When Pipelines Meet Fountain: Fast Data Dissemination in Wireless Sensor Networks

Wan Du, Jansen Christian Liando, Huanle Zhang, Mo Li
School of Computer Engineering, Nanyang Technological University, Singapore
{duwan, cjansen, Huanle.Zhang, limo}@ntu.edu.sg

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 the data dissemination process entirely over the fast and parallel pipelines. We implement Pando in Contiki and for TelosB sensor motes. We evaluate Pando's performance with various settings on two large-scale open testbeds, Indriya and Flocklab. Our experimental results show that Pando can provide 100% reliability and reduce the dissemination time of the state-of-the-art by 3.5×.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; C.2.2 [**Network Protocols**]: Protocol architecture

Keywords

Wireless sensor networks, Data dissemination, Fountain codes, Constructive interference.

1. INTRODUCTION

Data 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 [27, 39] 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 [17–19, 22, 28, 30, 36, 40] 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 [43] 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 [13] builds upon this principle and experimentally demonstrates that constructive interference can remove unnecessary channel contention and improve the flooding performance. Another work, PIP [38], exercises multi-channel communication and enables adjacent links in a multi-hop path to operate concurrently. Based on Glossy and PIP, Splash [7] enables data dissemination over the fast and parallel pipelines by combining constructive interference and pipelining, 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 [5,52] and the unreliability of constructive interference when the number of concurrent transmitters or the packet size is large [7,50], 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. Pando introduces randomness in the disseminated packets and fully exploits the wireless broadcast effect. It encodes data with Fountain codes [34] 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

SenSys'15, November 1–4, 2015, Seoul, South Korea..

DOI: http://dx.doi.org/10.1145/2809695.2809721.

re-disseminating the entire data object in rounds. The dissemination process is terminated once all nodes in the network accumulate sufficient encoded packets and successfully recover the original data. Pando accomplishes the dissemination process entirely over the contention-free pipelines. Thus, it can fully use the network resource and asymptotically approach the network capacity.

Transforming the idea behind Pando into a practical system is challenging for at least two reasons. First, the encoding and decoding of packets using Fountain codes need to be integrated into the timing-critical operations of constructive interference and pipelining. Incautious design of the Fountain decoder 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 of the data dissemination has to be set and conveyed to the source. The 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, which is essential to limit the time offset of concurrent transmissions within 0.5 μs (the synchronization requirement for generating constructive interference with high probability [13]). By parallelizing data coding and transmissions, we also avoid adding 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 feedback process is carefully integrated with the pipelined data dissemination to guarantee the reliability of data dissemination. The threshold of channel silence is adapted dynamically according to the ambient noise and interference. 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. We design a set of techniques to incorporate the adaptation operations into the feedback process and pipelined data dissemination, such as scheduled silence of leaf nodes, fast channel switching and failure recovery.

We implement Pando in Contiki and evaluate its performance on Indriya [6] and Flocklab [31], two large-scale testbeds for wireless sensor networks. In our experiments, 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 [7] and Deluge [19], over various network configurations. Our experimental results suggest that Pando reduces the average dissemination time of Splash by 3.5×, corresponding to a reduction factor of 32.7× over Deluge.

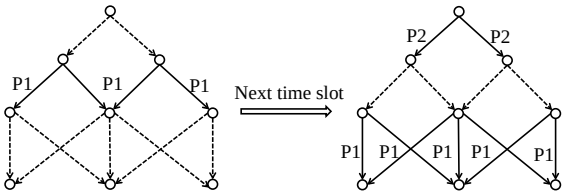


Figure 1: The pipelined tree built on constructive interference and pipelining. 'P1' and 'P2' represent the first packet and the second packet respectively.

2. MOTIVATION

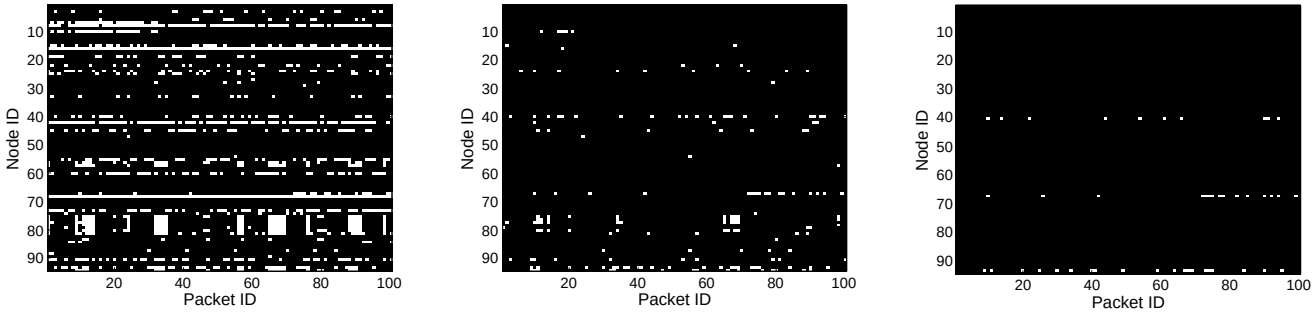
The goal of data dissemination is to deliver a bulk of data (e.g., an executable file or an application updating profile) to all the nodes in a network. Any packet loss may make the received data useless. Reliability and dissemination time are thus two key performance metrics for data dissemination. To achieve high reliability, previous protocols, like Deluge [19] and ReXOR [8], 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. For instance, two transmissions can occur concurrently only if they are apart from each other for at least three hops [19]. By leveraging constructive interference, Glossy [13] 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 [7].

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.

2.1 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 the upper layers (closer to the source) to the lower layers. In this paper, for the nodes at the same layer, we refer the nodes at their upper layer as their parent nodes and the nodes at the lower layer as their child nodes.

In the pipelined tree, the channel quality varies at different nodes [5, 52] and constructive interference becomes unreliable if the number of concurrent transmitters is large [50]. As a result, some packets may not be received by several nodes, and the data dissemination is not reliable. To the



(a) First round (85% reliability). **(b) Second round (96.8% reliability).** **(c) Third round (99% reliability).**

Figure 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.

best of our knowledge, Splash [7] 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 fast pipelines, a local recovery phase is performed. Nodes request the missing packets from their neighbors using CSMA/CA-based multiple access.

2.2 The long-tail problem

We conducted a series of experiments with Splash [7] on Indriya [6], the same testbed used in [7]. 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 2a 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 2a to Figure 2b, 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 2c 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 depicts the network reliability progress of Splash. It reveals that the long tail problem is severe in Splash. The total dissemination time of Splash for achieving 100% reliability is more than 10× larger than the time for achieving 75.9% reliability in the first round. Two obvious reasons can be found in Figure 3. Compared with the first round, the last two rounds provide limited contributions to the reliability progress. The contention-based local recovery (i.e., more than 13 seconds) is time-consuming. As reported in [7], the local recovery phase is much longer than the total transmission time of three dissemination rounds.

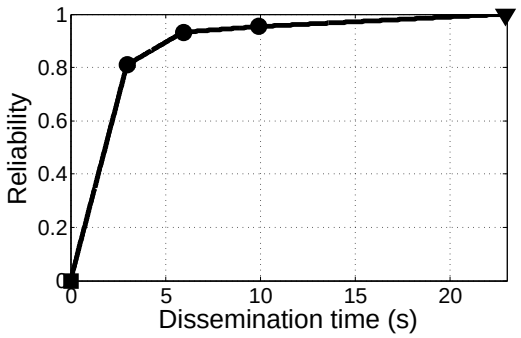


Figure 3: Network reliability of Splash at some specific time points. The end of 3 dissemination rounds is indicated by circle markers. The local recovery phase is from the end of the third round till the last node correctly receives the data object.

2.3 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. Instead of the identity of individual packets, the number of received packets becomes the key factor that determines the reliable reception of a data object. 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 2.2) 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*×.

Although Fountain codes have been implemented in some previous data dissemination protocols [17, 18, 40], 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. In a naive way of Fountain-enabled data dissemination, the source first disseminates a certain number of encoded packets and all nodes begin to decode after the dissemination. The decoding time is comparable with the dissemination time. To achieve 100% reliability, 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.

3. 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 4) is developed to trigger data dissemination, when general network activities (e.g., data collection or duty cycling) are being performed in sensor networks. During the data dissemination process, the general network activities are temporarily suspended, and the sensor mote resources (e.g., CPU and radio) are in active mode. 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.

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 3.2). A silence-based feedback scheme is developed for the one-way pipelines to promptly terminate the source and complete the dissemination process (Section 3.3). Packet-level adaptation of network density and channel diversity is adopted to boost the performance of constructive interference and pipelining (Section 3.4).

3.1 Preliminaries on Fountain codes

Fountain codes, like LT codes [33] and random linear codes [34], encode a data object of k packets, $\{X_1, X_2, \cdots, X_k\}$, into a rateless stream of encoded packets, $\{Y_1, Y_2, \cdots\}$. 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.

Although 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). Every encoded packet of LT codes is calculated by performing XOR operations of d packets that are randomly chosen from k original packets, where $d = 1, 2, \ldots, k$. 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. 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++ [40], which produces high coding efficiency for small data objects. Like in the previous work [10], we use a best seed of the random number generator. It produces the lowest decoding overhead and avoids the time consumption of random number generation in the decoding process. Although the random linear codes may provide higher coding efficiency, its computation overhead is high. It generates encoded packets using modular multiplication in Galois Field (GF) 2^8, which is time-consuming to execute compared with XOR operations.

3.2 Radio-driven coding scheme

We perform the encoding and decoding of LT codes along with the timing-critical communication 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 μs. 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 μs for general low-power microcontrollers) is much shorter than the latency of transmitting/receiving one byte by the radio hardware (i.e., 32 μs 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 [13] 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 RX mode to receive the next packet. They go into idle mode for a long duration before the end of the reception. Since the transmission time and the operations of constructive interference are predefined, the idle interval is constant for every packet transmission.

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. Based on the guard interval, the coding procedure can save its context before the hardware interrupt and resume its operation in the next idle interval. A timer is set at the beginning of the idle interval. It expires when the radio begins to receive the last 8th byte. The microcontroller saves the decoding context when the timer expires.

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 the transmission and reception performed by the radio hardware, the microcontroller tries to complete the decoding of the received packet in its idle intervals.

The Accumulative Gaussian Elimination (AGE) algorithm [10] 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, the decoding algorithm saves the context by recoding the decoding results and the intermediate variables, including the size of the submatrix, the coefficient matrix and the calculated results of data packets. The decoding computation occupies deterministic time which ends before the guard interval. In our implementation, the context saving costs at most 64 μs, which is much smaller than the guard interval (256 μs for IEEE 802.15.4 radios). The encoding can also be performed incrementally. The computation of encoding one packet is much less than the decoding of one packet, and thus nodes are able to generate one encoded packet every two time slots.

Timing process. Figure 4 presents the key timeline of Pando. When the microcontroller receives an interrupt for the end of a transmission (the SFD pin is set to low), it immediately sets the radio to the RX mode by sending two commands (i.e., channel setting and RXON) to the radio. After calibration (192 μs), the radio is ready to receive any incoming signals. When a 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

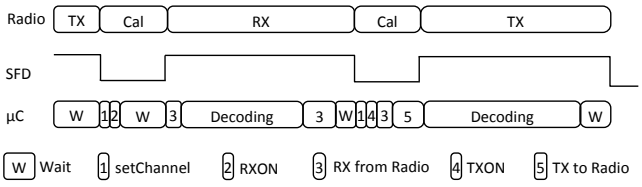


Figure 4: State transition of the radio and the microcontroller in Pando.

for a certain duration (e.g., 1688 μs 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. 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.

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 μs. It only takes 392 μs to read these bytes from the radio into the microcontroller via the SPI bus. Therefore, the microcontroller has 1688 μs for decoding during the reception of one packet. In the same principle, the microcontroller has a similar idle interval when the radio is transmitting.

Pando can be implemented on any hardware platforms that support constructive interference. Glossy [13] successfully demonstrated the feasibility of realizing constructive interference on IEEE 802.15.4-compliant hardware platforms. All the parameters of Pando, including RX/TX data rate, calibration interval and SPI transmission speed, are set according to the IEEE 802.15.4 standard. We do not need to tune these parameters for different hardware platforms.

3.3 Silence-based feedback scheme

As the source continuously pumps the rateless 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 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 ac-

knowledgement 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.

In our experiments using the above feedback scheme, some nodes stop working, 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 example of three-hop network, 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. Between the wake-up protocol and the data dissemination, we leave 10 time slots, during which all nodes measure the 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.

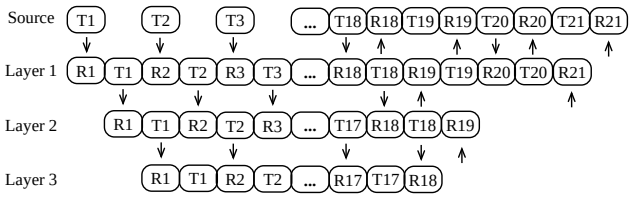


Figure 5: Dissemination process of a 16-packet data object in Pando. 'T1' refers to the first TX slot. 'R1' represents the first RX slot. The arrow indicates the signal transmission direction.

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 two 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. 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 5 that the overhead of the feedback process is small compared with the significant improvement of data dissemination efficiency.

3.4 Packet-level adaptation of channel diversity 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 [50] and the capture effect occurs at some locations of concurrent transmitters [13]. 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 [24].

Splash [7] 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 experiences three different channels at most in the three dissemination rounds respectively.

By applying Fountain codes in data dissemination, 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 diversity. 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 [5] are used circularly. At the beginning, all nodes in the network are set to a default channel to receive the first packet. After forwarding the received packet to their child nodes, they immediately switch to the channel allocated for the next packet. When the previous packets are being forwarded by the relay nodes, a new packet is released by the source. 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 sequence number of the last received packet. 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

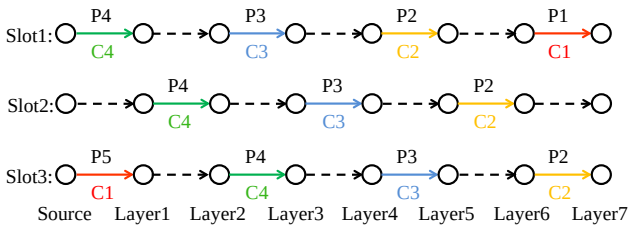


Figure 6: Packet-level channel allocation. 'P1' represents the first packet. 'C1' indicates the first channel. From slot 1 to slot 2, all the packets in the network are forwarded one hop further by the nodes of different layers with distinct channels.

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 μs 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 [7], 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, corresponding to the odd RX slots of their parent nodes, in which the parent nodes listen to the feedback from the leaf nodes. Therefore, the silence of leaf nodes in the time slots of odd packets does not impact the feedback process.

4. 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 the dissemination of a data object, 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 [13]. In our implementation, the start command is disseminated 20 times and it only takes tens of milliseconds in the fast and parallel pipelines. 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 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, because it is not overlapping with the channels of Wi-Fi (see http://www.wi-fi.org/) [45]. We disseminate the start command on channel 26 (the default channel in Pando). The start command includes information about the data object size. All nodes resume the previous network activities (e.g., data collection or duty cycling), when they complete the data dissemination and the feedback delivery.

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 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 in Pando. The bold arrows indicate how a 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 3.4, 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.

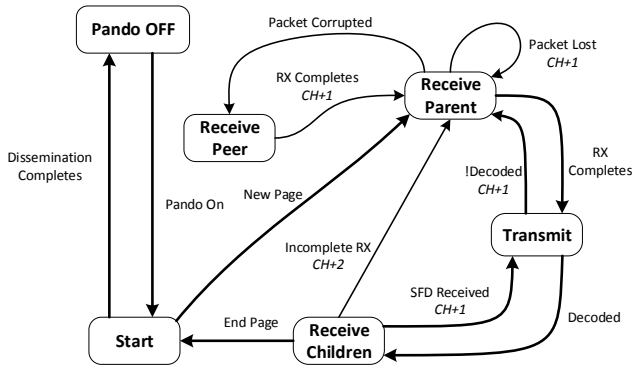


Figure 7: Node state machine in Pando.

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.

5. EVALUATION

We conduct a series of experiments on two large-scale open testbeds of wireless sensor networks, Indriya [6] and Flocklab [31]. 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).

We execute Pando while a conventional data collection protocols (e.g., CTP [15] for TinyOS and Contiki Collect [21] 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.

Protocols. In our experiments, we compare Pando with Deluge [19] and Splash [7], 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 [53], Rateless Deluge [17] and MT-Deluge [14]. 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 has no explicit termination time, we monitor

Table 1: Dissemination time (second) on Indriya[1].

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

Table 2: Dissemination time (second) on Flocklab.

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

the traces of all nodes and calculate the dissemination time as the last node correctly receives the data object.

5.1 Performance comparison

Table 1 and Table 2 show the dissemination time of three protocols on Flocklab and Indriya respectively. For each experiment, we run Deluge, Splash and Pando separately. For each protocol, five tests are conducted with different locations of the source node. Each run lasts 30 minutes. For Deluge and Splash, the data dissemination ends when the last node correctly recovers the original data. For Pando, it ends when the source receives the feedback and stops transmitting. 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 1 and Table 2, 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.

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. Pando fully utilizes the parallel pipelines during the entire dissemination process and the Fountain-coding computation in Pando does not add any overhead on the dissemination time.

Deluge has two versions: TinyOS's DelugeT2 and Contiki's Deluge. We use Contiki's Deluge, because it is difficult to execute TinyOS's DelugeT2 on remote testbeds which requires to run some tools on the machine connected 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 reduces the dissemination time of Deluge by 1337.8×. 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 [7]. The authors in [7] 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 current 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×.

[1]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 2 show that the dissemination time of Contiki's Deluge is orders of magnitude larger than Pando.

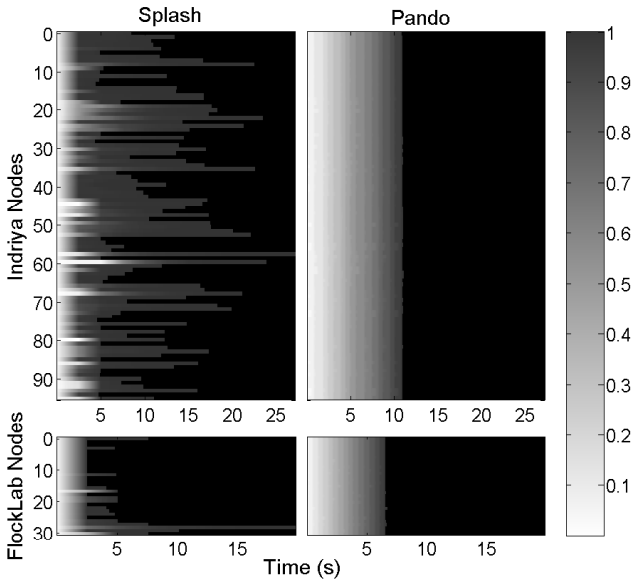


Figure 8: Reliability of individual nodes achieved by Pando and Splash on Indriya and Flocklab.

For the experiment results on Flocklab in Table 2, Pando produces higher improvement to the previous two protocols than the results on Indriya, since the sensor motes of Flocklab are more sparsely deployed and constructive interference performs better with small number of concurrent transmitters. Comparing Table 1 and Table 2, we also find that the gain of constructive interference and pipelining is huge. From Flocklab to Indriya, although the number of sensor nodes increases more than 2×, the dissemination time of both Pando and Splash only increases less than 1×. Based on constructive interference, the delivery of one packet from one layer of multiple nodes 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 of individual node. Figure 8 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 acknowledgement from all nodes in the last page dissemination. Splash's data dissemination ends at

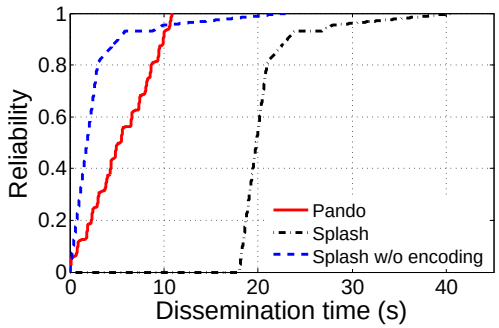


Figure 9: Network reliability progress on Indriya.

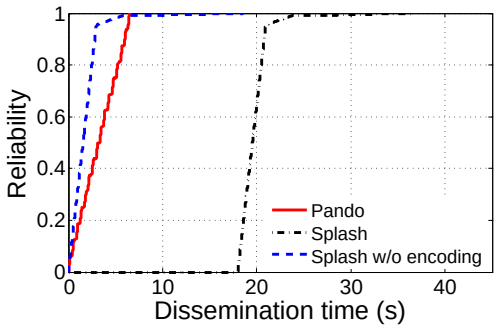


Figure 10: Network reliability progress on FlockLab.

18.7 seconds when the last node successfully receives the data object. The dissemination speed of Pando is 3.3× 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.

Reliability progress. Figure 9 and Figure 10 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× and 3.3× on Indriya and Flocklab respectively. 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 in the page-based data dissemination, Pando achieves much shorter dissemination time than Splash and the network reliability of Pando progresses smoothly for the dissemination of each data page. Pando is able to asymptotically approach the optimal dissemination time and adapt to different network deployments and dynamic link conditions.

Energy consumption. The nodes in wireless sensor networks are typically working in the duty-cycled mode, which reduces their energy consumption by setting them in low-power state periodically. The dissemination in the duty-cycled mode is extremely time-consuming due to the long-preamble transmissions [11]. Therefore, current data dissemination protocols [7, 14, 17–19, 40] keep the hardware resources on sensor motes (e.g., microcontroller and radio) in active state during the data dissemination process. During the dissemination, the power consumption of Pando is the same as the previous protocols, since all of them keep the nodes in active mode. Therefore, compared with the previous protocols, benefitting from a shorter dissemination time, Pando reduces the energy consumption of individual nodes by the same reduction factor as the dissemination time.

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 of the microcontroller or set the microcontroller to a power-saving mode during such idle intervals. Even if we enable Splash to sleep during the idle intervals of the microcontroller, the saving energy is very limited, because the energy consumption of sensor motes is dominated by the transceiver and the transceiver is transmitting or receiving packets during the whole data dissemination process. For example, the power consumption of the transceiver on TelosB is 69 mW (23mA*3V), which is much larger than the power consumption of the microcontroller, i.e., 5.4 mW (1.8mA*3V). Using the experiment traces in the above experiments, if we set the microcontroller to a power-saving mode (e.g., 0.002mA) during the idle intervals for Splash, only 5.2% energy could be saved. It is small, compared with the energy saving gain of Pando over Splash (69.5%).

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, 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.

Comparison with other protocols. We compare Pando with other data dissemination protocols by referring to their performance reported in their papers [8, 9, 14, 17, 22, 53]. All the previous protocols compare their performance with Deluge [19], the first reliable data dissemination protocol for wireless sensor networks. Based on the result reported in the papers of these protocols (i.e., reduction factor of dissemination time over Deluge), we can calculate the relative gain of Pando over other existing protocols.

Table 3 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 diversity, Pando asymptotically approaches the network capacity and reduces the dissemination time of Splash by more than 3.5×.

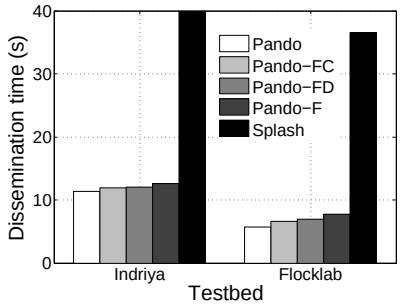


Figure 11: Contribution of individual techniques in Pando.

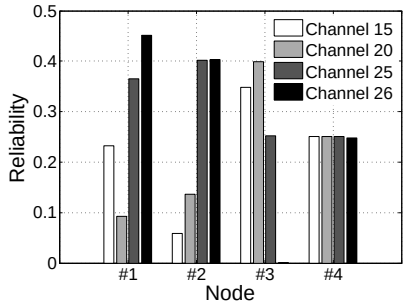


Figure 12: Channel quality experienced by four nodes of Indriya.

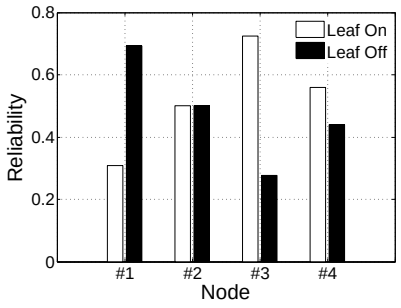


Figure 13: Network density experienced by four nodes of Indriya.

Table 3: Reduction factor of dissemination time over Deluge achieved by the existing data dissemination protocols in wireless sensor networks.

Protocols	Number of nodes	Data size (KB)	Reduction factor
MNP [22](2005)	100	5	1.21
MC-Deluge [53](2005)	25	24.3	1.60
Rateless Deluge [17](2008)	20	0.7	1.47
ReXOR [8](2011)	16	4	1.53
ECD [9](2011)	25	10	1.44
MT-Deluge [14](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

5.2 Contribution of individual techniques

To further analyze the gain of Pando, we execute Pando in several versions, in which the individual techniques of Pando are enabled separately. The results are shown in Figure 11. In Pando-F, the packet-level adaptation of both network density and channel diversity is disabled. The performance gain of Pando-F mainly comes from the Fountain coding. In Pando-FC, besides Fountain coding, we also enable the packet-level adaptation of channel diversity. Similarly, in Pando-FD, the packet-level adaptation of network density is enabled. Pando in Figure 11 represents the full version of Pando with all techniques.

Fountain codes. According to Figure 11, Pando-F can reduce the dissemination time of Splash by 3.2× and 4.7× on Indriya and Flocklab respectively. The fundamental gain of Pando comes from the integration of Fountain codes with the pipelined data dissemination. 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 diversity. By comparing the results of Pando-FC with Splash in Figure 11, we find that Pando with Fountain codes and packet-level adaptation of channel diversity can improve the reduction factor of dissemination time over Splash to 3.4 and 5.5 on Indriya and Flocklab respectively. Besides the improvement achieved by Fountain coding, the packet-level adaptation of channel diversity can further enhance the performance of Pando-F by 5.6% and 14.3% on Indriya and Flocklab.

Figure 12 shows the contribution of each channel to the reliability of four different nodes on Indriya. Different channels have different performance at different locations. Channel 26 contributes ignorable increase of reliability at the location of node 3. It may be caused by the interference from the other nodes or devices. Figure 12 fully demonstrates the diverse performance of the four available channels at different nodes, and suggests 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 11, 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 13 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 11, Pando reduces the dissemination time of Pando-F by 9.5% and 26.0% on Indriya and Flocklab respectively. 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.

Silence-based feedback. For the dissemination of one page, Pando needs an average of 276.4 ms and 135.0 ms to deliver the feedback information from the last successful node to the source on Indriya and Flocklab respectively. The number of consecutive silent slots M is set to 3 on Indriya, since the nodes of Indriya are more densely deployed and more packet losses occur. M is set to 2 on Flocklab for more efficient feedback collection. On both testbeds, there are no

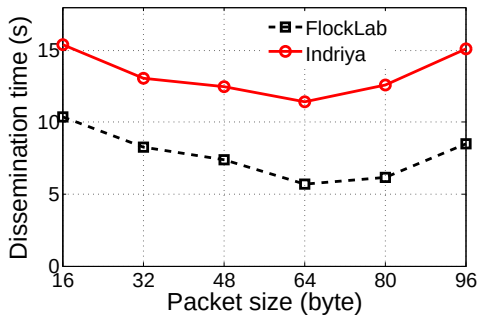


Figure 14: Performance for different payload sizes.

nodes that mistakenly stop their dissemination because of false channel silence. For the data object of 32 kBytes, the feedback delivery in Pando takes 4.4 seconds and 2.2 seconds in total on Indriya and Flocklab respectively. Although this amount of time is not directly used for disseminating the data packets, it is necessary to start the dissemination of a new data page and terminate the whole dissemination process. Even including the feedback time, Pando's dissemination time is still less than Splash by 28.7 seconds and 30.8 seconds on Indriya and Flocklab respectively.

In our experiments, we observe that the dynamic adaption of channel silence threshold $RSSI_{threshold}$ can well prevent the false negative of silence detection. The $RSSI_{threshold}$ could be from -91 dBm to -78 dBm for different nodes. For the node with a threshold of -78 dBm, it can detect a silence channel even the noise or interference is as large as -78 dBm. For the node with a threshold of -91 dBm, the noise and interference at its location are small, and it can successfully receive a packet with a signal of -89 dBm.

5.3 Effect of packet size

Figure 14 examines the performance of Pando with different packet sizes. The best performance of Pando comes with a moderate size of data packet, i.e., 64 bytes. When the packet size is small, the preamble of the physical and MAC layer introduces high percentage of overhead and occupies a large portion of the dissemination time. When the packet size is large, the performance of constructive interference decreases. From Figure 14, we see that the optimal packet size is 64 bytes for both testbeds. Although the performance of Pando depends on the best setting of packet size, the optimal setting does not change across testbeds with different densities and scales.

6. RELATED WORK

In-field deployments of wireless sensor networks normally require remotely wireless reprogramming [23, 26, 27, 35, 41, 46]. Content dissemination is also needed in other wireless networks, like vehicular networks [25] and social networks [29]. 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 [19, 28], the information of its neighbors [22] or its link quality [9]. The frequency of advertisement messages in Deluge is adjusted according to the network density [3]. The link correlation [1, 47, 48, 54] and opportunistic forwarding [16] are considered in flooding. The multi-channel scheme is used to improve the spatial reuse [30]. A backbone subnetwork is built to deliver data to the other nodes based on a minimum connected dominating set [36,49]. The delay constraint is taken into account in [20]. Selective dissemination is studied in [37, 42]. A data-centric data dissemination scheme is proposed in [4]. 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 addressed by capture effect for flooding in wireless multi-hop networks [24,32,44,51]. However, the performance of capture effect decreases when many nodes are transmitting simultaneously [32]. Many practical works, e.g., A-MAC [11], Glossy [13] and LWB [12], experimentally show that the reliability of constructive interference is high. However, the payload size of the transmitted packets in these works is small (i.e., 8 or 15 bytes). For the dissemination of a large data object, long payload size (e.g., 64 bytes) should be used. According to the experiments in [7], also confirmed by our experiments, constructive interference becomes unreliable for large packet size. Meanwhile, PIP [38] constructs a fast data delivery pipeline using multiple channels. Splash [7] partially eliminates the multiple access overhead in data dissemination by combining constructive interference and pipelining.

Fountain codes [34] have been widely used to improve the broadcasting efficiency. The light-weight fountain transmission is first realized by LT codes [33]. SYNAPSE++ [40] leverages LT codes to enhance the broadcasting efficiency of Deluge [19] in each hop. DLT [10] use LT codes to improve the networking performance of wireless sensors. Network coding schemes, like random linear codes and XOR coding, are also adopted to improve the single-hop broadcasting efficiency in wireless sensor networks [8, 17, 18]. The above protocols based on Fountain codes or network coding, however, are designed for single-hop transmissions with explicit acknowledgement.

7. 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 rateless 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.

8. ACKNOWLEDGMENTS

We thank our shepherd Prof. Silvia Santini and the anonymous reviewers for their valuable suggestions and feedback. We also thank the kind help and support offered by the Indriya testbed team led by Prof. Mun Choon Chan from NUS and the Flocklab testbed team led by Dr. Jan Beutel and Prof. Lothar Thiele from ETH Zurich. This work is supported by Singapore MOE AcRF Tier 2 MOE2012-T2-1-070 and NTU Nanyang Assistant Professorship (NAP) grant M4080738.020.

9. REFERENCES

[1] S. Alam, S. Sultana, Y. C. Hu, and S. Fahmy. SYREN: Synergistic link correlation-aware and network coding-based dissemination in wireless sensor networks. In *IEEE MASCOTS*, pages 485–494, 2013.

[2] A. Boulis, C.-C. Han, and M. B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. In *ACM MobiSys*, pages 187–200, 2003.

[3] S. Cho, H. Shin, S. Han, H. Cha, and R. Ha. Density-adaptive network reprogramming protocol for wireless sensor networks. *Wireless Communications and Mobile Computing*, 10(6):857–874, 2010.

[4] M. Ditzel and K. Langendoen. D3: Data-centric data dissemination in wireless sensor networks. In *The European Conference on Wireless Technology*, pages 185–188, 2005.

[5] M. Doddavenkatappa and M. C. Chan. P3: a practical packet pipeline using synchronous transmissions for wireless sensor networks. In *ACM/IEEE IPSN*, pages 203–214, 2014.

[6] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In *TRIDENTCOM*, 2011.

[7] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *USENIX NSDI*, pages 269–282, 2013.

[8] W. Dong, C. Chen, X. Liu, J. Bu, and Y. Gao. A lightweight and density-aware reprogramming protocol for wireless sensor networks. *IEEE Transactions on Mobile Computing*, pages 1403–1415, 2011.

[9] W. Dong, Y. Liu, C. Wang, X. Liu, C. Chen, and J. Bu. Link quality aware code dissemination in wireless sensor networks. In *IEEE ICNP*, pages 89–98, 2011.

[10] W. Du, Z. Li, J. C. Liando, and M. Li. From rateless to distanceless: Enabling sparse sensor network deployment in large areas. In *ACM SenSys*, pages 134–147, 2014.

[11] 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 *ACM SenSys*, pages 1–14, 2010.

[12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *ACM SenSys*, pages 1–14, 2012.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *ACM/IEEE IPSN*, pages 73–84, 2011.

[14] Y. Gao, J. Bu, W. Dong, C. Chen, L. Rao, and X. Liu. Exploiting concurrency for efficient dissemination in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 691–700, 2013.

[15] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *ACM SenSys*, pages 1–14, 2009.

[16] S. Guo, Y. Gu, B. Jiang, and T. He. Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links. In *ACM MobiCom*, pages 133–144, 2009.

[17] A. Hagedorn, D. Starobinski, and A. Trachtenberg. Rateless Deluge: Over-the-air programming of wireless sensor networks using random linear codes. In *ACM/IEEE IPSN*, 2008.

[18] I.-H. Hou, Y.-E. Tsai, T. F. Abdelzaher, and I. Gupta. AdapCode: Adaptive network coding for code updates in wireless sensor networks. In *IEEE INFOCOM*, 2008.

[19] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *ACM Sensys*, pages 81–94, 2004.

[20] H. S. Kim, T. F. Abdelzaher, and W. H. Kwon. Dynamic delay-constrained minimum-energy dissemination in wireless sensor networks. *ACM Transactions on Embedded Computing Systems*, 4(3):679–706, 2005.

[21] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler. Industry: beyond interoperability: pushing the performance of sensor network ip stacks. In *ACM SenSys*, pages 1–11, 2011.

[22] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. In *IEEE ICDCS*, pages 7–16, 2005.

[23] T.-t. T. Lai, Y.-h. T. Chen, H.-h. Chu, and P. Huang. Pipeprobe: mapping hidden water pipelines. In *ACM SenSys*, pages 375–376, 2009.

[24] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *ACM SenSys*, pages 1:1–1:14, 2013.

[25] I. Leontiadis, P. Costa, and C. Mascolo. Persistent content-based information dissemination in hybrid vehicular networks. In *IEEE PerCom*, pages 1–10, 2009.

[26] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft. Senshare: Transforming sensor networks into multi-application sensing infrastructures. In *EWSN*, pages 65–81, 2012.

[27] P. Levis, D. Gay, and D. Culler. Active sensor networks. In *USENIX NSDI*, pages 343–356, 2005.

[28] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks. In *USENIX NSDI*, pages 15–28, 2004.

[29] S. Li, L. Su, Y. Suleimenov, H. Liu, T. Abdelzaher, and G. Chen. Centaur: Dynamic message dissemination over online social networks. In *ICCCN*, pages 1–8, 2014.

[30] C.-J. M. Liang, R. Musăloiu-e, and A. Terzis. Typhoon: A reliable data dissemination protocol for wireless sensor networks. In *EWSN*, pages 268–285. Springer, 2008.

[31] 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 *ACM/IEEE IPSN*, pages 153–166, 2013.

[32] J. Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *IEEE INFOCOM*, pages 2491–2499, 2009.

[33] M. Luby. LT codes. In *IEEE FOCS*, pages 271–280, 2002.

[34] D. J. MacKay. Fountain codes. *IEE*

Proceedings-Communications, pages 1062–1068, 2005.

[35] L. Mottola, G. P. Picco, M. Ceriotti, S. Gună, and A. L. Murphy. Not all wireless sensor networks are created equal: A comparative study on tunnels. *ACM Transactions on Sensor Networks*, 7(2):1–33, 2010.

[36] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices. In *IEEE RTSS*, pages 777–789, 2005.

[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 *EWSN*, pages 178–193, 2010.

[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 *ACM Sensys*, pages 15–28, 2010.

[39] N. Reijers and K. Langendoen. Efficient code distribution in wireless sensor networks. In *ACM WSNA*, pages 60–67, 2003.

[40] M. Rossi, N. Bui, G. Zanca, L. Stabellini, R. Crepaldi, and M. Zorzi. SYNAPSE++: code dissemination in wireless sensor networks using fountain codes. *IEEE Transactions on Mobile Computing*, pages 1749–1765, 2010.

[41] S. Santini, B. Ostermaier, and A. Vitaletti. First experiences using wireless sensor networks for noise pollution monitoring. In *Proceedings of ACM REALWSN*, pages 61–65, 2008.

[42] M. Sathiamoorthy, K. R. Moghadam, B. Krishnamachari, and F. Bai. Helper node allocation strategies for content dissemination in intermittently connected mobile networks. In *IEEE SECON*, pages 55–63, 2014.

[43] B. Sirkeci-Mergen, A. Scaglione, and G. Mergen. Asymptotic analysis of multistage cooperative broadcast in wireless networks. *IEEE Transactions on Information Theory*, pages 2531–2550, 2006.

[44] D. Son, B. Krishnamachari, and J. Heidemann. Experimental study of concurrent transmission in wireless sensor networks. In *ACM SenSys*, pages 237–250, 2006.

[45] K. Srinivasan, M. Jain, J. I. Choi, T. Azim, E. S. Kim, P. Levis, and B. Krishnamachari. The κ factor: Inferring protocol performance using inter-link reception correlation. In *ACM MobiCom*, pages 317–328, 2010.

[46] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail. PIPENET: A wireless sensor network for pipeline monitoring. In *ACM/IEEE IPSN*, pages 264–273, 2007.

[47] T. Zhu, Z. Zhong, T. He and Z.-L. Zhang. Exploring link correlation for efficient flooding in wireless sensor networks. In *USENIX NSDI*, 2010.

[48] S. Wang, S. M. Kim, Y. Liu, G. Tan, and T. He. CorLayer: a transparent link correlation layer for energy efficient broadcast. In *ACM MobiCom*, pages 51–62, 2013.

[49] S. Wang, G. Tan, Y. Liu, H. Jiang, and T. He. Coding opportunity aware backbone metrics for broadcast in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(8):1999–2009, 2014.

[50] Y. Wang, Y. He, X. Mao, Y. Liu, and X.-Y. Li. Exploiting constructive interference for scalable flooding in wireless networks. *IEEE/ACM Transactions on Networking*, pages 1880–1889, 2013.

[51] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *IEEE EmNetS-II*, pages 45–52, 2005.

[52] Y. Wu, J. Stankovic, T. He, S. Lin, et al. Realistic and efficient multi-channel communications in wireless sensor networks. In *INFOCOM*, 2008.

[53] W. Xiao and D. Starobinski. Poster abstract: Exploiting multi-channel diversity to speed up over-the-air programming of wireless sensor networks. In *ACM SenSys*, pages 292–293. ACM, 2005.

[54] Z. Zhao, W. Dong, J. Bu, Y. Gu, and C. Chen. Link correlation aware data dissemination in wireless sensor networks. *IEEE Transactions on Industrial Electronics*, 2015.