

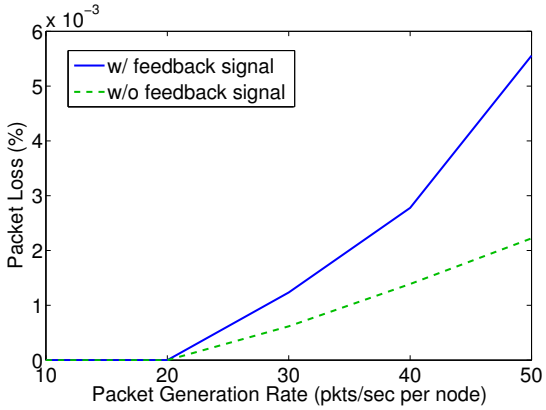


Fig. 14. The performance of data logging with and without the feedback signal.

The results are shown in Table III. The accuracy of the SD card datalogger is 100% for all the packet generation rates, which means that the data recorded by the SD card datalogger is the same as those recorded by the PC. The maximum packet generation rate of the sender is 50 pkts/sec per node. If the packet generation rate of the sender exceeds 50 pkts/sec per node, the system fails and the SD card does not record data.

To investigate the impact of the feedback signal on the data logging performance, we compare the following two scenarios: one is connecting the TelosB mote to the SD card datalogger; the other one is disconnecting the UART interface of the TelosB mote to the Arduino board. In the second scenario, since no data is sent from the TelosB mote to the Arduino board via UART, the feedback signal is always IDLE. Then we compare the packet loss rates based on the log messages recorded by the PC for the two scenarios. The results are depicted in Fig. 14. It shows that the difference of packet loss rates between with feedback signal and without feedback signal is at most 0.004%. Therefore, the addition of the feedback signal on the SD card datalogger does not have obvious impact on the data logging performance.

2) Three-node WSN

Now we use TeaCP with the SD card datalogger on a three-node WSN, whose topology is shown in Fig. 15. The three

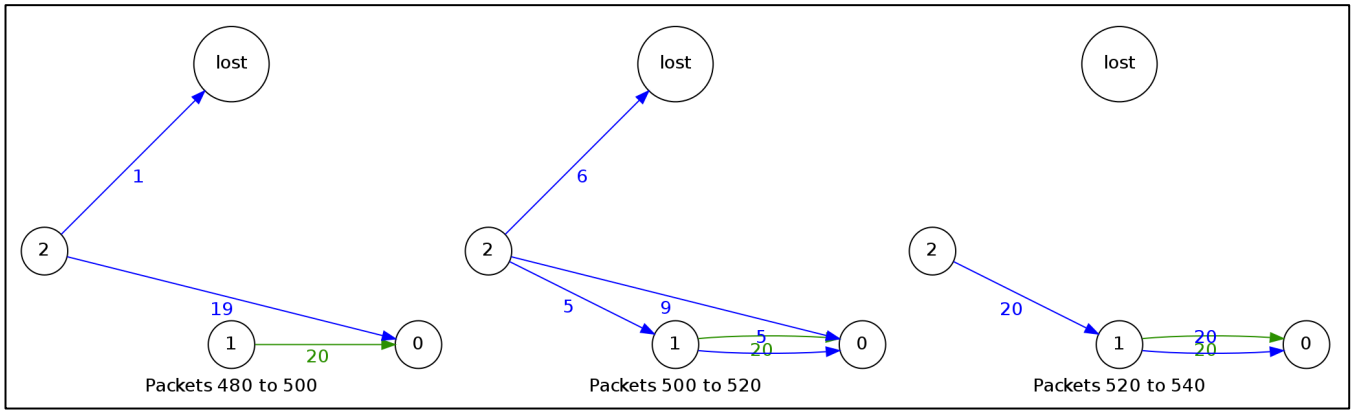


Fig. 16. Topology sample of CTP at 20 packets per image. The topologies are drawn based on packets with ID from 480 to 520 from every source node.

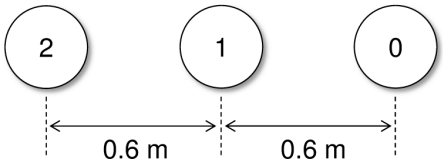


Fig. 15. Locations of sensors in the three-node WSN. Node 0 is the root node. The three nodes are placed in a line with 0.6 m apart between neighboring nodes.

Node statistics					
NodeID	AvgHops	Delivery%	AvgDelay (ms)	Throughput (pkts/sec)	Goodput (pkts/sec)
1	1.00	97.85	9.71	9.85	9.82
2	1.96	95.38	37.99	9.59	9.55
Network statistics					
	AvgHops	Delivery%	AvgDelay (ms)	Throughput (pkts/sec)	Goodput (pkts/sec)
	1.46	96.66	22.90	19.44	19.37

TABLE IV
ANALYSIS STATISTICS OBTAINED BY TEACP

nodes are placed in a line with distance of 0.6 m between neighboring nodes. The underlying collection protocol is CTP. The packet generation rate is 10 pkts/sec and the test runs for 20 minutes.

Fig. 16 shows three captions with a window size of 20 packets for CTP. In the first caption, with packet ID 480 to 500, node 1 and node 2 transmits packets directly to the root. In the second caption, with packet ID 500 to 520, 6 packets are lost and node 2 starts to choose node 1 as relay node. In the last caption, node 2 transmits all 20 packets it generates to node 1. The network topology change is probably due to channel contention of node 1 and node 2, which leads to node 2 taking more transmissions to send a packet to the root node. Based on shortest-path calculation of CTP, it takes fewer transmissions for node 2 to use node 1 as the next hop. Other evaluation statistics obtained by TeaCP is shown in Table IV. TeaCP calculates the statistics for both each node and the whole network, including average number of hops, delivery rate, average delay, throughput and goodput.

V. DISCUSSION

Within this context, we next discuss several of our simplifying assumptions in the current design that may benefit from future extensions.

Time synchronization: In TeaCP, we utilize an activator node to send an initial broadcast signal that activates each node and establishes rudimentary time synchronization. This may not be sufficiently accurate in some cases. For example, in large-scale deployments, all nodes may not receive a central activation signal or their reception times can be different. Under such conditions, TeaCP can be paired with a low-rate and possibly out-of-band time synchronization protocol. The problem of time synchronization within a distributed wireless sensor network has been extensively investigated in previous works (see, for example, [27, 28]).

In real experiments, the clock of each sensor node has localized drift and skew parameters. The authors in [29] have shown that clock drift is different for each node and also the amount of drift increases almost linearly with time. Therefore, we conjecture that if the experiment duration gets long, clock drift may become significant and the accuracy of the delay analysis could be impacted. However, for relatively short experiments (e.g., 30 minutes to an hour) and the same type of sensor nodes, clock drift does not introduce considerable errors compared with packet delays (around 2 ms for a duration of 30 mins [24, 25]). To mitigate the issues with clock drift, lightweight clock calibration methods could be implemented [29]. Note that, in many cases, distributed time synchronization protocols are implemented regardless (e.g., to support duty cycling) and therefore can be employed in parallel with TeaCP to improve its accuracy.

Routing protocols: The current implementation of TeaCP is designed for collection protocols, which route packets from sensor nodes to the root node. Since TeaCP is built on the application layer, it can be used, with some modifications, to evaluate and analyze the performance of other routing protocols such as one-to-many and any-to-any routing. These modifications include change of the packet generation patterns and the definitions of performance metrics in the post-experiment analysis component.

Traffic pattern: In our current implementation and performance evaluation, packet generation is timer driven with a constant rate. In the general case, nodes may have different sleep and wake-up cycles for power saving purposes, which will cause different traffic patterns in the network. Our baseline

fixed-rate model for the network traffic can be generalized to any arbitrary distribution, e.g., Poisson, bursty, etc.

VI. Conclusion and future work

We have developed TeaCP, a toolkit that focuses on visualization and analysis of network layer collection protocol performance, with an optional SD card datalogger. We have shown how our toolkit can be utilized to compare the performance of two well-established protocols, CTP and BCP, though our toolkit is also designed for simple evaluation and analysis of new collection protocols that follow a general mold. The visualization and analysis tools of TeaCP can also provide diagnostic information for data collection systems, and we have demonstrated this by identifying an inconsistency between BCP and the CC2420 radio chip in its standard configuration.

The current version of TeaCP runs on TinyOS and TOSSIM. For future work, it would be useful to extend TeaCP to support other WSN platforms such as Contiki OS [30], the Castalia simulator [31] and the Shawn simulator [32]. In TeaCP, the test configuration and the application layer components are written in the NesC language. To extend TeaCP to other WSN platforms, these two components may have to be rewritten (e.g., using C in Contiki OS). The post-experiment analysis component can be readily used for analyzing the log messages output from other WSN platforms since it is implemented with Python.

Acknowledgements

This work was supported in part by the US National Science Foundation (NSF) under grants CCF-0916892, CCF-0964652, CNS-1012910, and CNS-1409053. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF. The authors also would like to thank Idan Warsawski for his contributions to a preliminary version of TeaCP and Dr. Moshe Laifenfeld for fruitful discussions.

References

[1] W. Si, M. Hashemi, I. Warsawski, M. Laifenfeld, D. Starobinski, and A. Trachtenberg, "Teacp: A toolkit for evaluation and analysis of collection protocols in wireless sensor networks," in *COMCAS*, 2013.

[2] M. Hashemi, W. Si, M. Laifenfeld, D. Starobinski, and A. Trachtenberg, "Intra-car wireless sensors data collection: A multi-hop approach," in *VTC*, 2013.

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

[4] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002.

[5] M. H. Alizai, O. Landsiedel, and K. Wehrle, "Exploiting the burstiness of intermediate-quality wireless links," *International Journal of Distributed Sensor Networks*, 2012.

[6] M. Alresaini, M. Sathiamoorthy, B. Krishnamachari, and M. J. Neely, "Backpressure with Adaptive Redundancy (BWAR)," in *INFOCOM*, 2012.

[7] 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/.

[8] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis, "Ctp: An efficient, robust, and reliable collection tree protocol for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 10, no. 1, pp. 16:1–16:49, Dec. 2013.

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

[10] C. Buschmann, D. Pfisterer, S. Fischer, S. P. Fekete, and A. Kröller, "Spyglass: a wireless sensor network visualizer," *SIGBED Rev.*, vol. 2, no. 1, pp. 1–6, Jan. 2005.

[11] R. Jurdak, A. G. Ruzzelli, A. Barbirato, and S. Boivineau, "Octopus: monitoring, visualization, and control of sensor networks," *Wirel. Commun. Mob. Comput.*, vol. 11, no. 8, pp. 1073–1091, Aug. 2011.

[12] B. Parbat, A. K. Dwivedi, and O. P. Vyas, "Data Visualization Tools for WSNs: A Glimpse," *International Journal of Computer Applications*, vol. 2, no. 1, pp. 14–20, 2010.

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

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

[15] OMNeT++, http://www.omnetpp.org/.

[16] Tutornet: A Tiered Wireless Sensor Network Testbed, USC Embedded Networks Laboratory. http://enl.usc.edu/projects/tutornet/.

[17] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: a wireless sensor network testbed," in *IPSN*, 2005.

[18] V. Gutirrez, J. Galache, L. Snchez, L. Muoz, J. Hernndez-Muoz, J. Fernandes, and M. Presser, "Smartsantander: Internet of things research and innovation through citizen participation," in *The Future Internet*, ser. Lecture Notes in Computer Science, A. Galis and A. Gavras, Eds. Springer Berlin Heidelberg, 2013, vol. 7858, pp. 173–186.

[19] E. Pinedo-Frausto and J. Garcia-Macias, "An experimental analysis of zigbee networks," in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, Oct 2008, pp. 723–729.

[20] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, P. Levis, K. Pister, R. Struik, and J. Vasseur, "RPL: Ipv6 routing protocol for low power and lossy networks," 2011. [Online]. Available: http://tools.ietf.org/html/rfc6550

[21] J. Zhang, K. Xu, and Y. Liu, "Realization of spi interface simulated by msp430," *Computer Engineering and Design*, vol. 29, pp. 1169–1171, March 2008.

[22] SALAMANDER Project, "Build a datalogger for your wireless sensor network," http://www.instructables.com/id/build-a-datalogger-for-your-wireless-sensor-networ/.

[23] A. Banerjee, A. Mitra, W. A. Najjar, D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos, "Rise-co-s: high performance sensor storage and co-processing architecture." in *SECON*, 2005, pp. 1–12.

[24] J. M. Castillo-Secilla, J. M. Palomares, and J. Olivares, "Temperature-compensated clock skew adjustment," *Sensors*, vol. 13, no. 8, pp. 10 981–11 006, 2013.

[25] M. B. Uddin and C. Castelluccia, "Towards clock skew based services in wireless sensor networks," *Int. J. Sen. Netw.*, vol. 9, no. 1, pp. 24–37, Dec. 2011.

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

[27] K.-L. Noh, E. Serpedin, and K. Qaraqe, "A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization," *Wireless Communications, IEEE Transactions on*, vol. 7, no. 9, pp. 3318–3322, 2008.

[28] J. Liu, Z. Zhou, Z. Peng, J.-H. Cui, M. Zuba, and L. Fiondella, "Mobi-sync: efficient time synchronization for mobile underwater sensor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 2, pp. 406–416, 2013.

[29] J. Huang, S. Liu, G. Xing, H. Zhang, J. Wang, and L. Huang, "Accuracy-aware interference modeling and measurement in wireless sensor networks," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 172–181.

[30] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, Nov 2004, pp. 455–462.

[31] A. Boulis, "Castalia: Revealing pitfalls in designing distributed algorithms in wsn," in *SenSys '07*. New York, NY, USA: ACM, 2007, pp. 407–408.

[32] S. Fekete, A. Kroller, S. Fischer, and D. Pfisterer, "Shawn: The fast, highly customizable sensor network simulator," in *Networked Sensing Systems, 2007. INSS '07. Fourth International Conference on*, June 2007, pp. 299–299.