FastForward: High-Throughput Dual-Radio Streaming

Gholamhossein Ekbatanifard
Islamic Azad University, Lahijan branch
ekbatanifard@ieee.org

Philipp Sommer, Branislav Kusy
CSIRO, Australia
{philipp.sommer,brano.kusy}@csiro.au

Venkat Iyer, Koen Langendoen
Delft University of Technology
{v.g.iyer,k.g.langendoen}@tudelft.nl

***Abstract*—The high popularity of wireless sensor networks has led to novel applications with diverse, and sometimes demanding, data communication requirements, for example, streaming camera images in surveillance applications. In response bulk-data transfer protocols were proposed that provide low latency and high throughput communication over multiple hops. However, due to typical hardware platforms only providing a single radio, which implies that forwarding nodes need to serialize send and receive actions, the maximum end-to-end throughput is limited to 1/2 the radio capacity. To bridge this performance gap we present FastForward, a connection-oriented multi-hop bulk-data transfer protocol optimized for dual-radio platforms; data packets are sent across a path of alternating radio and frequency channels to exploit parallel transfers and avoid intra-path interference.**

We implemented FastForward in TinyOS to run on the Opal platform equipped with two IEEE 802.15.4 radios. In this paper we show that, with some minor tweaking of the original protocol stack to streamline internal access to the SPI bus, FastForward is capable of operating both radios in parallel so packets can be forwarded at full speed. We have evaluated FastForward on a 12-node testbed in an office environment. The sustained throughput peaks around 23.7 kBps, or 76 % of the radio capacity while the best single-radio protocol flattens out at 19 %. When introducing artificial packet loss the built-in link-level acknowledgements ensure that FastForward manages to deliver packets with high yield (close to 100 %) at the sink across 11 hops.

I. Introduction

Originally conceived for continuous sensing of the physical world at low data rates, wireless sensor networks (WSN) have since been used in a diverse set of applications. Many applications aspire to transmit large amount of data at high throughput and/or low latency. For example, structural health of the Golden Gate Bridge was sampled at 1 KHz in [1]. Brimon, a railway monitoring system, generates about 7 KB in each train pass for each sensor node [2]. Finally, Werner-Challen et al. have sampled volcano activity at 1.2 KB per second per node [3]. All of these applications require the network to perform at close to its maximum capacity over multiple hops and for significant amounts of time.

Traditional WSN data collection protocols do not support high data throughput or low latency well as they were mainly optimized for low-power operation. Increasing the rate at which radio packets are transmitted on the wireless channel makes it more likely that packets will interfere, thus decreasing the overall throughput. Large data transfer in WSNs has thus emerged as an important research topic in recent years, also known as bulk data transfer [4]–[6]. Most approaches consider the problem of achieving high throughput to be orthogonal to the low-power operation as ample energy resources are required to transfer large amounts of data. Effectively, the bulk transfer protocols (e.g., Flush [4] and PIP [5]) establish a multi-hop route between the source and the sink, disable duty-cycling, and reserve the radio channel(s) until the bulk transfer is over. Having the complete control over the radio, the protocols achieve high throughput and low latency by enforcing an optimal end-to-end schedule of packet transmissions across all hops. In particular, the TDMA technique employed in PIP has been shown to achieve throughputs of approximately 7 kBps (25 % of the available 802.15.4 capacity). More recently, Duquennoy et al. [6] have shown that the low-power operation does not need to be compromised during the bulk transfer. Using a combination of high-throughput bursts and a store-and-forward technique, the Low Power High Throughput (LPHT) [6] protocol achieves high throughput and low-power operation at the same time; only the two nodes engaged in sending/receiving packets switch off duty cycling, while all other nodes in the pipeline stay in low-power mode.

We note that *all* bulk transfer protocols face a fundamental problem on *single* radio platforms: the radio cannot be receiving and transmitting at the same time. The maximum theoretical end-to-end throughput for the bulk transfer is thus 50 % of the channel capacity, as also observed in [6]. Furthermore, single-radio bulk protocols use channel hopping to eliminate self-interference of packet transmissions with the down-stream traffic. The overhead of changing channels further decreases the throughput over multiple hops. The result is that the best performing protocols (e.g., PIP [5]) only achieve throughputs of approximately one quarter the 250 kbps channel capacity defined in the IEEE 802.15.4 standard [7].

In this paper we explore the advantages of using platforms like Opal [8] that include *multiple* radios. Although originally designed for resilience to external interference, the dual radios on the Opal platform can also be used to speed up bulk data transfer by streaming packets on both radios in parallel. We present FastForward, a novel connection-oriented bulk-data transfer protocol that achieves throughput of close to the theoretical radio capacity, a factor of four improvement over existing single-radio protocols. FastForward uses multiple radios to simultaneously receive and forward packets and multiple radio channels to cope with inter- and intra-path interference. This allows the data packets to be transmitted back-to-back at high throughput and low latency. FastForward also offers high reliability through a combination of mechanisms, such as link-layer acknowledgements and back pressure signaling at the source node. We evaluate FastForward in a series of experiments in an indoor office-space testbed. The results over streaming routes of upto 11 hops long indicate that FastForward achieves throughputs of up to 27.8 kBps, which is close to the underlying radio capacity, irrespective of the number of hops. We also evaluate the correctness of FastForward's internal mechanisms in networks with lossy links. We show that a combination of link-layer acknowledgments and back-pressure signaling helps FastForward to significantly improve reliability of end-to-end packet delivery in networks with lossy links. In summary, the contributions of this paper are:

- We provide a detailed description of FastForward showing that multiple radios allow for bulk data streaming at the theoretical channel capacity, irrespective of the number of hops. This is a factor of four improvement over the best implementation on single radio platforms.

- We discuss the MAC-level tuning of typical low-power protocols required to achieve high-throughput, such as interleaving of SPI transactions and setting back-off parameters to ultra short values.
- We report on an extensive set of experiments in a real-world testbed.

II. Wireless Channel Capacity

The capacity of the wireless channel is determined by the modulation scheme and data rate. The IEEE 802.15.4 standard [7] defines several modulation schemes for the physical layer. Offset quadrature phase-shift keying (O-QPSK) modulation is widely used for radio transceivers in the 2.4 GHz band, where four bits of the data stream are mapped to one of 16 symbols. Each symbol is then represented by a unique pseudo-random sequence of 32 chips to provide signal spreading. By default, the transmission of a single chip takes 0.5 µs, which results in an overall data rate of 250 kbps.

A. Upper Bound with a Single Radio Transceiver

The single-hop wireless channel capacity is limited by the format of the physical layer data unit and technical limitations of the radio transceiver, such as delays to perform state transitions [9]. Using a node with a single radio transceiver to receive and forward packets involves the following steps for each forwarded packet (see Figure 1): (1) receive packet, (2) send acknowledgment, (3) forward packet, and (4) wait for acknowledgment. Each transition from receive to transmit state ($RX \rightarrow TX$) and from transmit to receive state ($TX \rightarrow RX$) involves a delay in the order of a few tens of microseconds due to the radio hardware. To derive an upper bound for the throughput, we assume that there is no delay involved for reading/writing the packet buffers or sending commands to the transceivers state machine. Furthermore, data packets are acknowledged immediately and no packets or acknowledgments are lost. We assume a total size of 133 Bytes for data and 7 Bytes for acknowledgment packets and use the datasheet values for state transition timings of the Atmel AT86RF231 radio, this results in an upper bound of 117.4 kbps for the throughput, which is 47.0 % of the theoretical capacity.

B. Upper Bound with Dual Radio Transceivers

Using a node platform with two radio transceivers makes it possible to have a dedicated radio both for receiving and forwarding of data packets. Consequently, the first radio is now able to receive subsequent packets earlier since the packet forwarding has been delegated to the second radio, as shown in Figure 1. Therefore, we are able to achieve twice the throughput (234.9 kbps or 94.0 % of the channel capacity) by using two independent radio transceivers. Note that an alternative implementation could use one radio transceiver exclusively for all receive operations (data and acknowledgments), and the second radio exclusively for transmissions. This would allow to eliminate the time consuming transceiver state switching, but would incur additional overhead for switching radio channels. For the current hardware the channel and state switching overheads are of the same magnitude, so we would expect a similar performance.

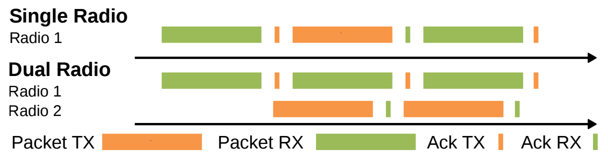


Figure 1: Upper bound for the channel capacity in the single (top) and dual radio transceiver case (bottom).

III. FastForward Protocol

In this section, we present the design of FastForward, a novel bulk transfer protocol targeted at multi-radio mote platforms, such as the Opal sensor node [8]. The FastForward protocol is intended to co-exist with other WSN protocols by alternating between periods of high throughput data transfer and low-power operation. During regular operation the network is running a MAC protocol that is optimized for minimizing the power consumption. When streaming is requested by the application, a subset of nodes invokes the FastForward protocol to deliver packets between a source and a sink at high throughput and with low latency.

A. Using Multiple Radio Transceivers

FastForward uses multiple radio transceivers to improve both throughput and latency of streaming. The basic idea is to alternate between radio transceivers at each hop. An intermediate node is receiving packets from its upstream neighbor using one radio while it is forwarding packets on another radio. One important requirement is that the two radios are capable of operating on different channels and/or frequency bands, to prevent in-band interference between the radios. The problem of self-interference on the streaming path needs to be considered beyond the radio range of a single node. For example, the first and the third node in Figure 2 transmit on different channels of the same radio. Specifically, FastForward requires at least two available channels on each of the radios and all channels need to be mutually interference free.

Since the sink node has two radios, it can efficiently support two separate streams at the same time. However, the two paths must be node disjoint and radio channels should not overlap between the two paths in order to achieve maximum throughput. A typical scenario of using two simultaneous streams would be a line topology with the sink placed in the middle of the line.

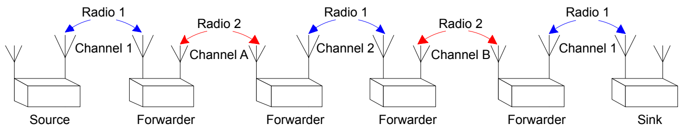


Figure 2: Example of a network topology for streaming.

B. Stream Setup

In the following, we briefly discuss mechanisms for using FastForward along existing network protocols. We assume that the streaming phase is initiated by the application layer either at the sink or source node [1], or follows a pre-defined schedule [10]. Starting a new stream requires all nodes along a path between the source and sink node to switch from low-power into streaming mode. Establishing a streaming connection includes an assignment of alternating radios and corresponding channels for each link along the path. It is well known that finding such an optimal route and channel allocation form non-trivial problems for wireless networks [11]. In this paper we determine for each node a good path of alternating radio channels to the base station beforehand, and provide this information at runtime by means of a lookup table to the base station that then can initiate the stream setup required for each specific experiment.

C. High Throughput

FastForward aims to provide best-effort, high-throughput multi-hop streaming. Every node has two radio interfacesoperating on different frequency bands. Using multiple radio transceivers connected to a single microcontroller provides several benefits over single-radio protocols:

- Both radios can receive and transmit packets independently from each other, thus doubling the overall throughput.
- Intra-path interference is reduced because a dedicated radio and/or channel is used for each link.
- Using multiple radios eliminates the overheads that relate to radio channel switching or the latency associated with bursts.

Data streaming over multiple radios requires a careful orchestration of the hardware with the software stack. Processing and forwarding of data packets involves shifting a packet from the receive buffer of one radio to the output buffer of the other radio. Unless implemented as a system-on-chip operation that requires a shared memory for both radios, shifting packets between radios and the microcontroller necessitates a sequence of bus accesses (e.g. the SPI bus available on common node architectures), which can create a bottleneck when the bus is shared as is the case for our platform. We expect this to be the typical case, as multi-radio platforms are usually designed to provide radio diversity and assume that only one radio is active at a time.

However, transferring a byte on the SPI bus is usually much faster than transmitting a byte over the wireless channel, so the serialization of the transfers across the SPI bus have a small impact on the overall forwarding latency. Note that SPI transactions slightly delay transmission of acknowledgments (ACKs) and transmission of the next packet in the queue after successful ACK reception. Overall, this introduces about 10% delay on our Opal platform.

D. Reliability

The goal of FastForward is to provide upper layers with a best-effort protocol for high throughput streaming. Similarly to other protocols, *e.g.*, CTP [12], we use acknowledgements and retransmissions at the link-layer to mitigate the effect of packet loss due to lossy wireless channels. A FIFO packet forwarding queue at each intermediate node is used to buffer packets until they have been successfully transmitted and acknowledged by the next hop. In case no acknowledgement has been received after several transmission attempts, however, the packet is dropped silently. It is then the responsibility of the application layer to request re-transmissions of lost packets to ensure end-to-end reliability when desired.

Using link-layer acknowledgements will increase the reliability when the packet reception rate (PRR) across intermediate links is low. Clearly, the overall throughput will decrease since a node has to wait for an acknowledgement from the receiver before it can transmit the next packet. However, the end-to-end latency will only slightly increase when using acknowledgements since the additional overhead to send an ACK packet over the radio is rather small. This trade off is explored in Section V.

E. Backpressure signaling

A secondary advantage of using link-layer acknowledgements is that they provide a crude - but effective - form of flow control, which is necessary to prevent the source node from choking a bottleneck node along the path with packets it cannot handle. Without rate control, we observed that packets accumulate and overflow relatively-small packet forwarding queues at nodes with slightly worse outgoing link quality. Consequently, packets are dropped en-route to the sink and costly end-to-end retransmissions are required for their retransmission. In FastForward, a forwarding node stops sending packet acknowledgements when its FIFO queue fills up completely. Consequently, the upstream node will keep transmitting its next packet until there is space available in the queue to store (and forward) that packet. This simple rate control mechanisms will cause back pressure on all upstream nodes in turn and eventually slow down the source node. In the future we plan to use more advanced schemes, e.g., windowing-based techniques, to reduce the signalling overhead (cf. Section V).

IV. Implementation

FastForward has been implemented for the Opal platform [8] using TinyOS 2.1.2. In this section, we discuss the challenges faced when implementing streaming applications on multi-radio platforms.

A. Fast Packet Forwarding

Forwarding packets as quickly as possible is key to achieving low latency, while short inter-packet intervals allow to operate at high throughput. In the following we discuss several optimizations to the TinyOS 2.1.2 ActiveMessage (AM) communication stack and the underlying radio drivers.

The FastForward protocol is implemented as an additional layer between the application and the dual-radio stack of the Opal platform. Figure 3 shows a high-level overview of the integration of FastForward within existing TinyOS components. Standard AM interfaces (`Send` and `Receive`) are used to transfer messages between the application and the streaming layer. Both radios have their separate hardware driver component, MAC and link layers. Received packets of both radio stacks are delivered to the `StreamForwardingEngine` component, which serves as a dispatcher for incoming packets. First, each packet is checked for duplicates. If a packet is not identified as a duplicate, it will be either delivered to the application layer (on the sink node) or it is added to the outgoing message queue (`SendQueue`) for forwarding. Packets in the send queue will be processed by the `StreamRoutingEngine` component based on the stream identifier assigned to each packet, which determines the address of the next hop and the radio used for forwarding the packet. Next, the packet is dispatched by the routing engine to the corresponding radio stack for transmission and it is added to the sent packet cache (`SendCache`).

Since it is assumed that the corresponding radio channel is reserved for exclusive access by that node and to maximize throughput, we transmit the packet immediately without performing clear channel assessments or random backoffs as in CSMA.

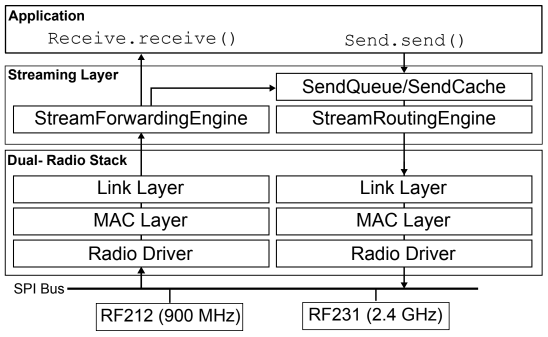


Figure 3: Overview of the TinyOS implementation of a dual-radio forwarder node.

B. Link-Layer Acknowledgments

By default, the request bit for an acknowledgment is set in the 802.15.4 packet header, requiring the receiver to send an acknowledgment packet. If no acknowledgment has been received by the sending node within a certain timeout (1 ms), the packet will be retransmitted. Clearly, There is a tradeoff between reliability and throughput when using link-layer acknowledgments (see also Section V). If no acknowledgment has been received even after multiple retransmissions, the packet is dropped. In this case, it is the responsibility of the application layer to inject the packet again at a later time. FastForward also allows applications to disable acknowledgments completely if high throughput is deemed more important than reliable end-to-end packet transport.

C. Shared Access to SPI Bus

The shared SPI bus on the Opal platform creates a bottleneck for data transfers between the two radios and the microcontroller [13]. In order to avoid concurrent access to the shared bus, TinyOS implements resource arbitration where a component can acquire exclusive access to the bus until the lock is released. Radio driver implementations in TinyOS (*e.g.*, the RF212/231 drivers) generally assume that the radio driver can hold a lock on the SPI bus until packet reception or transmission is completed. However, data transfers on the SPI bus are significantly shorter than the byte-wise transmission of packets over the air due to the difference in transmission speed. On the Opal platform, writing a SPI packet to the radio's TX buffer takes less than 10% of the time required to send it over the air. Therefore, we modified the TinyOS drivers for the RF212 and RF231 radios such that the SPI bus is released immediately when not in use (and re-acquired later), so that the other radio can access the bus meanwhile.

V. Evaluation

We evaluate FastForward primarily on our 12-node indoor testbed that covers two large wings of an office building separated by an outdoor space. We focus on evaluating the main two design features of our protocol: best-effort reliability and high throughput. Specifically, we introduce packet losses on selected links and selectively disable some of the FastForward mechanisms to evaluate their efficacy (Table I showcases the protocol versions).

A. Experimental Setup

Both radios are transmitting using O-QPSK modulation at 250 kbps and with a constant transmission power of 3 dBm.

Table I: Naming convention for the different FastForward protocol versions.

Name	Acknowledgment	Backpressure
noAckFastForward	no	no
noBPFastForward	yes	no
FastForward	yes	yes

Channel assignment has been performed manually to avoid intra-channel interference on nearby links. The initial network topology is a route spanning 12 nodes and 11 hops and we shortened the path length by one hop at a time. The sink node remained the same for all experiments, as well as the radio and channel assignment on each link.

We repeat an experiment 10 times on each topology and average the results. In each test run, the source node generated 1000 packets with a payload size of 100 Bytes each. However, the effective number of bytes transmitted over the wireless channels was 122 due to the overhead for the packet preamble and various headers. Packets are transmitted as quickly as possible without any additional delay (other than processing overhead) between packets.

The following metrics were used in our experiments: Throughput is defined as the raw number of bytes received corresponding to non-duplicate packets at the sink per second including the non-payload bytes such as packet preamble and headers. Data Yield is defined as the number of unique packets received at the sink divided by the number of packets transmitted by the source.

B. Indoor Testbed

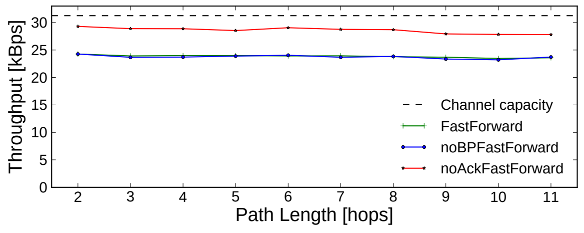
We first evaluate FastForward in our indoor testbed for data paths of up to 11 hops long. The throughput lines in Figure 4a demonstrate that our FastForward implementation achieves data streaming throughput close to the theoretical capacity of the underlying radio links, irrespective of the length of the data stream. FastForward with link-layer acknowledgements achieves a sustained throughput of 23.7 kBps (75.8 % of the maximum capacity), while single radio protocols can only stream at 50 % at best.

When the quality of the underlying links is high, a further 13.1 % boost in throughput can be achieved by disabling link-layer acknowledgements allowing FastForward to operate at 27.8 kBps (88.9 % of the channel capacity). However, the data yield will decrease when acknowledgements are not enabled, as shown in Figure 4b. On our 11 hops topology We observe a packet yield of 95.1 % in contrast to 100 % yield with acknowledgements. Finally, backpressure signaling offers limited benefits in this scenario with only good links as can be noticed from the overlap of the FastForward and noBPFastForward plots.

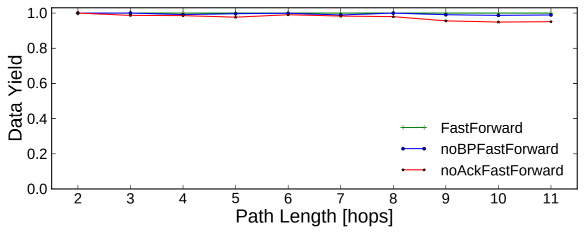
C. Lossy Links

We next evaluate the performance of FastForward in networks with lossy links. To be able to quantify the effect of packet losses in detail, we used our testbed with high-quality links and introduced a specific 20 % packet loss on the incoming link of the sink node. We then selectively disabled acknowledgments and backpressure mechanisms and studied their impact on FastForward's performance.

Figure 5a shows that end-to-end throughput is bounded by the throughput of individual links. Introducing a random 20 %

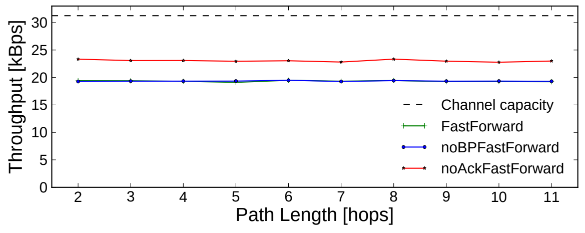


(a) Throughput

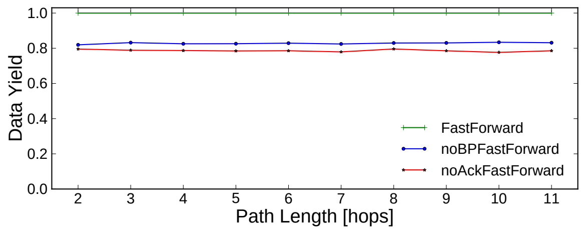


(b) Data yield

Figure 4: Performance of FastForward in the indoor testbed.



(a) Throughput



(b) Data yield

Figure 5: Performance of FastForward **with lossy links**.

packet loss on the last link before the sink decreases the end-to-end throughput by 20 %. On the other hand, FastForward delivers close to 100 % of packets to the sink if both acknowledgements and backpressure are enabled (see Figure 5b). Without backpressure signaling, the source node pushes more data in the network than the network is capable of delivering and 20 % of packets are dropped due to buffer overflows.

VI. Conclusion

In this paper we have presented the novel FastForward protocol that has been explicitly designed to take advantage of hardware platforms that provide multiple radios, like the Opal nodes including a 900 MHz and 2.4 GHz radio. Since the Opal nodes were originally designed for resilience to external interference, the idea of running both radios in parallel - to receive and forward data packets at the same time - required some tweaking of the TinyOS protocol stack. In particular we had to make modifications to the radio drivers to ensure a proper interleaving of the SPI bus accesses, and to various MAC-level parameters tuned for low-power operation (e.g., cutting back-off times short) to achieve the full potential of the Opal hardware with respect to end-to-end throughput. To handle packet loss across individual hops on the route from source to sink, FastForward uses link-layer acknowledgements and retransmissions.

The experimental results show that FastForward achieves throughput of up to 76 % of the radio capacity. Compared to the best single-radio protocol (LPHT) the throughput achieved by FastForward is a factor of four higher, which can be attributed to its parallel streaming on two radios and aggressive tuning of the MAC-level parameters. The acknowledgements and backpressure signaling incur a 10 % overhead in perfect conditions, but help to improve end-to-end reliability (data yield) close to 100% even when introducing (artificial) packet loss up to 20 %. We feel this is a price worth paying in real-world use cases.

The main area of our future work is to further optimize timing and interleaving of SPI transactions. We will also explore hardware modifications to our Opal platform, to enable packet forwarding without incurring CPU overhead. Finally, we would like to study energy-per-bit performance of our protocol and compare it to single-radio protocols.

References

[1] S. Pakzad, G. Fenves, S. Kim, and D. Culler, “Design and implementation of scalable wireless sensor network for structural monitoring,” *Journal of Infrastructure Systems*, vol. 14, no. 1, pp. 89–101, 2008.

[2] K. Chebrolu, B. Raman, N. Mishra, P. Valiveti, and R. Kumar, “BriMon: A Sensor Network System for Railway Bridge Monitoring,” in *Proc. MobiSys*, 2008, pp. 2–14.

[3] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, “Deploying a wireless sensor network on an active volcano,” *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[4] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica, “Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks,” in *Proc. SenSys*, 2007, pp. 351–365.

[5] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale, “PIP: A Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer,” in *Proc. SenSys*, 2010, pp. 15–28.

[6] S. Duquennoy, F. Österlind, and A. Dunkels, “Lossy Links, Low Power, High Throughput,” in *Proc. SenSys*, Nov. 2011, pp. 12–25.

[7] “802.15.4-2006 - IEEE Standard for Information technology,” IEEE, Tech. Rep., 2006.

[8] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brunig, “Opal: A multiradio platform for high throughput wireless sensor networks,” *IEEE Embedded Systems Letters*, vol. 3, no. 4, pp. 121–124, 2011.

[9] F. Österlind and A. Dunkels, “Approaching the Maximum 802.15. 4 Multi-hop Throughput,” in *Proc. HotEmNets*, 2008.

[10] R. Flury and R. Wattenhofer, “Slotted Programming for Sensor Networks,” in *Proc. IPSN*, 2010, pp. 24–34.

[11] Y. Wu, J. Stankovic, T. He, and S. Lin, “Realistic and efficient multi-channel communications in wireless sensor networks,” in *Proc. INFOCOM*, 2008, pp. 1193–1201.

[12] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, “Collection Tree Protocol,” in *Proc. SenSys*, 2009, pp. 1–14.

[13] J. Ko, K. Klues, C. Richter, W. Hofer, B. Kusy, M. Bruenig, T. Schmid, Q. Wang, P. Dutta, and A. Terzis, “Low power or high performance? a tradeoff whose time has come (and nearly gone),” in *Proc. EWSN*, 2012, pp. 98–114.