

# Pando: Fountain-Enabled Fast Data Dissemination With Constructive Interference

Wan Du, *Member, IEEE, ACM*, Jansen Christian Liando, *Student Member, IEEE*,  
Huanle Zhang, *Member, IEEE*, and Mo Li, *Senior Member, IEEE, Member, ACM*

**Abstract**—This paper presents *Pando*, a completely contention-free data dissemination protocol for wireless sensor networks. *Pando* encodes data by Fountain codes and disseminates the rateless stream of encoded packets along the fast and parallel pipelines built on constructive interference and channel diversity. Since every encoded packet contains innovative information to the original data object, *Pando* avoids duplicate retransmissions and fully exploits the wireless broadcast effect in data dissemination. To transform *Pando* into a practical system, we devise several techniques, including the integration of Fountain coding with the timing-critical operations of constructive interference and pipelining, a silence-based feedback scheme for the one-way pipelined dissemination, and packet-level adaptation of network density and channel diversity. Based on these techniques, *Pando* can accomplish data dissemination entirely over the fast and parallel pipelines. We implement *Pando* in Contiki and for TelosB motes. We evaluate *Pando* with various settings on two large-scale open test beds, Indriya and Flocklab. Our experimental results show that *Pando* can provide 100% reliability and reduce the dissemination time of state of the art by  $3.5\times$ .

**Index Terms**—Wireless sensor networks, data dissemination, fountain codes, constructive interference.

## I. INTRODUCTION

**D**ATA dissemination reliably diffuses a bulk of data to all the nodes in a wireless sensor network. It provides a fundamental service for many applications, like on-the-air reprogramming [26] and application updating [2]. One important metric of data dissemination for these applications is reliability, which is measured by the proportion of the data received at each node over the data to be disseminated. To provide high reliability, traditional data dissemination protocols [20], [21], [24], [27], [28], [36], [39] flood data packets over multiple hops with contention-based access control protocols like CSMA/CA, which suffers long backoff delay and severely limits the channel spatial reuse. Recent advance

in physical-layer cooperative broadcasting [41] allows multiple synchronized transmissions constructively add on each other. Multiple senders can thus transmit the same packet simultaneously and the constructively interfered transmissions can be successfully decoded at the receivers. Glossy [16] builds upon this principle and experimentally demonstrates that constructive interference can remove unnecessary channel contention and improve the flooding performance. Another work, PIP [38], builds a pipeline that enables adjacent links in a multi-hop path to operate concurrently with multiple channels. Based on Glossy and PIP, Splash [6] enables data dissemination over the fast and parallel pipelines by combining constructive interference and pipelining. It is based on a tree topology of sensor networks. The tree has multiple layers from the source to the lowest layers. Different layers have multiple sensor nodes. The fast and parallel pipelines enables the multiple nodes at the same layer and the nodes at different layers to transmit at the same time, thereby significantly improving the channel utilization.

The combination of constructive interference and pipelining, however, represents a challenge to the reliability of data dissemination. Due to the varied quality of different channels at different nodes [4], [49] and the unreliability of constructive interference when the number of concurrent transmitters or the packet size is large [6], [47], the packet reception rate at different nodes in a network may vary significantly. As a result, the same data object needs to be disseminated in multiple rounds to recover specific packets missing at few nodes. To achieve high reliability, Splash requires three rounds of data dissemination and employs contention-based local recovery after three rounds of pipelined data dissemination. Splash suffers from the long tail problem, where the dissemination time for achieving 100% reliability is orders of magnitude longer than the time needed for 80% reliability.

This paper presents *Pando*, a completely contention-free data dissemination protocol, which tackles the long tail problem in data dissemination of constructive interference and pipelining. To accomplish the dissemination entirely over the contention-free pipelines, *Pando* introduces randomness in the disseminated packets. It encodes data with Fountain codes [35] and continuously pumps the rateless stream of encoded data into the network. Since every encoded packet contains innovative information for the original data, receivers become insensitive to the loss of individual packets. *Pando* avoids the overhead of re-disseminating the entire data object

Manuscript received September 24, 2015; revised March 29, 2016 and August 4, 2016; accepted September 1, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Y. Eun. This work was supported in part by the Singapore MOE AcRF Tier 1 under Grant RG17/13 2, in part by the MOE AcRF Tier 2 under Grant MOE2012-T2-1-070, and in part by the NTU Nanyang Assistant Professorship under Grant M4080738.020. A preliminary version of this work was published in the Proceedings of ACM SenSys 2015 [13].

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: duwan@ntu.edu.sg; cjansen@ntu.edu.sg; huanle.zhang@ntu.edu.sg; limo@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TNET.2016.2614707

in rounds. The dissemination process is terminated once all nodes in the network successfully recover the original data by accumulating sufficient encoded packets.

It is challenging to disseminate Fountain-encoded packet over the contention-free pipelines for at least two reasons. First, the pipelined dissemination cannot be interrupted or suspended by Fountain coding. The encoding and decoding of packets need to be integrated into the timing-critical operations of constructive interference and pipelining. Glossy [16] must limit the time offset of concurrent transmissions within  $0.5 \mu s$  to meet the synchronization requirement for generating constructive interference with high probability. Incautious design will lead to uncertainty in data transmission, which may impair the synchronization of constructive interference and paralyze the pipelined data dissemination. Second, a stop condition has to be set and conveyed to the source. Data dissemination should be terminated once all nodes recover the original data object. The source stops pumping packets into the network and the relaying nodes stop forwarding packets to their lower layer. In the pipelined data dissemination, however, all nodes are either receiving packets from their upper layer or relaying the received packets to their lower layer all the time. No explicit feedback channels can be used to convey the acknowledgements from all nodes to the source.

By tackling the above challenges, we make the following three contributions.

1. We design a radio-driven coding scheme where the computation of Fountain encoding and decoding is accommodated into the intervals between the timing-critical operations of highly-synchronized constructive interference and pipelining. Fountain coding and transmission controls are tailored to guarantee a deterministic software delay. By parallelizing data coding and transmissions, we also avoid adding any coding latency in dissemination time.

2. We devise a novel feedback approach in the one-way pipelines to timely acknowledge the successful recovery of the data object and stop the dissemination process. We leverage the silence of channels to aggregate and convey the acknowledgements from the leaf nodes to the source. The threshold of channel silence is adapted dynamically according to the ambient noise and interference. The feedback process is carefully integrated with the pipelined data dissemination to guarantee the reliability of data dissemination. Packet loss is also considered to eliminate its impact on data dissemination and feedback delivery.

3. We apply packet-level adaptation of channel diversity and network density to boost the dissemination efficiency. By adapting the channel allocation and the number of concurrent transmitters for every packet, all nodes experience cycled channel assignments and better wireless diversity. This avoids few poorly-performing nodes to hinder the entire pipelines. To incorporate the adaptation operations into the feedback process and pipelined data dissemination, we design a set of techniques, such as scheduled silence of leaf nodes, fast channel switching and failure recovery.

We implement Pando in Contiki and evaluate its performance on Indriya [5] and Flocklab [29], two large-scale testbeds for wireless sensor networks. In our experiments,

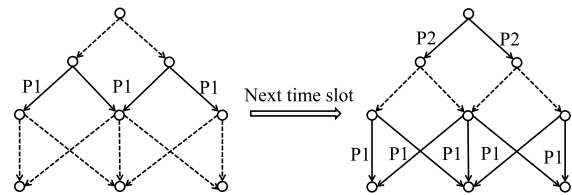


Fig. 1. The pipelined tree built on constructive interference and pipelining. 'P1' and 'P2' represent the first packet and the second packet respectively.

Pando is able to successfully disseminate a data file of 32 kBytes over a network of up to 6 hops within 6 seconds. We compare Pando with state-of-the-art data dissemination protocols, Splash [6] and Deluge [21], over various network configurations. Our experimental results suggest that Pando reduces the average dissemination time of Splash by  $3.5\times$ , corresponding to a reduction factor of  $32.7\times$  over Deluge.

## II. MOTIVATION

The goal of data dissemination is to deliver a bulk of data (e.g., an executable file or an application updating profile) to all the nodes in a network. Any packet loss may make the received data useless. Reliability and dissemination time are thus two key performance metrics for data dissemination. To achieve high reliability, previous protocols, like Deluge [21] and ReXOR [7], transmit the data object hop by hop with explicit acknowledgements and retransmissions. The network resources are significantly underutilized in such a way due to high multiple access overhead and low spatial reuse. By leveraging constructive interference, Glossy [16] allows multiple nodes to simultaneously transmit one packet without backoff or spatial multiplexing. The spatial reuse is further improved by enabling multiple layers transmitting concurrently on distinct channels [38]. Therefore, the data dissemination time could be essentially reduced by combining constructive interference and pipelining [6].

In this section, we experimentally study the data dissemination enabled by constructive interference and pipelining, and reveal its long-tail problem. We also demonstrate the gain and challenges of incorporating Fountain coding into the pipelined data dissemination.

### A. Constructive Interference and Pipelining

Based on constructive interference and pipelining, a tree topology is built, as depicted in Figure 1. One packet is forwarded simultaneously by all nodes at a same layer which interfere constructively with each other; meanwhile, multiple packets can be transmitted concurrently by different layers that make use of distinct channels. In each time slot (corresponding to the transmission time of one packet), all received packets are disseminated one-hop further from upper layers (closer to the source) to lower layers. For the nodes at the same layer, we refer the nodes at their upper layer as their parent nodes and the nodes at their lower layer as their child nodes.

In the pipelined tree, the channel quality varies at different nodes [4], [49] and constructive interference becomes unreliable if the number of concurrent transmitters is large [47].

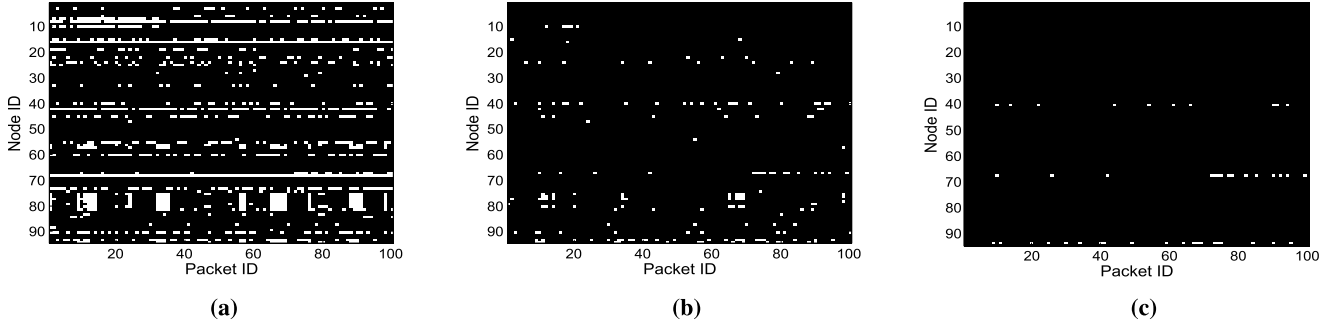


Fig. 2. The packet reception performance of every node after each round in Splash. One black point represents that the packet is correctly received by the relative node. One white point refers to a packet loss. Due to the space limitation, only the first 100 packets (instead of all 500 packets) in each round are presented. (a) First round (85% reliability). (b) Second round (96.8% reliability). (c) Third round (99% reliability).

As a result, some packets may not be received by several nodes, and the data dissemination is not reliable. To the best of our knowledge, Splash [6] is the latest data dissemination protocol that combines constructive interference and pipelining. It disseminates the data object multiple times and relies on a local recovery phase to achieve high reliability. In the first two rounds, it disseminates the same data object twice. In the third round, 500 XOR-encoded packets are disseminated. Every encoded packet is the linear combination of 20 randomly-selected original packets. When a node receives an encoded packet, it can recover a missing original packet only if it has received 19 original packets which were used to generate that encoded packet. After the dissemination over the contention-free pipelines, a local recovery phase is performed. Nodes request the missing packets from their neighbors using CSMA/CA-based multiple access.

### B. The Long-Tail Problem

We conducted a series of experiments with Splash [6] on Indriya [5], the same testbed used in [6]. A data object of 32 kBytes (i.e., 500 packets of 64-byte payload) is disseminated in each experiment. Figure 2 depicts a snapshot example of the packet reception for every node after each round. Figure 2 demonstrates that the network reliability is high (85%) after the first round although the packet reception rate varies for different nodes. The network reliability is calculated as the average reliability of all nodes in the network. From Figure 2 to Figure 2, the second round only increases the reliability from 85% to 96.8%. It is inefficient to repeatedly disseminate *the whole data object* to *all nodes*. On average, for one node, 76.2% of its received packets are duplicate. Figure 2 shows that the reliability improvement of XOR coding is limited (from 96.8% to 99%). 97.8% of the XOR packets cannot contribute to the recovery of original data. The empirical design of Splash (e.g., 500 XOR-encoded packets) deviates from the best setting which requires an optimal number of XOR packets according to the reception of the first two rounds.

Figure 3 shows that the overhead of XOR coding in Splash is high and the long tail problem is severe. The XOR encoding and decoding in Splash (i.e., around 18 seconds) require a large amount of flash memory access and XOR operations

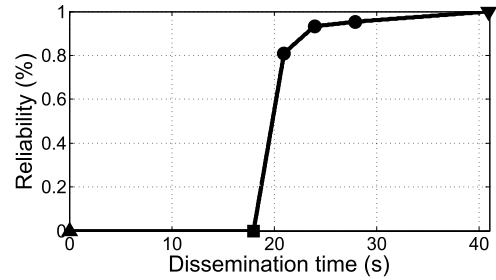


Fig. 3. Network reliability achieved by Splash at some specific time points. Before data dissemination, Splash first needs to generate 500 XOR-encoded packets for the third round. The end of XOR encoding is labeled by a square marker. The end of 3 dissemination rounds is indicated by circle markers. The local recovery phase is from the end of the third round till the last node correctly receives the data object.

among the selected packets (i.e.,  $915 \mu s$  for encoding one packet). To generate a XOR-encoded packet, the source needs to read 20 original packets from the flash memory. In the third dissemination round, all nodes write the received packets into the flash memory. At the end of this round, they read each encoded packet and the relative original packets from the flash memory for decoding. In addition, due to the long tail problem, the total dissemination time of Splash for achieving 100% reliability is more than  $10\times$  larger than the time for achieving 75.9% reliability in the first round. Two obvious reasons can be found in Figure 3. The dissemination in the second and third rounds has limited contributions to the reliability progress. The contention-based local recovery (i.e., around 13 seconds) is time-consuming. As reported in [6], the local recovery phase is much longer than the total transmission time of three dissemination rounds.

### C. Fountain Coding in Data Dissemination

To address the long-tail problem in data dissemination, we use Fountain codes to introduce randomness in the disseminated packets. We disseminate the Fountain-encoded packets along the fast and parallel pipelines. Every encoded packet is a linear combination of several randomly-selected original packets. Regardless of which packets one node has received, every encoded packet provides innovative information to the recovery of the original data. When a node accumulates



sufficient amount of encoded packets, it recovers the original data by simple linear determinant calculation.

By incorporating Fountain coding into the pipelined data dissemination, duplicate retransmissions are avoided and the network capacity can be asymptotically approached as rateless-encoded packets are disseminated over the fast and parallel pipelines. For example, if we replace the data packets in the same trace of the above experiment (Section II-B) with Fountain-encoded packets, we can achieve 100% network reliability by just disseminating 1132 encoded packets, thereby reducing the dissemination time of Splash by  $7.2\times$ .

Although Fountain codes have been implemented in some previous data dissemination protocols [20], [39], they all use Fountain codes to improve the performance of single-hop transmissions. The integration of Fountain coding with pipelined data dissemination is challenging. The highly-synchronized operations of constructive interference and pipelining can be easily paralyzed by incautious design of Fountain coding. During the dissemination, all nodes (except the source) are either receiving or transmitting on the downlinks. They do not have any time to decode a packet or send an acknowledgement to their parent node.

#### D. Simple Integration Does Not Work

A simple way to enable Fountain coding in pipelined data dissemination is to let the source first disseminate a certain number of encoded packets and all nodes begin to decode after the data dissemination. To achieve 100% reliability, however, a local recovery phase has to be performed or the source must know the decoding result of every node. It is troublesome to collect the feedback from all nodes using CSMA/CA-based multiple access and multi-hop routing. After the feedback collection, the source has to resume the data dissemination if any nodes have not received sufficient packets. Such a loop may have to be performed multiple times. The above simple solution cannot eliminate the overhead of multiple access and defies the goal of constructive interference and pipelining. Moreover, the decoding time impose high overhead to the total dissemination time.

Pando is the first practical work which employs Fountain codes for data dissemination with constructive interference and pipelining in large-scale sensor networks. Pando accomplish the encoding and decoding of Fountain code while the data dissemination is ongoing. Nodes perform decoding in parallel with transmissions. Furthermore, Pando stops the data dissemination only if all nodes in the network have successfully recover the original data. The entire data dissemination process is completed over the fast pipelines without any pause.

### III. PANDO

Pando works as a service middleware in the protocol stack to provide fast and reliable data dissemination. A wake-up protocol (details in Section IV) is developed to trigger data dissemination. During the data dissemination process, the general network activities (e.g., data collection or duty cycling) are temporarily suspended, and the sensor mote resources (e.g., CPU and radio) are in active mode. All nodes resume

the previous network activities (e.g., data collection or duty cycling), when they complete the data dissemination. Since Pando's dissemination is fast (6 seconds to disseminate a file of 32 kBytes across a network of 31 nodes), Pando has negligible impact on other network activities. Compared with existing data dissemination protocols that use much longer time for data dissemination, Pando essentially reduces the energy consumption of individual nodes and prolongs the network lifetime.

We execute Pando while a conventional data collection protocols (e.g., CTP [18] for TinyOS and Contiki Collect [23] for Contiki) is running. Pando builds the pipelined tree based on the topology information provided by Contiki Collect. Every node only needs to know its hop count, and the ID of its parent node and its child nodes. Since data collection is a general application of sensor networks, a hierarchical tree is usually maintained throughout the network lifetime. Therefore, the construction of pipelined tree imposes negligible overhead on Pando's data dissemination.

In this section, after a preliminary analysis of Fountain codes, we introduce three techniques that are used to make Pando into a practical system. The Fountain coding in Pando is radio-driven. The coding computation is accommodated into the CPU idle intervals of transmission operations; at the same time, the integration is carefully tailored to guarantee the strict synchronization of constructive interference and pipelining (Section III-B). A silence-based feedback scheme is developed for the one-way pipelines to promptly terminate the source and complete the dissemination process (Section III-C). Packet-level adaptation of network density and channel diversity is adopted to boost the performance of constructive interference and pipelining (Section III-D).

#### A. Preliminaries on Fountain Codes

Fountain codes, like Luby Transform (LT) codes [34] and random linear codes [35], encode a data object of  $k$  packets,  $\{X_1, X_2, \dots, X_k\}$ , into a stream of encoded packets,  $\{Y_1, Y_2, \dots\}$ . One encoded packet is generated by linearly combining a certain number of randomly-selected original packets. Once a node correctly receives  $m$  ( $m \geq k$ ) encoded packets, it can recover the original data using the Gaussian Elimination (GE) or Belief Propagation (BP) decoding algorithm. The coding efficiency is calculated as  $k/m$ . A comprehensive overview of Fountain codes can be found in [35].

Pando is not limited to any specific Fountain codes. In our current implementation, we choose LT codes in consideration of its succinct design (i.e., low computation overhead). Although Random Linear (RL) codes and Raptor codes may provide higher coding efficiency, they involve intensive computation. RL code generates encoded packets using modular multiplication in Galois Field (GF)  $2^8$  and Raptor code applies a fixed error correction code before LT coding. In LT codes, every encoded packet is calculated by performing XOR operations of  $d$  packets that are randomly chosen from  $k$  original packets, where  $d = 1, 2, \dots, k$ . For an encoded packet  $Y_i$ ,  $1 < i < \infty$ , the degree  $d$  is determined by a distribution  $\rho(d) = P_d$ , where  $P_d$  is the probability that  $d$  original packets are selected to encode  $Y_i$ .

Although the BP algorithm is more computationally-light, we use the GE algorithm, because the latter provides higher coding efficiency, i.e., it can successfully decode the received packets as long as the coefficient matrix of the received packets has full rank. The computation of GE is decomposed into small pieces and performed in parallel with data transmissions. The default robust Soliton degree distribution produces low coding efficiency for the applications of sensor networks with small number of packets. We implement an optimized degree distribution proposed in SYNAPSE++ [39], which produces high coding efficiency for small data objects. Like in our previous work [9], we use a best seed for the random number generator that produces the lowest decoding overhead. Since the coefficient matrix can be generated beforehand, we eliminate the time consumption of random number generation in the decoding process.

### B. Radio-Driven Coding Scheme

We perform the encoding and decoding of LT codes along with the timing-critical operations of highly-synchronized constructive interference and pipelining. Since the microcontroller is mostly in idle mode while the radio is receiving or transmitting, the microcontroller can execute the coding computation in parallel with the transmissions performed by the radio hardware. The goal of our design is to find the largest idle intervals of the microcontroller for Fountain coding, under the precondition of controlling the synchronization offset between the concurrent transmissions of constructive interference within  $0.5 \mu s$ . Thus, we need to guarantee that the generated idle CPU intervals are constant and the coding computation occupies deterministic time.

*Constant Intervals of Idle CPU:* We generate two large constant idle intervals in the operations of constructive interference and pipelining based on two observations. First, the time to read/write one byte from/into the radio through the Serial Peripheral Interface (SPI) bus (e.g.,  $5.75 \mu s$  for general low-power microcontrollers) is much shorter than the latency of transmitting/receiving one byte by the radio hardware (i.e.,  $32 \mu s$  for IEEE 802.15.4-compliant transceivers). Second, although the operations of constructive interference and pipelining are timing-critical, they only need to execute a few CPU commands at some important time points, e.g., sending commands to set the radio to the right state when an interruption is received. Therefore, after reading/writing one packet from/into the radio and executing the commands, the microcontroller goes into idle mode for a long time that can be used for Fountain encoding and decoding.

We design Pando on top of the flooding protocol in Glossy [16] and the pipelining of PIP [38]. In Glossy, hardware interrupts are used to implicitly synchronize the concurrent transmissions. When multiple nodes receive a packet simultaneously, they detect the interrupt for the end of the reception at the same time. Based on the interrupt, their microcontrollers immediately set their radios to TX mode and write the data into the TX buffer of the radios. They then enter idle mode till the end of the transmission. After the transmission, the microcontrollers also detect an interrupt and set the radios to

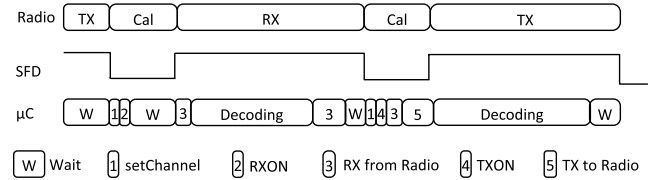


Fig. 4. State transition of the radio and the microcontroller.

RX mode to receive the next incoming packet. They go into idle mode for a long duration before the end of the reception. Since the transmission time of one packet and the operations of constructive interference are predefined, the idle intervals are constant for every packet transmission.

Figure 4 presents the key timeline of Pando. When the microcontroller receives an interrupt at the end of a transmission (the SFD pin is set to low), it immediately sets the radio to RX mode by sending two commands (i.e., channel setting and RXON, command 1 and 2 in Figure 4) to the radio. After calibration ( $192 \mu s$ ), the node receives the first two bytes (Glossy header for synchronization) of a packet. Instead of reading the following bytes one by one as Glossy, it begins to decode the previously-received encoded packet for a certain duration (e.g.,  $1688 \mu s$  in our implementation). It then resumes reading the received data from the radio and switches to waiting state during the guard interval. By detecting an interrupt for the end of a packet reception (the SFD pin is set to low), the microcontroller sets the sending channel and sends a TXON command (command 4 in Figure 4). After that, it reads the last eight bytes of the received packet from the RX buffer of the radio and writes the whole received packet to the TX buffer of the radio. Then, the microcontroller is able to perform Fountain coding before the radio begins to transmit the last eight bytes.

To make sure that the coding computation in the constant idle interval and the following hardware interrupt do not impact each other, a guard interval, corresponding to the transmission time of 8 bytes, is added before the end of each interval. A timer is set at the beginning of each idle interval. When it expires, the coding procedure saves the decoding context and resumes its operation in the next idle interval.

*Deterministic Coding Time:* To be safely inserted into the constant idle intervals, the coding computation should be decomposable and each decomposed piece should occupy deterministic time. We begin to decode once two encoded packets are received. New received packets can be added into the decoding process incrementally. Upon receiving a packet from its parent nodes, a node has two time slots to decode that packet before the next packet is received. In these two slots, the node transmits that packet to its child nodes and receives the new packet from its parent nodes respectively. During two time slots, the microcontroller tries to complete the decoding of the received packet in its idle intervals.

The Accumulative Gaussian Elimination (AGE) algorithm [9] is used for incremental decoding. Although the BP algorithm is less computationally expensive, AGE provides higher coding efficiency (e.g., a coding efficiency of 89% for a

data object of 16 original packets). AGE transforms the top-left square submatrix of the coefficient matrix into an identity matrix when few packets are received. As new packets are received, AGE extends the submatrix gradually. The original data is recovered when the size of the submatrix becomes equal to the number of original packets.

At the end of each idle interval, if the decoding of the last received packet is not finished, the decoding algorithm saves the context by recoding the intermediate variables and some decoding results, including the size of the submatrix, the coefficient matrix and the calculated results of data packets. When the next idle interval begins, the decoding process is resumed. Therefore, the decoding computation occupies deterministic time which ends before the guard interval. In our implementation, the context saving costs at most  $64 \mu s$ , which is much smaller than the guard interval ( $256 \mu s$  for the IEEE 802.15.4 standard).

The encoding processing can also be performed incrementally. The computation of encoding one packet is much less than the decoding complexity. The source is able to generate one encoded packet every two time slots.

**Results:** We have implemented the radio-driven coding scheme in Contiki for TelosB sensor motes, a typical sensor mote type that has been widely used in current sensor network deployments. A TelosB sensor mote includes a TI CC2420 radio which supports the IEEE 802.15.4 standard. The packet size in Pando is set to 73 bytes, including 64-byte payload, 7-byte MAC header and 2-byte footer. Except the 8-byte guard interval, the transmission time of the first 65 bytes is  $2080 \mu s$ . It only takes  $392 \mu s$  to read these bytes from the radio into the microcontroller via the SPI bus. Therefore, the microcontroller has  $1688 \mu s$  for decoding during the reception of one packet. In the same principle, the microcontroller has a similar idle interval when the radio is transmitting.

The radio-driven coding scheme is a general solution to enable Fountain encoding and decoding in pipelined data dissemination. Pando has no special requirements on the underlying hardware and can be implemented on any hardware platforms that support the implementation of constructive interference in Glossy [16]. Glossy successfully demonstrated the feasibility of realizing constructive interference on IEEE 802.15.4-compliant hardware platforms. Glossy can also be easily ported to other radio platforms, like the ones feature a RAM for packet buffering.

### C. Silence-Based Feedback Scheme

As the source continuously pumps the Fountain-encoded packets into the network, Pando approaches 100% reliability automatically over the fast and parallel pipelines. Based on the radio-driven coding approach, all nodes can successfully decode the received packets when they are disseminating data packets. To promptly terminate the source and complete the dissemination process, every node needs to send an acknowledgement to the source when it correctly recovers the original data object. In the pipelined data dissemination, however, nodes are either receiving new packets from their parent nodes or relaying the received packets to their child nodes. The acknowledgement transmission of one node should not

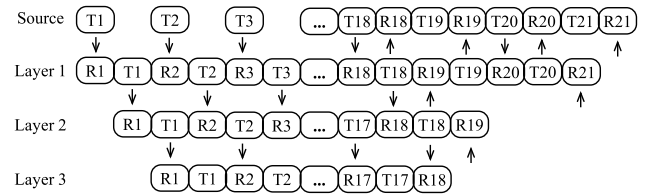


Fig. 5. Dissemination process of a 16-packet data object in Pando. ‘T1’ refers to the first TX slot. ‘R1’ represents the first RX slot. The arrow indicates the signal transmission direction.

impact its role as a forwarder or the synchronized transmissions of other nodes. A feedback scheme is needed to convey the acknowledgement of every node to the source during the data dissemination process.

A silence-based feedback scheme is devised for the one-way parallel pipelines. With this feedback scheme, the acknowledgement of every node is delivered from the lower layers, merged at the relay node, and finally reaches the source. When a node succeeds in decoding, it can generate the Fountain encoded packets by itself using the recovered packets and continue transmitting the encoded packets to its child nodes without receiving new packets from its parent nodes. The successful nodes can thus monitor the transmission of their child nodes in the RX slots. In addition, the nodes at upper layers are likely to recover the original data earlier than the nodes at lower layers, since all packets are forwarded from the source to the lower layers. Therefore, the nodes at upper layers are likely to be ready for detecting the feedback before their child nodes succeed in decoding.

In Pando, a leaf node stops transmitting when it successfully recovers the original data. Its parent nodes (i.e., relay nodes) can detect such silence by measuring the Received Signal Strength Indicator (RSSI) value on the sending channel of that leaf node. A relay node terminates transmitting, when all its child nodes become silent, corresponding to a RSSI value lower than a threshold ( $RSSI_{threshold}$ ). The silence of a relay node indicates that both itself and its child nodes have successfully recovered the original data object. In such a way, the acknowledgements from all child nodes are aggregated at that relay node. The fused feedback can be further conveyed to the source across the network.

**Loss of Synchronization:** In our experiments using the above feedback scheme, some nodes stop working during dissemination. It is because the transmissions are always triggered by the lower layer and the time offset among the concurrent transmissions increases. We solve this problem by requiring the relay nodes to synchronize back with their parent nodes periodically. Since the dissemination is triggered from the source, the downlink packets have more precise synchronization information and are used to calibrate the relay nodes. In our current implementation, nodes listen to the feedback channel only in the odd RX slots.

As an example, Figure 5 demonstrates the dissemination process of a 16-packet data object in Pando. The encoded packets are disseminated from the source to the other nodes layer by layer. Every node needs 17 encoded packets (one extra packet is the coding overhead) to recover the original data.



In this example, we assume no packet loss occurs. All nodes correctly recover the original data object when the 17th packet is received. When the leaf nodes at the third layer receive the 17th packet, after two time slots, they succeed in decoding and stop transmitting. The relay nodes at the second layer detect the channel silence in their 19th RX slot and terminate their dissemination. The fused feedback of silence is captured by the nodes at the first layer in their 21st RX slot and finally reaches the source. The whole dissemination process is terminated after the 21st RX slot of the source. Figure 5 shows that the silenced-based feedback scheme is efficient. Along the path, the relay nodes only need 1 or 3 slots to forward the aggregated feedback to their parent nodes. In this three-hop example, five slots (13.6 ms) are sufficient for the feedback aggregation and delivery after the last leaf node succeeds in decoding.

*False Negative of Channel Silence:* Due to ambient noise or interference from nearby devices using the same frequency band, a node may not be able to detect the channel silence, when all its child nodes succeed in decoding and stop transmitting. To eliminate the false negative of channel silence, we dynamically adapt the RSSI threshold of channel silence ( $RSSI_{threshold}$ ) according to the noise and interference condition. Generally,  $RSSI_{threshold}$  must be larger than the instant noise level. Between the wake-up protocol and the data dissemination, we leave 10 time slots, during which all nodes measure the average RSSI of free channels (denoted as  $RSSI_{silence}$ ). During data dissemination, when a node successfully receives a packet, it measures the channel RSSI and updates the RSSI value of successful packet reception (denoted as  $RSSI_{rx}$ ). In our experiments on two large-scale testbeds deployed in both indoor and outdoor environments,  $RSSI_{rx}$  is always larger than  $RSSI_{silence}$ . We set  $RSSI_{threshold}$  as  $(RSSI_{silence} + RSSI_{rx})/2$ . If some noise or interference appears,  $RSSI_{rx}$  increases and  $RSSI_{threshold}$  becomes thus large. When the child nodes are not transmitting, with such a high  $RSSI_{threshold}$ , the channel is measured as silent even in the appearance of noise or interference.

*False Positive of Channel Silence:* When a node fails to receive a packet, it cannot forward that packet. Its parent node may interpret this silence as successful decoding. To filter out such false silence, every node only infers that all its child nodes have completed the reception when  $M$  consecutive silent slots are detected. In practical,  $M$  is small based on three facts. First, even if a node cannot transmit, its parent node is likely to detect a high RSSI by the signal from the other child nodes at the same layer. Second, it is rare that a node cannot receive the packet from its parent node consecutively, especially when our packet-level adaptation of network density and channel allocation is applied. Third,  $RSSI_{threshold}$  cannot be too high to cause the false positive of channel silence, as it cannot be larger than  $RSSI_{rx}$ , which is measured when the child nodes are transmitting. In our implementation,  $M$  is set to 3, which can filter out all false silences in a large-scale sensor network that is densely deployed in a building. We will show in Section V that the overhead of the feedback process is small compared with the significant improvement of data dissemination efficiency.

#### D. Adaptation of Channel Allocation and Network Density

In pipelined data dissemination, some nodes may have poor packet reception due to bad channel quality or unreliable constructive interference. Some channels may be lossy at some node locations because of ambient noise or interference from other nodes or devices. The bad channel can be replaced by exchanging the assigned channel among nodes. On the other hand, constructive interference may not work well at some nodes as the synchronization offset is high. The performance of these nodes can be improved by changing the set of concurrent transmitters, since constructive interference is likely to be strong for a small number of concurrent transmitters [47] and the capture effect occurs at some locations of concurrent transmitters [16]. With capture effect, a radio is able to receive one signal despite the interference from other transmitters, when the signal is stronger and arrives earlier than the interference [25]. Splash [6] changes the channel allocation and network density after each dissemination round. Such round-level adaptation, however, cannot effectively adapt to the channel variations. Few poorly-performing nodes may hinder the entire pipelines for a whole round. Moreover, every node can only experience three different channels for transmission or reception at most in the three dissemination rounds respectively.

Pando enables packet-level adaptation of channel allocation and network density. Pando allocates every packet with a specific channel and changes the number of concurrent transmitters for every packet transmission. Nodes set their RX/TX channel according to the packets they are going to receive or transmit. Packet-level adaptation can only be enabled in the data dissemination of Fountain-encoded packets. As the encoded packets are independent to each other, the loss of an individual packet caused by unreliable constructive interference or bad channel at some nodes can be immediately compensated by the following packets disseminated with different channels and network densities. Compared with round-level adaptation, Pando avoids that some poorly-performing nodes hinder the dissemination process for a long time. Moreover, every node circularly experiences all available channels which fully exploits the wireless diversity.

A set of techniques are designed to incorporate the adaptation operations into the pipelined data dissemination and the silence-based feedback process. In order not to impact the synchronized transmissions of constructive interference and pipelining, channel switching is divided into two steps, i.e., calculating which channel to use for the next packet and setting the channel in the radio. The first step is executed in the idle interval of the microcontroller and the last step introduces constant software delay. A recovery mechanism is adopted to address the failure caused by packet losses. The transmissions of leaf nodes are scheduled according to the feedback scheme.

*Channel Allocation:* Figure 6 presents the data dissemination process of packet-level channel allocation. Every packet is assigned with a unique channel. Four channels (i.e., Channel 15, 20, 25, 26 in the IEEE 802.15.4 standard) that experience light link correlations [4] are used circularly. At the beginning, all nodes in the network are set to a default channel to receive the first packet. After forwarding

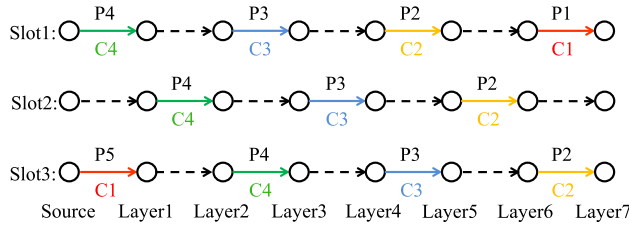


Fig. 6. Packet-level channel allocation. ‘P1’ represents the first disseminated packet. ‘C1’ indicates the first channel. From slot 1 to slot 2, all packets in the network are forwarded one hop further by the nodes of different layers with distinct channels.

the received packet to their child nodes, they immediately switch to the channel allocated for the next packet. When the relay nodes are forwarding packets, the source transmits a new packet. The minimum distance between two packets using a same channel is at least eight hops, which are large enough to avoid the intra-path interference in typical sensor networks [38].

Channel switching is performed at the end of each transmission. To ensure that it produces deterministic software delay, as shown in Figure 4, channel switching is enabled by sending two commands (i.e., channel setting and RXON) to the radio immediately when the SFD interrupt for the end of transmission is detected. The channel of the next packet is calculated according to the slot sequence. The computation is short (addition of two integers) and performed before the microcontroller goes to waiting state during the guard interval. It does not cause any software delay to channel switching.

If a node misses a packet because the signal is not detected or the CRC checking fails, it stays at the same channel and tries to overhear that packet transmitted by the other nodes at the same layer. Before the overhearing, the node starts a RX timer to record a time slot. If the overhearing fails, it can still switch the channel to the new coming packet from the parent nodes in time. When multiple packets are missing consecutively, the node is also able to set to the right channel by counting the slot sequence using the RX timer. The slot sequence is calibrated frequently when a packet is received. The racing problem occurs if a node begins to receive after its parent nodes transmit. Such misalignment becomes worse as more time slots pass. To solve the racing problem, the RX timer is set slightly shorter than a time slot. In our implementation, nodes switch to the next channel 50  $\mu$ s earlier than the start time of the next transmission.

**Network Density:** In Pando, although leaf nodes do not have any child nodes, they transmit the received packets to enable the overhearing of the nodes at the same layer. The transmissions of leaf nodes do not impose any overhead to the data dissemination time, since they use a unique channel and happen at the same time with other transmissions.

To adapt the network density at packet level, the leaf nodes transmit a received packet only when the sequence number of that packet is even. For the received packets with odd sequence number, the leaf nodes do not transmit them. Because more than 50% of nodes in a data collection tree are leaf nodes [6], disabling the transmissions of leaf nodes for odd packets can significantly reduce the number

of concurrent transmitters. In addition, if a node does not receive an even packet correctly, it does not transmit either. Since the set of successful receivers varies for each packet, the set of concurrent transmitters also changes for the following transmission of that packet. Therefore, the network density is altered for every packet and the reliability of constructive interference is improved.

The leaf nodes do not transmit in the time slots of odd packets. When considering the feedback scheme, these silent slots may be wrongly detected as successful decoding by the parent nodes. Our integrated design of the network density adaptation and the feedback scheme avoids such misunderstanding. In the time slots of odd packets when the leaf nodes are not transmitting, the parent nodes are receiving packets from the upper layer and do not listen to their child nodes for feedback detection. As in Figure 5, the leaf nodes only transmit in their even TX slots, when their parent nodes are listening on their channel.

#### IV. IMPLEMENTATION

We implement Pando on TelosB sensor motes with Contiki operating system. In this section, we introduce some technical details which enable the Pando implementation.

**Wake-up Protocol of Pando:** To incorporate Pando’s data dissemination with the general network activities of sensor networks, before data dissemination, the source first broadcasts a start command across the network multiple times. The reception probability of such a short command (4 bytes) over a multi-hop sensor network of constructive interference is more than 99.99% for 6 retransmissions [16]. In our implementation, the start command is transmitted 20 times and it only takes tens of milliseconds. After the dissemination of the start command and the silent interval of 10 time slots, the source begins to transmit the first encoded packet. The data dissemination task has the highest priority rather than the other tasks, like sensing or local transmissions. All nodes set the channel to the default channel to receive the first disseminated packet, once they receive the start command.

The general applications of sensor networks usually work on a single channel. The channel 26 of the IEEE 802.15.4 standard is widely adopted in real deployments of sensor networks, since it is not overlapping with the channels of Wi-Fi [42]. We disseminate the start command on channel 26 (the default channel in Pando). The start command includes information about the data object size.

**Page-Based Dissemination:** We divide the data object (e.g., 32 kBytes) into small pages (e.g., 2 kBytes in our current implementation) and disseminate the pages sequentially due to the memory constraints of wireless sensor motes (e.g., 10-kByte RAM memory on TelosB). The source begins to disseminate a new page when the current page has been correctly received by all nodes. The source knows the dissemination of one page has terminated thank to the silence-based feedback scheme. When a node succeeds in decoding and completes its feedback task, it switches to the default channel and waits for the packets of the next page. If the dissemination of the current page is not completed, the successful nodes may receive some packets from some other nodes that are



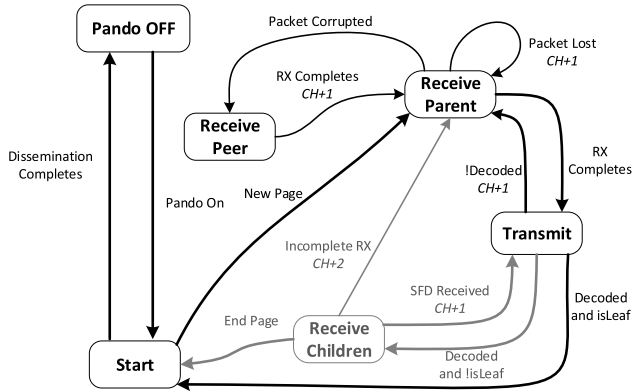


Fig. 7. State machine of relay nodes and leaf nodes in Pando. Relay nodes have all the functions of leaf nodes (in black) and can also receive packets from their child nodes (in gray).

still transmitting packets. They check the sequence number of the received packets. If it is larger than the last packet they received, they infer that the packets are from the current page; otherwise, they start the dissemination of a new page.

*State Machine of Nodes:* Figure 7 illustrates the state machine of the nodes, including both relay nodes and leaf nodes. Besides all the functions of leaf nodes, relay nodes also need to listen to the channel of their child nodes. When a relay node detects the silence of all its child nodes, it becomes a leaf node. The bold arrows indicate how a relay node performs in an ideal case. At first, all nodes receive packets from their upper layer and immediately relay the packets to their lower layer. When a node successfully recovers the original data, it performs the following four operations sequentially: receiving a packet from its parent node, forwarding that packet to its child nodes, listening to its child nodes, and transmitting a self-generated encoded packet to its child nodes. It switches to the next page when it detects that all its child nodes are silent. Upon the completion of one page, the source starts a new page or terminates the whole dissemination process.

We introduce several failure recovery mechanisms to cope with packet losses. As described in Section III-D, if a packet from the upper layer is corrupted, the node stays in the same channel and tries to receive that packet from the other nodes at the same layer. After that, the node switches to the next channel for the coming packet from its upper layer. If no SFD interrupts are detected on the feedback channel, the node loses synchronization and goes back to listen to its parent node by setting the channel accordingly. The node follows the right circulation of channels by using a RX timer to count the dissemination slots.

*Safe RX Buffer Reading:* In the reception slots, the radio-driven coding scheme must perform Fountain coding and accomplish reading the received data (65 bytes) from the radio. After the RXON command, the microcontroller sets a coding timer of 1688  $\mu s$  and begins to decode. When the timer expires, with the multiple SPI access, the microcontroller initializes the SPI bus once and reads all the 65 bytes consecutively. At the end of reading, however, the microcontroller may try to read the last few bytes when they have not been written in the buffer. To handle the underflow problem of RX buffer, we read the last five bytes individually using single SPI access.

TABLE I  
DISSEMINATION TIME (SECOND) ON INDRIYA<sup>1</sup>

Test #	Size (hops)	Pando	Splash w/o encoding	Splash
1	7	9.7	22.3	40.2
2	8	13.3	22.5	40.4
3	9	10.4	21.6	39.4
4	8	12.4	21.9	39.7
5	7	11.0	22.9	40.8
Average	8	11.4	22.3	40.1

As in Glossy [16], before each reading, the microcontroller keeps checking the FIFO pin that indicates whether the RX buffer of the radio is empty or not. It only begins to read one byte from the radio when the FIFO pin turns to high.

## V. EVALUATION

We conduct a series of experiments on two large-scale open testbeds of wireless sensor networks, Indriya [5] and Flocklab [29]. Both testbeds deploy sensor motes inside a building and operate on 2.4 GHz. Indriya currently has 99 TelosB sensor motes with dense deployment. In Flocklab, 31 Tmote Sky sensor motes are sparsely deployed with low network density and some sensors are installed outdoors. The packet size is set to 64 bytes for most of the experiments and the data object is 32 kBytes (i.e., 500 packets).

*Protocols:* In our experiments, we compare Pando with Deluge [21] and Splash [6], of which we could find the source code. They are two state-of-the-art data dissemination protocols. Deluge is the first reliable data dissemination protocol developed in 2004. Many protocols have been devised in the last decade to improve its performance, such as MC-Deluge [51], Rateless Deluge [20] and MT-Deluge [17]. They all disseminate a data object in small pages hop by hop using a handshake mechanism and CSMA/CA-based multiple access. In 2013, Splash is designed to partially eliminate the contention overhead using constructive interference and pipelining. It is the latest data dissemination protocol with the best reported performance. For the other data dissemination protocols that we cannot find the source code, we compare Pando with them by literature studies.

*Metrics:* We evaluate the data dissemination protocols using four metrics: reliability, dissemination time, energy consumption and memory cost. Pando's dissemination time is measured from the start of protocol (e.g., transmission of the start command) to the termination of the source. Since Splash and Deluge do not have an explicit termination time, we monitor the traces of all nodes and calculate the dissemination time as the last node correctly receives the data object.

### A. Performance Comparison

Table I and Table II show the dissemination time of three protocols on Flocklab and Indriya respectively. For each experiment, we run Deluge, Splash and Pando separately.

<sup>1</sup>We encountered several technical problems for running Contiki's Deluge on the remote testbed Indriya. The experiments of Contiki's Deluge succeeded on Flocklab and the results in Table II show that the dissemination time of Contiki's Deluge is orders of magnitude larger than Pando.

TABLE II  
DISSEMINATION TIME (SECOND) ON FLOCKLAB

Test #	Size (hops)	Pando	Splash w/o encoding	Splash	Deluge (2 kBytes)
1	6	5.7	18.4	36.4	481.4
2	5	4.3	18.3	36.3	471.7
3	9	7.7	18.4	36.4	481.4
4	6	6.4	18.3	36.3	481.4
5	5	4.4	18.6	36.5	466.9
Average	6	5.7	18.7	36.5	476.6

For each protocol, five tests are conducted with different locations of the source node. Before the data dissemination, Splash needs to first generate the XOR-encoded packets for the third round. Splash spends about 18 seconds to encode 500 XORed packets on TelosB with TI MSP430 microcontroller. In Table I and Table II, we also present the dissemination time of Splash without encoding in which the time spent on encoding is not included in the dissemination time of Splash. Pando does not have any extra coding overhead since it can encode and decode during data dissemination.

The experiment results reveal that all three protocols can achieve 100% network reliability. It takes Pando 11.4 seconds and 5.7 seconds to disseminate a file of 32 kBytes on Indriya and Flocklab respectively. Splash needs about 40.1 seconds and 36.5 seconds. Pando reduces the dissemination time of Splash by an average factor of 3.5 and 6.4 on Indriya and Flocklab, corresponding to 71.6% or 84% reduction respectively. Even not considering the encoding time in Splash, the reduction factor achieved by Pando over Splash is 2.0 and 3.3 on Indriya and Flocklab.

Deluge has two versions: TinyOS's DelugeT2 and Contiki's Deluge. We use Contiki's Deluge, since it is difficult to execute TinyOS's DelugeT2 on a remote testbed, which requires to send the disseminated data file and some commands from the connected PC to the source node. In our experiments on Flocklab, Contiki's Deluge spends 476.6 seconds to disseminate 2 kBytes, corresponding to 7625.6 seconds for 32 kBytes. Pando and Splash reduce the dissemination time of Deluge by 1337.8 $\times$  and 407.8 $\times$  respectively. Since the implementation of Deluge in Contiki is less efficient than DelugeT2, we calculate the reduction factor over Deluge based on the result of DelugeT2 (524 seconds) reported in [6]. The authors in [6] did the experiments on Indriya with 139 sensor nodes. At present, only 99 sensor nodes are available on Indriya. The dissemination time of DelugeT2 on Indriya is approximated as 373.2 seconds (i.e.,  $524 * 99/139$  seconds). The reduction factor over DelugeT2 achieved by Splash and Pando is 9.3 and 32.7 respectively. Splash has improved the performance of Deluge significantly. Pando further improves the gain of Splash by 3.5 $\times$ .

For the experiment results on Flocklab in Table II, Pando produces higher improvement to the previous two protocols than the results on Indriya, since the sensor nodes of Flocklab are more sparsely deployed and constructive interference performs better with small number of concurrent transmitters. From Flocklab (Table II) to Indriya (Table I), although the number of sensor nodes increases more than 2 $\times$ ,

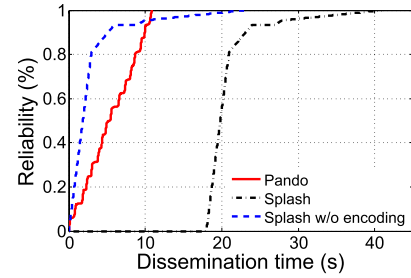


Fig. 8. Network reliability progress of Pando and Splash on Indriya.

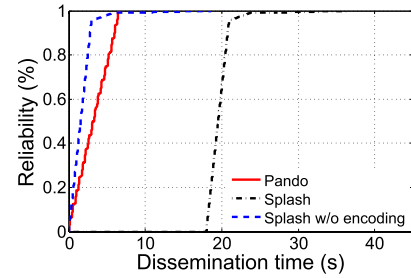


Fig. 9. Network reliability progress of Pando and Splash on FlockLab.

the dissemination time of both Pando and Splash only increases less than 1 $\times$ . Based on constructive interference, the delivery of one packet from one layer to another layer can be completed by just one transmission. With pipelining, the transmissions on different layers can occur simultaneously. Pando makes the gain of constructive interference and pipelining bloom in data dissemination, since it completes the entire data dissemination process over the fast parallel pipelines.

*Reliability Progress:* Figure 8 and Figure 9 depict the progress of the average network reliability achieved by Splash and Pando on Indriya and Flocklab respectively. The experiment results reveal that Splash has high XOR encoding overhead and long tail problem. Pando completes the data dissemination even before the end of encoding of Splash. Even if we do not count the encoding time in the data dissemination time for Splash (i.e., Splash w/o encoding), it still has the long-tail problem. Pando eliminates the long-tail problem in pipelined data dissemination and reduces the dissemination time of Splash by 2.0 $\times$  and 3.3 $\times$  on Indriya and Flocklab respectively. As studied in Section II, the long tail is caused by three limitations of Splash, including duplicate disseminations, inefficient XOR coding and time-consuming local recovery. Pando overcomes the long tail problem by incorporating Fountain coding in data dissemination. Although the reliability progress rate of Pando is lower than Splash (i.e., Splash w/o encoding) in the first round due to the feedback collection for every data page, Pando achieves much shorter total dissemination time than Splash.

*Reliability of Individual Node:* Figure 11 demonstrates the reliability of individual nodes. The encoding overhead is not included in the dissemination time of Splash. The dissemination time of all protocols is when all nodes achieve 100% reliability. It is the time that the last node (the last row) turns to black. On Flocklab, Pando completes the data dissemination within 5.7 seconds when the source receives the aggregated

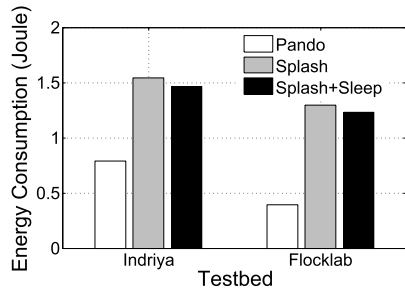


Fig. 10. Energy consumption of Pando and Splash on Indriya and FlockLab.

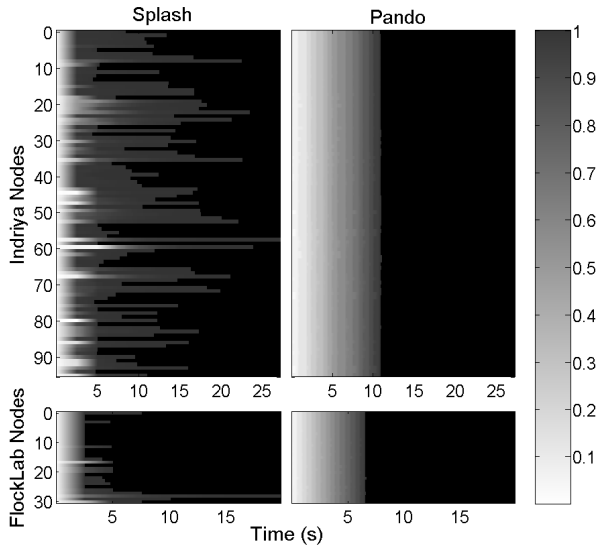


Fig. 11. Reliability of individual nodes achieved by Pando and Splash on Indriya and FlockLab.

acknowledgement from all nodes in the last page dissemination. Splash's data dissemination ends at 18.7 seconds when the last node successfully receives the data object. The dissemination speed of Pando is  $3.3\times$  faster than Splash. Pando completely eliminates the contention-based multiple access overhead, but Splash has to disseminate the data object multiple times and rely on the local recovery to achieve 100%. Moreover, in Pando, all nodes present a similar progress rate. The packet reception rate is more uniformly distributed across all the nodes, since the network density and channel allocation are adapted for every packet transmission.

**Energy Consumption:** Figure 10 presents the energy consumption of Pando and Splash on Indriya and FlockLab. Wireless sensor nodes are typically working in duty-cycled mode, which reduces their energy consumption by setting them in low-power state periodically. Since the dissemination in the duty-cycled mode is extremely time-consuming due to the long-preamble transmissions [14], current data dissemination protocols [6], [17], [20], [21], [39] keep the hardware resources on sensor nodes (e.g., microcontroller and radio) in active state during the data dissemination process. The power consumption of Pando is the same as the previous protocols, since all of them keep the nodes in active mode. Therefore, in Figure 10, Pando reduces the energy consumption of Splash by the same reduction factor as the dissemination time, i.e.,  $2.0\times$  and  $3.3\times$  on Indriya and Flocklab.

TABLE III  
REDUCTION FACTOR OF DISSEMINATION TIME OVER DELUGE ACHIEVED BY THE EXISTING DATA DISSEMINATION PROTOCOLS

Protocols	Number of nodes	Data size (KB)	Reduction factor
MNP [24](2005)	100	5	1.21
MC-Deluge [53](2005)	25	24.3	1.60
Rateless Deluge [19](2008)	20	0.7	1.47
ReXOR [8](2011)	16	4	1.53
ECD [9](2011)	25	10	1.44
MT-Deluge [16](2011)	20	0.7	2.42
SYREN [1](2013)	21	0.5	1.6
Splash [7](2013)	139	32	9.3
Pando	99	32	32.7

The Fountain coding in Pando is performed during the idle intervals of the microcontroller and does not impose any extra energy consumption. The default design of Splash or the other data dissemination protocols do not exploit the idle interval or set the microcontroller to a power-saving mode during such idle intervals. Even if we enable Splash to sleep during the idle intervals, the saved energy is very limited, i.e., 5.2% in Figure 10. It is small, compared with the energy saving gain of Pando over Splash (69.5%). For the data dissemination protocols of constructive interference and pipelining, including Splash and Pando, the transceiver is always transmitting or receiving packets during the whole data dissemination process. The energy consumption of sensor nodes is dominated by the transceiver. For example, the power consumption of the transceiver on TelosB is 69 mW ( $23\text{mA}\times 3\text{V}$ ), which is much larger than the power consumption of the microcontroller, i.e., 5.4 mW ( $1.8\text{mA}\times 3\text{V}$ ). Even if we put the microcontroller to a power-saving mode ( $0.002\text{mA}\times 3\text{V}$ ) in the idle interval, the saved energy is limited.

**Comparison With Other Protocols:** We compare Pando with other data dissemination protocols. As in [6], we refer to the performance reported in the papers of the existing works [7], [8], [17], [20], [24], [51]. All the previous protocols compare their performance with Deluge [21], the first reliable data dissemination protocol for wireless sensor networks. Based on the reduction factor of dissemination time over Deluge reported in the papers of these protocols, we can calculate the relative gain of Pando over other existing protocols.

Table III tabulates the reduction factor of dissemination time produced by the previous data dissemination protocols and Pando over Deluge. The protocols developed before Splash can only provide a reduction factor around 1.6 (maximum 2.4) over Deluge, because they all are based on the contention-based multiple access. Pando and Splash are orders of magnitude faster than Deluge, since they are based on constructive interference and pipelining. By fully exploiting the fast and parallel pipelines as well as the packet-level adaptation of network density and channel allocation, Pando asymptotically approaches the network capacity and reduces the dissemination time of Splash by more than  $3.5\times$ .

**Memory Cost:** Pando uses 6.24 kBytes of RAM, 32 kBytes of data flash memory, and 31.49 kBytes of ROM (program flash memory). TelosB is composed of 10 kBytes of RAM,



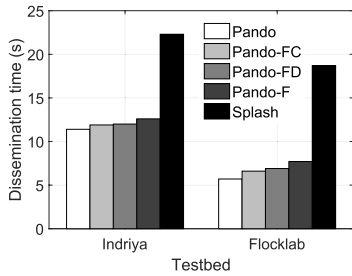


Fig. 12. Contribution of individual techniques in Pando.

1 MBytes of data flash memory, and 48 kBytes of ROM. Pando uses about 2.24 kBytes of RAM to store the received packets of current page. Without temporarily storing of encoded packets in the flash memory, Pando minimizes the data memory cost and significantly reduces the number of data flash memory access. Splash has to store the received packets in the data flash memory and read them into RAM for post-processing after the 3 dissemination rounds.

### B. Contribution of Individual Techniques

To further analyze the gain of Pando, we execute Pando in several versions, in which only some individual techniques of Pando are enabled. The results are shown in Figure 12. In Pando-F, only Fountain coding is enabled, and the packet-level adaptation of both network density and channel allocation is disabled. In Pando-FC, besides Fountain coding, we also enable the packet-level adaptation of channel allocation. Similarly, in Pando-FD, the packet-level adaptation of network density is enabled. Pando in Figure 12 represents the full version of Pando with all techniques.

**Fountain Codes:** According to Figure 12, Pando-F can reduce the dissemination time of Splash by  $1.8\times$  and  $2.6\times$  on Indriya and Flocklab respectively. The fundamental gain of Pando comes from the integration of Fountain codes with the pipelined data dissemination. With Fountain code, Pando disseminates independent Fountain-encoded packets along the fast and parallel pipelines during the whole dissemination process. We improve the utilization efficiency of the pipelined data dissemination and also eliminate the overhead of contention-based local recovery. To tailor LT coding into the highly-synchronized communication on the resource constrained sensor motes, we disseminate the data object in small pages, which introduce some overhead due to the feedback collection for each page. In our current implementation, we adopt a page size of 32 packets. The gain of Pando can be further improved by enlarging the page size with more powerful hardware.

**Packet-Level Adaptation of Channel Allocation:** By comparing the results of Pando-FC with Splash in Figure 12, we find that Pando with Fountain codes and packet-level adaptation of channel allocation can improve the reduction factor of dissemination time over Splash to 1.9 and 3.1 on Indriya and Flocklab respectively. Besides the improvement achieved by Fountain coding, the packet-level adaptation of channel allocation can further enhance the performance of Pando-F by 5.6% and 14.3% on Indriya and Flocklab.

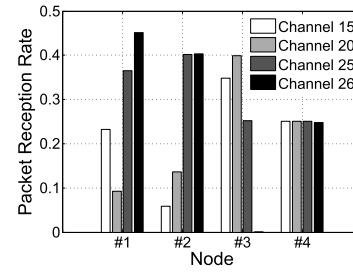


Fig. 13. Channel quality of four nodes on Indriya.

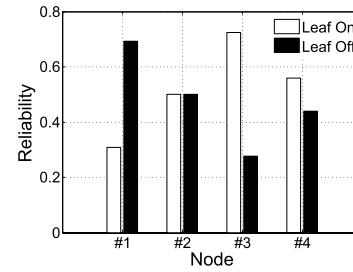


Fig. 14. Network density of four nodes on Indriya.

Figure 13 shows the contribution of each channel to the reliability of four randomly-selected nodes on Indriya testbed. Different channels have different performance at different locations. Channel 26 contributes ignorable increase of reliability at the node 3. It may be caused by the interference from the other nodes or devices. The experiment results suggest the necessity of packet-level channel diversity. By changing the channel allocation at packet-level, every node in the network experiences the four channels equally and circularly. The packet reception becomes more uniform across different nodes. Therefore, Pando maximizes the performance of Fountain codes in data dissemination, as it does not have to spend much time to deliver some packets to few poorly-performing nodes.

**Packet-Level Adaptation of Network Density:** As shown in Figure 12, the packet-level adaptation of network density of Pando-FD reduces the dissemination time of Pando-F by 4.8% and 10.4% respectively on Indriya and Flocklab. Figure 14 presents the contributions of two different network densities (determined by whether the leaf nodes transmit) for four different nodes of Indriya. The performance of concurrent transmissions is related to the node locations. Some nodes work well when the neighboring leaf nodes participate in the concurrent transmissions; the others may benefit more for a small number of concurrent transmitters. With packet-level adaptation of network density, Pando makes the reception performance of individual nodes more uniformly distributed across the network.

**Packet-Level Adaptation of Channel Diversity and Network Density:** According to the experiment results in Figure 12, Pando reduces the dissemination time of Pando-F by 9.5% and 26.0% on Indriya and Flocklab. The dissemination time of Pando is 5.7 seconds on Flocklab and the dissemination time of Pando-F is 7.7 seconds. Such significant improvement

demonstrates that the packet-level adaptation of channel diversity and network density is necessary for the Fountain-enabled data dissemination of constructive interference and pipelining.

## VI. RELATED WORK

In-field deployments of wireless sensor networks normally require remotely wireless reprogramming [26]. A variety of data dissemination protocols have been developed based on different techniques in the last decade. The best next-hop forwarder is selected by investigating its program version [21], [27], the information of its neighbors [24] or its link quality [8], [53]. The frequency of advertisement messages in Deluge is adjusted according to the network density [3]. The link correlation [1], [43], [45], [54] and opportunistic forwarding [19] are considered in flooding. The multi-channel scheme is used to improve the spatial reuse [28]. A backbone subnetwork is built to deliver data to the other nodes based on a minimum connected dominating set [36], [46], [52], [55]. Such a skeleton construction scheme has also been used in the information diffusion of mobile social networks [31]–[33]. The delay constraint is taken into account in [22]. Selective dissemination is studied in [37] and [40]. The above protocols disseminate data hop by hop using CSMA/CA or TDMA, which limits the spatial reuse and imposes heavy overhead of multiple access.

The multiple access overhead is first addressed by capture effect for flooding in wireless multi-hop networks [25], [30], whereas the performance of capture effect decreases when many nodes are transmitting simultaneously [30]. Many practical works, e.g., A-MAC [14], Glossy [16] and LWB [15], experimentally show that the reliability of constructive interference is high. Side-channel transmissions and rate adaptation have been studied in WLAN [44], [48]. However, they focus on the transmissions of individual packets. Splash [6] partially eliminates the multiple access overhead in data dissemination by combining constructive interference and pipelining.

Fountain codes [35] have been widely used to improve the broadcasting efficiency. The light-weight fountain transmission is first realized by LT codes [34]. SYNAPSE++ [39] leverages LT codes to enhance the broadcasting efficiency of Deluge [21] in each hop. DLT [9], [10] uses LT codes to improve the networking performance of wireless sensors. SoftLight [11], [12] develops a new rateless code for the soft hint enabled bit-level erasure channels of the screen-camera visible light communication. In [50], two feedback schemes are developed to estimate the number of redundant transmissions needed at the source. Network coding schemes, like random linear codes and XOR coding, are also adopted to improve the single-hop broadcasting efficiency in wireless sensor networks [20]. The above protocols based on Fountain codes or network coding, however, are designed for single-hop transmissions with explicit acknowledgement.

## VII. CONCLUSION

This paper presents *Pando*, a completely contention-free data dissemination protocol for wireless sensor networks.

By integrating Fountain codes with constructive interference and pipelining, Pando is able to continuously disseminate Fountain-encoded packets over the parallel pipelines. Three techniques, including radio-driven encoding and decoding, silence-based feedback scheme and packet-level adaptation of network density and channel diversity, are developed to transform Pando into a practical system, which asymptotically approaches the network capacity. Experiment results demonstrate that Pando can provide 100% reliability and significantly outperform the previous data dissemination protocols in dissemination time and energy consumption. In future works, it is interesting to analytically study the performance of Pando and the other data dissemination protocols. Moreover, accurate measurement of the worst-case packet reception rate of the nodes in the network is worthy to investigate to provide more efficient stop mechanism of the pipelined data dissemination.

## ACKNOWLEDGMENT

The authors thank the kind help and support offered by the Indriya testbed team led by Prof. M. C. Chan from NUS and the Flocklab testbed team led by Dr. J. Beutel and Prof. L. Thiele from ETH Zurich.

## REFERENCES

- [1] S. M. I. Alam, S. Sultana, Y. C. Hu, and S. Fahmy, "SYREN: Synergistic link correlation-aware and network coding-based dissemination in wireless sensor networks," in *Proc. IEEE MASCOTS*, Aug. 2013, pp. 485–494.
- [2] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proc. ACM MobiSys*, 2003, pp. 187–200.
- [3] S. Cho, H. Shin, S. Han, H. Cha, and R. Ha, "Density-adaptive network reprogramming protocol for wireless sensor networks," *Wireless Commun. Mobile Comput.*, vol. 10, no. 6, pp. 857–874, 2010.
- [4] M. Doddavenkatappa and M. Chan, "P<sup>3</sup>: A practical packet pipeline using synchronous transmissions for wireless sensor networks," in *Proc. ACM/IEEE IPSN*, Apr. 2014, pp. 203–214.
- [5] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3D wireless sensor network testbed," in *Proc. TRIDENTCOM*, 2011, pp. 302–316.
- [6] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast data dissemination with constructive interference in wireless sensor networks," in *Proc. USENIX NSDI*, 2013, pp. 269–282.
- [7] W. Dong, C. Chen, X. Liu, J. Bu, and Y. Gao, "A lightweight and density-aware reprogramming protocol for wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 10, no. 10, pp. 1403–1415, Oct. 2011.
- [8] W. Dong, Y. Liu, C. Wang, X. Liu, C. Chen, and J. Bu, "Link quality aware code dissemination in wireless sensor networks," in *Proc. IEEE ICNP*, Oct. 2011, pp. 89–98.
- [9] W. Du, Z. Li, J. C. Liando, and M. Li, "From rateless to distanceless: Enabling sparse sensor network deployment in large areas," in *Proc. ACM SenSys*, 2014, pp. 134–147.
- [10] W. Du, Z. Li, J. C. Liando, and M. Li, "From rateless to distanceless: Enabling sparse sensor network deployment in large areas," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2498–2511, Aug. 2016.
- [11] W. Du, J. C. Liando, and M. Li, "Soft hint enabled adaptive visible light communication over screen-camera links," *IEEE Trans. Mobile Comput.*, 2016, to be published, doi: 10.1109/TMC.2016.2551750.
- [12] W. Du, J. C. Liando, and M. Li, "Softlight: Adaptive visible light communication over screen-camera links," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1620–1628.
- [13] W. Du, J. C. Liando, H. Zhang, and M. Li, "When pipelines meet fountain: Fast data dissemination in wireless sensor networks," in *Proc. ACM SenSys*, 2015, pp. 365–378.
- [14] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis, "Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless," in *Proc. ACM SenSys*, 2010, pp. 1–14.

- [15] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proc. ACM SenSys*, 2012, pp. 1–14.
- [16] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proc. ACM/IEEE IPSN*, Apr. 2011, pp. 73–84.
- [17] Y. Gao, J. Bu, W. Dong, C. Chen, L. Rao, and X. Liu, "Exploiting concurrency for efficient dissemination in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 4, pp. 691–700, Apr. 2013.
- [18] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. ACM SenSys*, 2009, pp. 1–14.
- [19] S. Guo, Y. Gu, B. Jiang, and T. He, "Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links," in *Proc. ACM MobiCom*, 2009, pp. 133–144.
- [20] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless Deluge: Over-the-air programming of wireless sensor networks using random linear codes," in *Proc. ACM/IEEE IPSN*, Apr. 2008, pp. 457–466.
- [21] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. ACM Sensys*, 2004, pp. 81–94.
- [22] H. S. Kim, T. F. Abdelzaher, and W. H. Kwon, "Dynamic delay-constrained minimum-energy dissemination in wireless sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 4, no. 3, pp. 679–706, Aug. 2005.
- [23] J. Ko *et al.*, "Industry: Beyond interoperability: Pushing the performance of sensor network ip stacks," in *Proc. ACM SenSys*, 2011, pp. 1–11.
- [24] S. S. Kulkarni and L. Wang, "MNP: Multihop network reprogramming service for sensor networks," in *Proc. IEEE ICDCS*, Jun. 2005, pp. 7–16.
- [25] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proc. ACM SenSys*, 2013, p. 1.
- [26] P. Levis, D. Gay, and D. Culler, "Active sensor networks," in *Proc. USENIX NSDI*, vol. 2, 2005, pp. 343–356.
- [27] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. USENIX NSDI*, 2004, pp. 15–28.
- [28] C.-J. M. Liang, R. Musäloiu-E., and A. Terzis, "Typhoon: A reliable data dissemination protocol for wireless sensor networks," in *Proc. EWSN*, 2008, pp. 268–285.
- [29] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proc. ACM/IEEE IPSN*, 2013, pp. 153–166.
- [30] J. Lu and K. Whitehouse, "Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2491–2499.
- [31] Z. Lu, X. Sun, Y. Wen, and G. Cao, "Skeleton construction in mobile social networks: Algorithms and applications," in *Proc. IEEE SECON*, Jun. 2014, pp. 477–485.
- [32] Z. Lu, X. Sun, Y. Wen, G. Cao, and T. L. Porta, "Algorithms and applications for community detection in weighted networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 11, pp. 2916–2926, Nov. 2015.
- [33] Z. Lu, Y. Wen, and G. Cao, "Information diffusion in mobile social networks: The speed perspective," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1932–1940.
- [34] M. Luby, "LT codes," in *Proc. IEEE FOCS*, 2002, pp. 271–280.
- [35] D. J. C. MacKay, "Fountain codes," in *Proc. IEE*, May 2005, pp. 1062–1068.
- [36] V. Naik, A. Arora, P. Sinha, and H. Zhang, "Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices," in *Proc. IEEE RTSS*, Dec. 2005, pp. 277–286.
- [37] B. Pásztor, L. Mottola, C. Mascolo, G. P. Picco, S. Ellwood, and D. Macdonald, "Selective reprogramming of mobile sensor networks through social community detection," in *Proc. EWSN*, 2010, pp. 178–193.
- [38] 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. ACM Sensys*, 2010, pp. 15–28.
- [39] M. Rossi, N. Bui, G. Zanca, L. Stabellini, R. Crepaldi, and M. Zorzi, "SYNAPSE++: Code dissemination in wireless sensor networks using fountain codes," *IEEE Trans. Mobile Comput.*, vol. 9, no. 12, pp. 1749–1765, Dec. 2010.
- [40] M. Sathiamoorthy, K. R. Moghadam, B. Krishnamachari, and F. Bai, "Helper node allocation strategies for content dissemination in intermittently connected mobile networks," in *Proc. IEEE SECON*, Jun. 2014, pp. 55–63.
- [41] B. Sirkeci-Mergen, A. Scaglione, and G. Mergen, "Asymptotic analysis of multistage cooperative broadcast in wireless networks," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2531–2550, Jun. 2006.
- [42] K. Srinivasan *et al.*, "The  $\kappa$  factor: Inferring protocol performance using inter-link reception correlation," in *Proc. ACM MobiCom*, 2010, pp. 317–328.
- [43] T. Zhu, Z. Zhong, T. He, and Z.-L. Zhang, "Exploring link correlation for efficient flooding in wireless sensor networks," in *Proc. USENIX NSDI*, 2010, p. 4.
- [44] G. Wang, S. Zhang, K. Wu, Q. Zhang, and L. M. Ni, "Tim: Fine-grained rate adaptation in wans," *IEEE Trans. Mobile Comput.*, vol. 15, no. 3, pp. 748–761, Mar. 2016.
- [45] S. Wang, S. M. Kim, Y. Liu, G. Tan, and T. He, "CorLayer: A transparent link correlation layer for energy efficient broadcast," in *Proc. ACM MobiCom*, 2013, pp. 51–62.
- [46] S. Wang, G. Tan, Y. Liu, H. Jiang, and T. He, "Coding opportunity aware backbone metrics for broadcast in wireless networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 1999–2009, Aug. 2014.
- [47] Y. Wang, Y. He, X. Mao, Y. Liu, and X.-Y. Li, "Exploiting constructive interference for scalable flooding in wireless networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1880–1889, Dec. 2013.
- [48] K. Wu *et al.*, "HJam: Attachment transmission in WLANs," *IEEE Trans. Mobile Comput.*, vol. 12, no. 12, pp. 2334–2345, Dec. 2013.
- [49] Y. Wu, J. A. Stankovic, T. He, and S. Lin, "Realistic and efficient multi-channel communications in wireless sensor networks," in *Proc. INFOCOM*, Apr. 2008, pp. 1867–1875.
- [50] W. Xiao, S. Agarwal, D. Starobinski, and A. Trachtenberg, "Reliable rateless wireless broadcasting with near-zero feedback," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1924–1937, Dec. 2012.
- [51] W. Xiao and D. Starobinski, "Exploiting multi-Channel diversity to speed up over-the-air programming of wireless sensor networks," in *Proc. ACM SenSys*, 2005, pp. 292–293.
- [52] Z. Zhao, J. Bu, W. Dong, T. Gu, and X. Xu, "Coco++: Exploiting correlated core for energy efficient dissemination in wireless sensor networks," *Ad Hoc Netw.*, vol. 37, pp. 404–417, Feb. 2016.
- [53] Z. Zhao, W. Dong, J. Bu, T. Gu, and G. Min, "Accurate and generic sender selection for bulk data dissemination in low-power wireless networks," *IEEE/ACM Trans. Netw.*, 2016, to be published, doi: 10.1109/TNET.2016.2614129.
- [54] Z. Zhao, W. Dong, J. Bu, Y. Gu, and C. Chen, "Link-correlation-aware data dissemination in wireless sensor networks," *IEEE Trans. Ind. Electron.*, vol. 62, no. 9, pp. 5747–5757, Sep. 2015.
- [55] Z. Zhao, Z. Wang, G. Min, and Y. Cao, "Highly-efficient bulk data transfer for structured dissemination in wireless embedded network systems," *J. Syst. Archit.*, to be published, doi: <http://dx.doi.org/10.1016/j.sysarc.2016.09.001>.

**Wan Du** (S'10–A'11–M'15) received the B.E. and M.S. degrees in electrical engineering from Beihang University, China, in 2005 and 2008, respectively, and the Ph.D. degree in electronics, electrotechnics, and automation from the University of Lyon (École centrale de Lyon), France, in 2011. He is currently a Research Fellow with the School of Computer Science and Engineering, Nanyang Technological University, Singapore.

His research interests include cyber-physical system, distributed networking systems, and mobile systems. He is a member of the ACM.

**Jansen Christian Liando** (S'16) received the B.E. degree in computer science from Nanyang Technological University (NTU), Singapore, in 2014. He is currently a Research Assistant with the School of Computer Science and Engineering, NTU. His research interests include wireless sensor networks, mobile networking, and mobile computing.

**Huanle Zhang** (S'13–M'14) received the B.S. degree from Hangzhou Dianzi University in 2011, and the M.S. degree from the University of Electronic Science and Technology of China in 2014. He is currently a Project Officer with Nanyang Technology University, Singapore. His research interests include mobile systems and applications, small cells, and Internet of Things.

**Mo Li** (S'07–M'09–SM'16) received the B.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2004, and the Ph.D. degree in computer science and engineering from The Hong Kong University of Science and Technology in 2009. He is currently an Associate Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include distributed systems, wireless sensor networks, pervasive computing and RFID, and wireless and mobile systems. He is a member of the ACM.