Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks

Manjunath Doddavenkatappa, Mun Choon Chan, Ben Leong
National University of Singapore

Abstract

It is well-known that the time taken for disseminating a large data object over a wireless sensor network is dominated by the overhead of resolving the contention for the underlying wireless channel. In this paper, we propose a new dissemination protocol called *Splash*, that eliminates the need for contention resolution by exploiting constructive interference and channel diversity to effectively create fast and parallel pipelines over multiple paths that cover all the nodes in a network. We call this *tree pipelining*. In order to ensure high reliability, Splash also incorporates several techniques, including exploiting transmission density diversity, opportunistic overhearing, channel-cycling and XOR coding. Our evaluation results on two large-scale testbeds show that Splash is more than an order of magnitude faster than state-of-the-art dissemination protocols and achieves a reduction in data dissemination time by a factor of more than 20 compared to DelugeT2.

1 Introduction

A data dissemination protocol, like Deluge [14], is a fundamental service required for the deployment and maintenance of practical wireless sensor networks because of the need to periodically re-program sensor nodes in the field. Existing data dissemination protocols employ either a contention based MAC protocol like CSMA/CA [6, 5, 7, 10, 12, 30, 18, 14] or TDMA [17] for resolving the multiple access problem of the wireless channel. As there is a large amount of data that needs to be disseminated to all the nodes in the network, there is often severe contention among the many transmissions from many nodes. Existing MAC protocols incur significant overhead in contention resolution, and it has been shown that Deluge can take as long as an hour to program a 100-node sensor network [27].

In this paper, we propose a new data dissemination protocol, called *Splash*, that completely eliminates contention overhead by exploiting constructive interference. Splash is scalable to large, multi-hop sensor networks and it is built upon two recent works: Glossy [9] and PIP [24]. Glossy uses constructive interference in practical sensor networks to enable multiple senders to transmit the same packet simultaneously, while still allowing multiple receivers to correctly decode the transmitted packet. Like Glossy, we eliminate the overhead incurred in contention resolution by exploiting constructive interference. Raman et al. showed in PIP that a pipelined transmission scheme exploiting channel diversity can avoid self interference and maximize channel utilization for a single flow over multiple hops by ensuring that each intermediate node is either transmitting or receiving at any point of time. Splash uses constructive interference to extend this approach to *tree pipelining*, where each level of a dissemination tree serves as a stage of the pipeline.

While the naive combination of synchronized and pipelined transmissions achieves substantial gains in the data dissemination rate by maximizing the transmission opportunities of the senders, it also creates a significant reliability issue at the receivers. First, in order to improve efficiency, we need to use a large packet size (i.e. at least 64 bytes). However, increasing packet size reduces the reliability of constructive interference as the number of symbols to be decoded correctly increases [9]. Second, channel quality varies significantly among different channels, and there are typically only a small number of available channels that are of sufficiently good quality. If a poor channel is chosen for a stage of the pipeline, the pipeline transmission may be stalled.

Splash includes a number of techniques to improve the packet reception rate. (1) We improve the reception rates over all receivers by exploiting transmitter density diversity by varying the number of transmitters between transmission rounds. When the sets of transmitters are varied, the sets of receivers that can decode the synchronized transmissions correctly also change. Hence, different sets of nodes are likely to correctly decode packets

during different transmission rounds. The challenge is to maximize the differences among different transmission rounds. (2) We increase reception opportunities by incorporating opportunistic overhearing which involves early error detection and channel switching. A node in Splash identifies a corrupted packet on-the-fly during its reception and switches its channel to overhear the same packet when it is being forwarded by its peer nodes in the dissemination tree. (3) We exploit channel diversity to improve packet reception ratio by varying the channels used between different transmission rounds. This is particularly important since the use of the same bad channel can stall the pipeline transmission consistently. (4) Finally, we utilize a simple XOR coding scheme to improve packet recovery by exploiting the fact that most receivers would have already received most of the packets after two transmission rounds.

We implemented Splash in Contiki-2.5 and we evaluated the protocol on the Indriya testbed [3] with 139 nodes and the Twist testbed [13] with 90 nodes. We compare Splash to both Deluge [14] in Contiki and to the much improved DelugeT2 implemented in TinyOS. As we use DelugeT2 as a baseline, it allows us to compare Splash to many of the existing dissemination protocols in the literature as most of them are also compared to Deluge. Our results show that Splash is able to disseminate a 32-kilobyte data object in about 25 seconds on both the testbeds. Compared to DelugeT2, Splash reduces dissemination time on average by a factor of 21, and in the best case, by up to a factor of 57.8. This is significantly better than MT-Deluge [10], the best state-of-the-art dissemination protocol, which achieves a reduction factor of only 2.42 compared to Deluge.

The dissemination performance of our current implementation of Splash achieves a *network-wide* goodput of 10.1 kilobits/sec per node for a multihop network of 139 nodes with up to 9 hops. Splash's goodput is higher than that of all the network-wide data dissemination protocols [6, 5, 7, 10, 12, 30, 18, 14, 17] previously proposed in the literature. Splash's performance is comparable to Burst Forwarding [8], the state-of-the-art pipelined bulk transfer protocol over TCP for sensor networks, which is able to achieve a goodput of up to 16 kilobits/sec, but only for a single flow over a single multihop path.

Finally, Splash is also significantly more compact than DelugeT2 in terms of memory usage. Splash uses 9.63 and 0.68 kilobytes less ROM and RAM respectively than DelugeT2. Given that it is not uncommon for sensor devices to have only about 48 and 10 kilobytes of ROM and RAM respectively, these are significant savings in memory, that will be available for use by sensor applications.

The rest of the paper is organized as follows. In Section 2, we discuss the related work. Section 3 presents our measurement study of constructive interference on a practical testbed. We present Splash and the details of its implementation in Section 4. Section 5 presents our evaluation results on the Indriya and Twist testbeds. Finally, we conclude in Section 6.

2 Related Work

In their seminal work on Glossy [9], Ferrari et al. showed that constructive interference is practical in wireless sensor networks. They observed that there is a high probability that the concurrent transmissions of a same packet will result in constructive interference if the temporal displacement among these transmissions is smaller than 0.5 microsecond. The implementation of Glossy is able to meet this requirement and a small packet can be flooded to all nodes with deterministic delays at the relay nodes which allows accurate network-wide synchronization. Glossy is designed to flood a single packet at a time, e.g., a control packet. On the other hand, a dissemination protocol needs to achieve bulk transfer of large packets, which introduces a new set of problems such as the need for 100% reliability, pipelining, channel switching, and scalability in terms of both network size and constructive interference.

The scalability of constructive interference was recently studied by Wang et al. [28]. They showed that the reliability of constructive interference decreases significantly when the number of concurrent transmitters increases, where *reliability* is defined as the probability that a packet that is concurrently transmitted by multiple transmitters will be decoded correctly at a receiver. While [28] is the first work to study this problem, it is based on theory and simulations, and does not include any experimental evaluation. Our empirical results show that the scalability problem highlighted is actually more severe in practice. Wang et al. also proposed Spine Constructive Interference based Flooding (SCIF) to mitigate the scalability problem, but the correctness of SCIF assumes many conditions that are hard to achieve in practice. For example, length of a network cell is half of the radio communication range. In contrast, our strategy for handling the scalability problem is a fully practical solution based on collection tree protocols such as CTP [11] and the observation that typically more than 50% of nodes in a collection tree are leaf nodes even at the lowest transmission power where the underlying network is connected [4].

A key challenge in implementing pipelining over a multihop path is self interference: a node's next packet can interfere with its immediate previously forwarded packet. There are two common solutions. First, we can introduce inter-packet gaps such that the previous packet would be out of the interference range before attempting to transmit the next packet [15]. However, this method

would drastically reduce the end-to-end throughput as a long gap of 5 packet transmission times is required for a single flow in practice [15]. Moreover, in the case where multiple data flows are active, this method is ineffective because of inter-flow interference. The second solution is to exploit channel diversity [23, 24, 8]. However, we observe that this approach ignores two practical issues that can severely degrade the performance of its packet pipeline. First, although the IEEE 802.15.4 standard defines 16 non-overlapping channels, the number of channels of usable quality is typically much smaller in practice because of various factors, e.g., interference from WiFi channels [21]. Second, the approach ignores the fact that links for routing are typically chosen on the best available channel, and the performance of other channels on such links can be poor in practice. These two issues can severely degrade the performance by stalling the packet pipeline.

As dissemination is a fundamental service in sensor networks, there are numerous protocols in the literature [6, 7, 10, 12, 18, 14, 17]. Typically, they are epidemic approaches incorporating special techniques in order to reduce the incurred overhead. Such techniques include Trickle suppression [20], network coding [12], exploiting link qualities [6], virtual machines [19], etc. While existing protocols differ in their techniques, they all share a common feature that they employ a MAC protocol like CSMA/CA or TDMA for contention resolution, and typically their dissemination times are in the order of minutes for disseminating full images in practical networks. Our goal in this paper is to completely eliminate contention overhead by exploiting constructive interference and we show that by doing so, we can reduce the dissemination time by an order of magnitude compared to existing approaches.

3 Measurement Study

To understand the behavior of simultaneous transmissions in real-world setups, we conducted a measurement study of constructive interference on the Indriya [3] wireless sensor testbed. In particular, we studied the scalability of simultaneous transmissions and correlation among packet receptions across different nodes decoding such transmissions.

We used the code from the Glossy [9] project in our experiments, our experimental methodology is similar to that adopted by Ferrari et al. in [9]. An initiator node broadcasts a packet to a set of nodes which in turn forward the received packet concurrently back to the initiator. This results in constructive interference at the initiator, where we measured the reliability of the reception. Since our goal is to use constructive interference for the dissemination of large objects, we used the maximum packet size of 128 bytes in our experiments. In addition, the payload of each packet was randomized. Our experiments were carried out on the default Channel 26, unless specified otherwise. Channel 26 is one of the only two ZigBee channels that does not overlap with the commonly used WiFi channels [21].

3.1 Scalability

In Fig. 1, we plot the reliability of packet reception against the number of concurrent transmitters for three randomly chosen initiators on three different floors of the Indriya testbed. In each experiment, both the initiator and the randomly chosen set of concurrent transmitters were located on the same floor. We recorded over 1,000 packet transmissions on each floor on Channel 26. We see from Figs. 1(a) and 1(b) that reliability generally decreases when there are more concurrent transmitters.

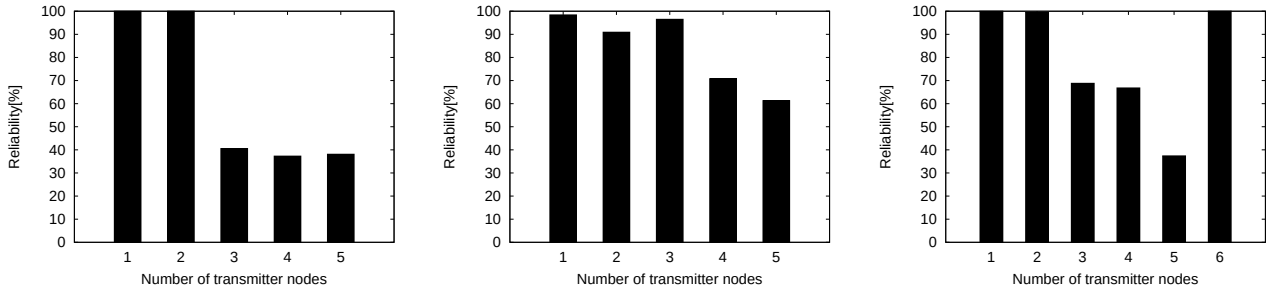
In fact, it had been shown by Wang et al. [28] through analytical model and simulation that the reliability of constructive interference decreases when the number of concurrent transmitters increases, due to the increase in the probability of the maximum time displacement across different transmitters exceeding the required threshold for constructive interference. Our measurements suggest that the highlighted problem is more severe in practice, and even a small number of three to five concurrent transmitters can significantly degrade the reception at a receiver.

However, it is sometimes possible for an increase in the number of concurrent transmitters to result in improved reception reliability. In particular, we see in Fig. 1(c) that by adding a sixth node, the reliability increases from about 37% to 100%. This is likely caused by the *capture effect* since the sixth node was located some 2 meters away from and within line of sight of the initiator.

This suggests that the impact of the number of transmitters (transmission density) on reception reliability does not follow a fixed trend like what was predicted by Wang et al. [28]. But depends also on the positions of the concurrent transmitters relative to the receiver. So, instead of attempting to determine the optimal transmission density, we can try to transmit at both high and low transmission densities to improve reception reliability.

3.2 Receiver Correlation

In existing dissemination protocols, it is common for a node to attempt to recover missing packets from its neighbors. It is hence important for us to understand the correlation of the packets received by neighboring receivers. While Srinivasan et al. had previously investigated the correlation of packets received by the receivers



(a) Floor 1 (b) Floor 2 (c) Floor 3

Figure 1: Plot of reliability against the number of concurrent senders.

in a sensor network [26], they did not study the correlation in the presence of constructive interference.

To this end, we set up an experiment involving 21 nodes spanning an area of 30m $\times$ 30m on the 3rd floor of Indriya. One node was designated as the initiator node, ten nodes were randomly chosen to serve as relays, and the remaining ten were used as receivers. The initiator broadcasts a packet once every second over a duration of four hours and the relay nodes forward the packet concurrently, which results in constructive interference at the various receiver nodes. As Srinivasan et al. had earlier shown that WiFi interference is the most likely reason for correlations in packet reception [26], we repeated this experiment on two separate channels: Channel 26, which is non-overlapping with the WiFi channels occupied in the building where Indriya is deployed, and Channel 22, which overlaps with an occupied WiFi channel.

We investigated the correlation among the packet receptions at the receiver nodes (R) by computing the Pearson's correlation coefficient at a granularity of one packet. We present the coefficient values for Channels 26 and 22 in Table 1. Note that as a coefficient matrix corresponding to a channel is symmetric, we represent data corresponding to the two channels in a single table (matrix). The values in the lower half of the table (below the diagonal) correspond to Channel 26 and the upper half corresponds to Channel 22.

As expected, for Channel 26, which does not overlap with an occupied WiFi channel, the correlation coefficients are small. This suggests that the packet receptions across different receivers are effectively independent. On the other hand, for Channel 22, which overlaps with an occupied WiFi channel, the coefficients are relatively large, indicating that there is significant correlation in the reception at the various receivers. Our results suggest that it might be hard for a node to recover missing packets from its neighbors if a noisy channel like Channel 22 is used, since many neighboring nodes would likely be missing the same packets.

Table 1: Correlation coefficients observed on Channel 26 (lower half) and Channel 22 (upper half).

R	1	2	3	4	5	6	7	8	9	10
1	**1.0**	.56	.62	.64	.57	.58	.60	.52	.55	.58
2	.04	**1.0**	.52	.63	.51	.54	.46	.53	.50	.55
3	0.0	-.02	**1.0**	.55	.48	.56	.46	.44	.46	.49
4	.05	.23	0.0	**1.0**	.61	.61	.52	.63	.59	.68
5	.04	.07	.01	.13	**1.0**	.51	.52	.51	.61	.53
6	.03	.09	-.01	.13	.03	**1.0**	.46	.48	.50	.53
7	.03	.12	0.0	.16	.06	.09	**1.0**	.45	.49	.47
8	.02	.11	-.01	.17	.06	.11	.13	**1.0**	.49	.66
9	.02	.03	.01	.06	.08	.02	.05	.02	**1.0**	.49
10	.02	.10	0.0	.15	.10	.09	.17	.21	.05	**1.0**

4 Splash

In this section, we describe Splash, a new data dissemination protocol for large data objects in large sensor networks that completely eliminates contention overhead by exploiting constructive interference and pipelining.

Raman et al. proposed PIP (Packets in Pipeline) [24] for transferring bulk data in a pipelined fashion over a single path of nodes over multiple channels. They exploit channel diversity to avoid self interference by having each intermediate node use a different channel to receive packets. A key insight of this pipeline approach is that at any point in time, an intermediate node is either transmitting or receiving packets and this achieves the maximal utilization of air time.

Splash can be considered as an extension of PIP's approach that incorporates three key innovations to support data dissemination to multiple receivers over multiple paths:

1. *Tree pipelining* which exploits constructive interference to effectively create parallel pipelines over multiple paths that cover all the nodes in a network. In our approach, a collection tree is used in the reverse direction for dissemination which in turn allows us to mitigate the scalability problem of the

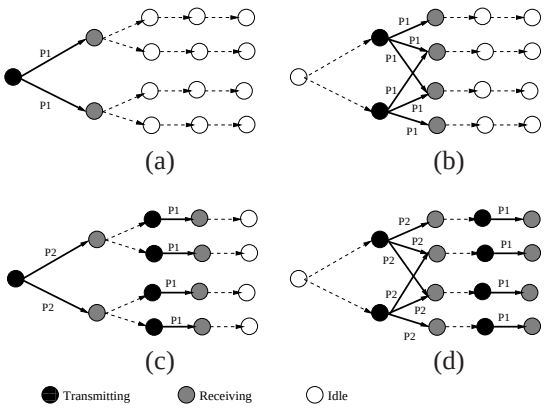


Figure 2: Illustration of pipelining over a tree.

constructive interference and to minimize the differences that exist among the performance of different channels.

2. *Opportunistic overhearing* from peers by exploiting multiple pipelines, which provides each node with more chances of receiving a packet.

3. *Channel cycling* that increases the chance of reusing a good channel while avoiding interference. Different channels are used at different stages of the pipeline between different transmission rounds to avoid stalling of the pipeline in case a bad channel is inadvertently chosen.

In the rest of this section, we discuss in detail various components of Splash and some of its implementation details.

4.1 Tree Pipelining

Splash is the first protocol to exploit constructive interference to support pipelining over a dissemination tree in which each level of the tree acts as one stage of the pipeline. This is illustrated in Fig. 2.

In the first cycle (see Fig. 2(a)), the root node (level zero) transmits the first packet P1. The receivers at the first level, which are synchronized upon receiving P1, will simultaneously forward P1 in the second cycle so that these simultaneous transmissions interfere constructively at the nodes on the second level (see Fig. 2(b)). In the third cycle (see Fig. 2(c)), while nodes at the second level forward P1 to the third level, the root node simultaneously transmits the second packet P2. Note that these simultaneous transmissions of different packets do not interfere with each other as each level of the tree is configured to transmit/receive packets on a different channel. In Fig. 2(c), P2 is transmitted on the receiving channel of the first-level nodes while P1 is transmitted on a different receiving channel for the third-level nodes. Note also that a third-level node will receive transmissions from several second-level nodes, instead of just one. We have omitted some of the transmission arrows in Fig. 2(c) to reduce clutter.

This results in a tree-based pipeline in which packets are disseminated in a ripple-wave-like fashion from the root. Except for the root node (which only transmits), all the nodes are either transmitting or receiving at all times once the pipeline is filled (see Fig. 2(d)). This allows Splash to achieve maximum possible end-to-end throughput.

The tree structure is needed to allow Splash to coordinate transmissions and channel assignment, also to ensure that each transmission is forwarded to every node in the network. Splash uses an underlying collection protocol like CTP [11] to derive its tree structure. We believe that our approach would incur minimal overhead as a CTP-like collection protocol is an integral part of most sensor network applications and we can make use of its existing periodic beacons in order to build the dissemination tree. Moreover, as CTP-like protocols are typically data-driven and they are designed to build stable trees by preferring stability over adaptability [1], diverting some of its periodic beacons for another use will not affect the stability of its data collection tree.

In practice, collection protocols often attempt to use the best links on the best channel (typically Channel 26) to build a tree. However, the performance of the other channels on such links is often not comparable to that of the best channel. So, if a dissemination tree is built using the default channel, the link quality on the same transmitter-receiver pair may be good on the default channel but poor on a different channel. On the other hand, building the dissemination tree on the poorest channel is also not a viable option since the network may not even be connected on such channels. Our approach therefore is to use the best channel (Channel 26) to build the dissemination tree at a lower transmission power but to use the maximum transmission power during dissemination. Our hypothesis is that the performance of different channels at the maximum transmission power is likely be comparable to that of the best channel at a lower transmission power.

Opportunistic Overhearing. In the transmission pipeline, each node is either receiving or transmitting. When a node is unable to successfully decode a transmission, it will be unable to relay the packet to the next stage. In such instances, instead of idling, such a node can switch to listening mode and attempt to recover the missing packet by overhearing the transmissions of its peers on the same level of the dissemination tree. This means that each node effectively has two opportunities

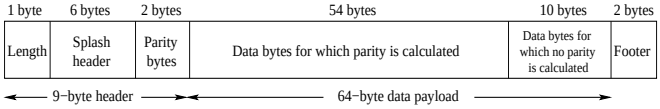


Figure 3: Packet format used in Splash.

to receive a given packet.

The decision to overhear transmissions has to be made before a node has completely received and decoded a packet, because to achieve constructive interference, a node needs to start calibrating its radio for transmission even before the packet to be transmitted is completely read from the radio hardware buffer. By the time a node completely reads, decodes and identifies packet corruption, its peers would have started calibrating their radio for transmission, and they begin transmissions before the node can switch over to overhearing mode which involves calibrating the radio for reception.

In order to address this issue, we add two bytes of parity information of the data payload bytes that are located before the last 12 bytes of the packet as the time required to receive these 12 bytes is the minimum amount of time necessary for verifying packet corruption and to either switch channel for overhearing in the case of corruption or to calibrate the radio for synchronous transmissions otherwise. Fig. 3 depicts format of a Splash packet with its default data payload size of 64 bytes. The parity of the first 54 bytes of data is computed and inserted in the header. This allows a receiving node to detect any corruption in these bytes as soon as it receives the 54^{th} data byte. If bit corruption is detected by the parity check, the reception of the current packet is aborted and the node immediately switches its channel to the receiving channel of its next hop nodes so that it can attempt to overhear the same packet while it is being forwarded by its peers in the next cycle. If corruption occurs within the last 12 bytes of the packet, the packet will not be recoverable with opportunistic overhearing.

4.2 Channel Cycling & Channel Assignment

Channel Cycling. It is well-known that the quality of channels is a function of both temporal and spatial variations. To ensure that nodes do not keep using the same (poor) channel, we use a different channel assignment between different rounds of dissemination in order to reduce the impact of the bad channels. In the case where the root transmits the same packet twice, by incorporating opportunistic overhearing and channel cycling, a node can potentially receive a packet 4 times, and possibly over 4 different channels. If the reception on one of the channels is bad, the packet could possibly be successfully decoded on one of the remaining channels.

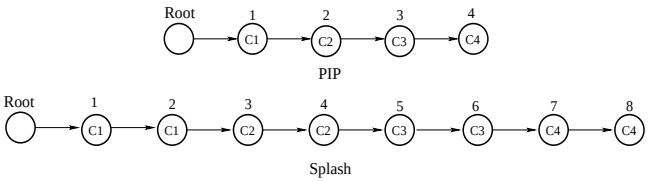


Figure 4: Channel assignment.

We coordinate channel switching between different dissemination rounds of Splash by transmitting a small 7-byte control packet. After every round of dissemination, the control packet is flooded from the root node over the tree pipeline by exploiting constructive interference 20 times. We do so because while there is a probability of some nodes not receiving this packet if we flood it only once, it has been shown that the probability that a node will receive such a small control packet over constructive interference is more than 0.999999 for ten retransmissions on Channel 26 [9]. We flood 20 times for good measure because we do not always use a channel that is as good as Channel 26. Also, we can afford to do so because flooding the packet 20 times takes only a few tens of milliseconds. After the completion of these 20 floods, a node that received the control packet at least once will switch to a pre-assigned channel on which it is expected to receive data packets in the next dissemination round. If a node still fails to receive the control packet, a timeout is used and the node recovers any missing data packets during local recovery.

Channel Assignment. In Fig. 4, we illustrate the channel assignment strategies for PIP and Splash using only four channels (C1, C2, C3, and C4). There are two key advantages of our assignment strategy. First, it allows more efficient channel cycling than PIP's method by allowing to cycle good channels in pairs on consecutive pipeline stages. Second, it supports a longer pipeline if interference extends to several hops as observed in a deployment on the Golden Gate Bridge [15]. However, in our strategy, we need to ensure that we do not use pairs of adjacent channels on consecutive pairs of stages as adjacent channels interfere with each other [29].

In our current implementation of Splash, we choose the ZigBee channels in such a way that they are either non-overlapping or only partially overlapping with the 3 most commonly used WiFi channels (channels 1, 6 and 11). On the testbeds which have network diameters not more than 9 hops, we observed that Splash's channel assignment strategy needs only four such ZigBee channels to avoid any interference.

4.3 Exploiting Transmission Density Diversity

We had shown in Section 3.1 that the effect of the number of transmitters (transmission density) on reception reliability for constructive interference does not follow a fixed trend but depends on the positions of the concurrent transmitters relative to the receiver.

Our key insight is that we can exploit *diversity in transmission density* to improve reliability, not by attempting to determine the optimal number of transmitters, but by transmitting the full data object twice using different transmission densities. In the first round, data is disseminated over the dissemination tree but only non-leaf nodes are asked to transmit. Since typically more than 50% of nodes in a tree are leaf nodes even at the lowest transmission power where the underlying network is connected [4], the number of concurrent transmitters is significantly reduced. In the second round, transmissions are made by all the nodes at each level of the tree. By using more transmitters, some nodes which were not reachable in the first round might now be reached. Moreover, a higher node density is also helpful in specific cases because of the capture effect as we discussed in Section 3.1.

4.4 XOR Coding

After two rounds of dissemination using different transmission densities, we observed in our experiments (see Section 5) that a considerable percentage of the nodes (about 50%) received most but not all the disseminated packets. This is a bad situation for local recovery because even though the number of missing packets may be small, there would be significant wireless contention if too many nodes attempted to recover the missing packets locally from their neighbors. This would significantly reduce the gain achieved through constructive interference by the first two rounds of dissemination.

While it is possible to perform a few more rounds of simple dissemination, we found that the potential gain was limited. This is because the missing packets are different among the different nodes and the root has no way of efficiently determining which exact packets are missing. If all packets are disseminated again, the overhead is very high with minimal gain.

This motivated us to use a third round of dissemination based on XOR coding instead. XOR coding is best suited for recovering missing packets if a node already has most of the packets and only a small portion is missing. Assume that a node already has a fraction p of the total packets. If the degree of the XOR packet is n (i.e. the coded packet is constructed by performing an XOR operation on n packets), then the likelihood that the packet is useful (i.e. that the receiving node had earlier received $n-1$ out of the n packets successfully) is $n(1-p)p^{n-1}$. This likelihood is maximized when $n = \frac{-1}{\ln(p)}$. We found in our experiments that p is about 95% after the first two rounds of dissemination, so in our current implementation, we set $n = 20 \approx \frac{-1}{\ln(0.95)}$.

In the third round, the payload in each packet is the result of 20 randomly chosen packets XORed together. To minimize the overhead, we do not indicate the identities of the packets used in the XOR operations in the packet header. Instead, we use the sequence number of the packet as a seed for choosing these packets based on a predefined pseudo-random function. This allows a receiver to decode packets without any additional overhead. In addition, like the first round of dissemination, only non-leaf nodes participate in forwarding XORed packets in the third round.

Naively, it might seem like it is sufficient to send $\frac{1}{20} = 5\%$ of the total number of packets. However, we found empirically (see Section 5.2) that such an approach is not sufficient to achieve a high packet recovery rate. Instead we send all the original packets with each original packet XORed with 19 randomly chosen packets. This ensures that every single packet is retransmitted at least once, and it also means that the third dissemination round is equivalent to the first two rounds in length.

We also considered using a fountain or rateless code during the "regular" dissemination rounds instead of introducing a third round of simple XOR-coded dissemination. However, we decided not to do so because of the associated decoding costs. In the experiments with Rateless Deluge [12], the decoding process can easily take more than 100 seconds for a 32-kilobyte data object. In comparison, Splash can disseminate the same object in about 25 seconds with simple XOR coding.

4.5 Local Recovery

After three rounds of dissemination, typically about 90% of the nodes would have downloaded the complete data object and most of the remaining nodes would have downloaded most of the object. This makes local recovery practical. Local recovery also allows the nodes to exploit spatial diversity and non-interfering nodes in different parts of the network can simultaneously recover the missing packets from their neighbors.

We implement a very simple CSMA/CA-based local recovery scheme on the default Channel 26. As Splash uses an underlying collection tree protocol to build its dissemination tree, a node will have link quality estimates for its neighboring nodes. A node with missing packets will send a bit vector containing information on the missing packets to a neighbor, starting with the one with the best quality link. If this neighbor has any of the missing packets, it will forward these packets to the

requesting node; if not, the requesting node will ask the next neighbor. If a node reaches the end of its neighbor list and it still has missing packets, it will start querying its neighbors afresh. Because the network is fully connected, this local recovery procedure is guaranteed to converge. Also, as most (about 90%) nodes already have the full data object, it converges quickly (see Section 5.2).

4.6 Implementation Challenges

The key requirement for constructive interference is that nodes have to transmit the same packet at the same time. Glossy satisfies this requirement as a set of nodes receiving a packet are synchronized to the SFD (Start Frame Delimiter) interrupt from the radio hardware (ChipCon2420 (CC2420)) signalling the end of the reception of a packet. Splash is built upon the source code for Glossy [9]. The challenge is to transform the Glossy code into a dissemination protocol while retaining its capability to perform synchronized transmissions.

Channel Switching. First, we added the capability for switching channels for the pipelining operations. Upon receiving a packet, a node switches its channel to that of its next hop nodes, transmits the received packet, and then switch back to its receiving channel to listen for the next packet. Channel switching for transmission has to be performed only after completely receiving an incoming packet and before submitting the transmit request to the radio for forwarding the received packet. The time taken for channel switching cannot vary too much across nodes as such variations desynchronize their submission of the transmit request.

On the other hand, as the clocks of microcontrollers are not synchronized across nodes, the time taken for channel switching can vary from node to node. Our goal is to minimize such variations by enabling channel switching by executing only a minimal number of instructions between the completion of the reception of a packet and the submission of the request for its transmission (forwarding).

The operation of channel switching involves writing to the frequency control register of the radio hardware and then calibrating the radio for transmission. The action of writing to a register in turn involves enabling the SPI (Serial Peripheral Interface) communication by pulling down a pin on the radio, communicating the address of the register to be written, writing into the register and finally disabling the SPI access. Similarly, radio calibration involves enabling the SPI, transmitting a command strobe requesting for calibration and disabling the SPI. While the actual operations of calibration and register access take more or less constant time, enabling the SPI twice, once for the register access and another time for transmitting the command strobe can add to the variability and cause desynchronization. In order to avoid this, we exploit the multiple SPI accesses capability of the CC2420 radio which allows register access and to send strobes continuously without having to re-enable the SPI. Using this feature, we enable the SPI only once at the beginning of a channel switching operation.

We further minimize the number of in-between instructions to be executed by splitting the channel switching into two phases. In the first phase, we enable the SPI access and communicate the address of the frequency control register to the radio. In the second phase, we write into the register and transmit the command strobe to start transmit calibration. The number of in-between instructions is minimized by the fact that we overlap the first phase with the packet reception by the hardware. This way we execute only the second phase between the completion of the reception of a packet and the submission of the request for its transmission.

Accessing External Flash. Another important requirement for a dissemination protocol is that the data object has to be written into the external flash because typical sensor devices only have a small amount of RAM. In Splash, since a node is always either transmitting or receiving a packet at any given point of time, flash access has to be overlapped with a radio operation, so we write a packet to the flash while it is being transmitted by the radio. As flash access is faster than the radio transmission rate [8], the write operation completes before the radio transmission and does not cause any synchronization issues.

Handling GCC Compiler Optimizations. Although the arrival of the SFD interrupt indicating completion of the reception of a packet is synchronized across nodes, its service delay varies from node to node. The key implementation feature of Glossy is that each node executes a different number of “nop” assembly instructions based on its interrupt service delay so that all the nodes submit a request to the radio hardware at the same time for forwarding the received packet.

The most challenging problem faced during implementation is the fact that the optimization feature of the GCC compiler affects the service delay for the SFD interrupt (perhaps for some other interrupts too). Without enabling compiler optimizations, the resulting binary (a collection application coupled with Splash) was too large to fit into a sensor device. However, with optimizations enabled, minor changes to parts of the code could change the service delay, making it difficult to set the number of “nop” instructions to be executed. However, this issue can be handled as changes to the code will change the minimum duration required for servicing the SFD interrupt. While it is tedious, we can account for this change by measuring the minimum service delay after making a

change that affects the service delay. The same procedure was followed in the development of Glossy.

5 Performance Evaluation

In this section, we present the results of our evaluations carried out on the Indriya [3] and Twist [13] testbeds.

Indriya is a three-dimensional sensor network with 139 TelosB nodes spanning three floors of a building at the National University of Singapore. We compare Splash against TinyOS's DelugeT2, the de facto standard data dissemination protocol for sensor networks. For Splash, a low power setting of -10 dBm is used to build the dissemination tree and the maximum transmission power of 0 dBm is used for dissemination. For DelugeT2, we use the maximum transmission power of 0 dBm on Channel 26. We disseminate a 32-kilobyte data object for both Splash and DelugeT2.

Splash has a data payload of 64 bytes in every packet. We will show in Section 5.3 that the performance of DelugeT2 varies depending on the packet size, but there is no clear relationship between packet size and performance. Also, the impact of packet size is relatively insignificant. In this light, we adopted the default payload size of 22 bytes for DelugeT2 in our experiments on Indriya, unless otherwise stated.

The Twist sensor testbed is deployed at the Berlin University and currently it has 90 Tmote Sky devices. The experimental settings on Twist are similar to that on Indriya, except for the following differences: first, we use a lower transmission power of -15 dBm to build the dissemination tree for Splash, as Twist is a much smaller deployment than Indriya. Second, instead of using TinyOS's DelugeT2, we use Contiki's Deluge. This is because to execute TinyOS's DelugeT2, we need to execute some tools on a machine connected to base-station nodes (root nodes) which is difficult in a case of a remote testbed like Twist. We retain default settings of Contiki's Deluge including 0 dBm transmission power and Channel 26. Moreover, its default payload size of 64 bytes is also retained as Twist is a smaller deployment with stable links of good quality.

We execute Splash as a part of Contiki collection protocol [16] and Splash accesses the collection protocol's data in order to build the dissemination tree. We execute DelugeT2 as a part of TinyOS collection protocol, CTP [11] by coupling the DelugeT2 with the TinyOS's standard "TestNetwork" application with its default settings. We also compare Splash against DelugeT2 running as a standalone golden image (GI) without CTP. Note that the standalone version is seldom used in practice, as a dissemination protocol is only useful when coupled with a real application.

5.1 Summary of Testbed Results

The summary of our results on Indriya and Twist are shown in Tables 2 and 3 respectively. For each experimental run, we randomly picked a node as the root of the dissemination tree. In the tables, "size" indicates the depth of the Splash's dissemination tree, and R1, R2 and R3 indicate the average *reliability* per node after the first, second and third rounds of dissemination respectively. By reliability, we refer to the fraction of the data object that has been successfully downloaded by a node. $N_{R3-100\%}$ is the proportion of nodes that have 100% of the disseminated data object after the third round. Recall that XOR coding is employed in the third dissemination round. R_{lr} indicates the average reliability per node after local recovery. T_{Splash} is the time taken for Splash to complete the dissemination, i.e. when *every* node in the network has successfully downloaded the entire data object. Similarly, $T_{DelugeT2+CTP}$, $T_{DelugeT2GI}$, and T_{Deluge} are the corresponding times taken for DelugeT2 with CTP, DelugeT2 as standalone golden image, and Contiki's Deluge respectively, to complete the dissemination.

Indriya Testbed. We observe from Table 2 that on average Splash takes about 25 seconds (see T_{Splash}) to complete the dissemination of a 32-kilobyte object, while DelugeT2 coupled with CTP takes about 524 seconds. Splash reduces dissemination time by an average factor of 21.06 (93.68% reduction). Splash also outperforms DelugeT2 running as a standalone golden image by a factor of 12.43 (89.2% reduction). One obvious drawback of DelugeT2 is that there is a large variation in its dissemination time, ranging from 209 seconds to 1300 seconds. This is likely due to variations in the conditions of the default Channel 26 since DelugeT2 uses a fixed channel. By using multiple rounds of dissemination, opportunistic overhearing, and channel cycling, Splash is more resilient to variations in the channel conditions. In particular, a node in Splash has the potential to receive a packet up to 6 times, and more importantly, on up to 6 different channels. If the quality of one or two channels is bad, a packet can potentially be successfully decoded on one of the other remaining channels.

We also observe that the dissemination time for DelugeT2 as golden image is usually less than DelugeT2 with CTP. This is because dissemination traffic in the latter case has to contend with CTP's application traffic. While Splash relies on Contiki's Collection Protocol to build its dissemination tree, like Glossy [9], Splash disables all the interrupts other than the Start Frame Delimiter interrupt during its three rounds of dissemination where constructive interference is exploited. This means that any underlying application will be temporarily suspended and most of the Splash's traffic will be served exclusively without interference from any application traf-

Table 2: Summary of results for 139-node Indriya testbed.

		Splash						DelugeT2	
Tree No.	size [hops]	R1 [%]	R2 [%]	R3 [%]	$N_{R3-100\%}$ [%]	R_{lr} [%]	T_{Splash} [sec]	$T_{DelugeT2+CTP}$ [sec]	$T_{DelugeT2GI}$ [sec]
1	5	84.54	97.23	98.47	91.30	100.00	22.49	1300	924
2	6	86.52	96.91	98.58	92.03	100.00	22.61	286	160
3	7	76.68	94.62	97.80	86.23	100.00	23.18	209	286
4	7	88.02	96.12	97.78	92.75	100.00	23.74	218	158
5	9	76.97	93.65	96.69	81.88	100.00	23.86	649	180
6	7	76.73	95.27	98.16	89.86	100.00	25.98	610	160
7	7	80.75	93.51	96.98	89.13	100.00	26.25	365	379
8	7	83.57	94.43	96.01	87.68	100.00	26.89	377	277
9	5	82.46	95.26	97.47	85.51	100.00	28.09	676	313
10	8	84.28	94.92	96.70	86.23	100.00	28.39	550	216
Average		82.05	95.19	97.46	88.26	100.00	25.15	524	305.3

Table 3: Summary of results for 90-node Twist testbed.

		Splash						Deluge
Tree No.	size [hops]	R1 [%]	R2 [%]	R3 [%]	$N_{R3-100\%}$ [%]	R_{lr} [%]	T_{Splash} (for a 32KB file) [sec]	T_{Deluge} (for a 2KB file) [sec]
1	4	90.58	97.09	99.22	94.38	100.00	20.07	356.60
2	4	81.08	94.70	99.31	92.13	100.00	20.19	431.48
3	4	86.53	96.19	98.00	91.01	100.00	22.79	351.67
4	4	78.64	94.10	98.12	84.09	100.00	23.37	518.19
5	4	81.42	93.95	97.98	89.89	100.00	23.41	467.00
6	4	78.04	93.55	96.82	85.39	100.00	26.66	439.81
7	4	83.90	95.18	97.54	89.89	100.00	26.79	345.28
8	4	83.70	93.64	96.45	84.27	100.00	27.32	388.68
9	6	81.58	93.35	97.02	85.39	100.00	27.45	484.10
10	5	80.78	93.09	97.11	85.39	100.00	29.25	397.59
Average		82.62	94.48	97.76	88.18	100.00	24.73	418.04

fic. On the other hand, because DelugeT2 is built on TinyOS services, it is not possible to completely disable all the interrupts during its execution. DelugeT2 as golden image provides us with the baseline performance without interference from application traffic. Note that application suspension in Splash is not a problem as most sensor applications have no real-time requirements. Moreover, interrupts are re-enabled long before the completion of dissemination, before starting the round of local recovery that dominates the dissemination time (see Fig. 7). Applications are suspended for only about 8.2 seconds while disseminating the 32-kilobyte object.

Twist Testbed. As shown in Table 3, Splash's performance on Twist is similar to that on Indriya. It takes about 25 seconds on average to complete the dissemination of a 32-kilobyte object. On the other hand, because the Contiki implementation of Deluge is less efficient, it takes about 418 seconds to disseminate a much smaller object of 2 kilobytes. Note that Contiki Deluge is a thin implementation with minimal functionality that allows only minimal changes to its settings. Hence, Splash is able to significantly outperform Contiki's Deluge even when disseminating a data object that is 16 times larger. Splash effectively achieves a *network-wide* goodput of above 10 kilobits/sec per node on both Indriya and Twist testbeds, which is higher than that of all existing network-wide data dissemination protocols [6, 5, 7, 10, 12, 30, 18, 14, 17] in the literature.

Memory Consumption. Splash not only outperforms DelugeT2 in terms of speed, it is also much more efficient than DelugeT2 in terms of memory usage. Splash requires only 11.38 kilobytes of ROM and 0.13 kilobytes of RAM whereas DelugeT2 requires 21.01 and 0.81 kilobytes of ROM and RAM respectively. Hence, Splash uses 9.63 kilobytes of ROM and 0.68 kilobytes of RAM less than DelugeT2. Given that it is not uncommon for sensor devices to have only about 48 and 10 kilobytes of ROM and RAM respectively, these are significant savings in memory, that will be available for use by sensor applications.

Comparison to Existing Protocols. Because we were not able to obtain the code for the state-of-the-art dissemination protocols ECD [6] and MT-Deluge [10], we used an indirect method to compare Splash against them and other existing dissemination protocols [5, 12, 30, 18]. It turns out that these protocols are all evaluated against Deluge and so we have a convenient common baseline with which to compare against without having to implement and evaluate them individually. We present the relative performance of Splash to these protocols in Ta-

Table 4: Comparison of Splash to existing protocols.

Protocol	No. of nodes	File size [KB]	Reduction factor
MNP ([18], 2005)	100	5	1.21
MC-Deluge ([30], 2005)	25	24.3	1.6
Rateless Deluge ([12], 2008)	20	0.7	1.47
ReXOR ([5], 2011)	16	4	1.53
ECD ([6], 2011)	25	10	1.44
MT-Deluge ([10], 2011)	20	0.7	2.42
Splash	139	32	21

ble 4. In the fourth column, we present the reduction factor achieved by each of these algorithms compared to Deluge. It is evident that Splash's performance is significantly better than that of the state-of-the-art protocols. Not only is Splash faster by an order of magnitude, but we also achieve this improvement on a larger testbed and with a bigger file than all the previous algorithms. Note also that most of the results for the existing protocols in Table 4 are compared against classical Deluge (Deluge 2.0 of TinyOS 1), which is in fact slower than DelugeT2, against which we have compared Splash.

Energy Consumption. Duty cycling is typically adopted by applications that transmit a data packet once in a while, and not for dissemination that involves transfer of large amounts of data [25]. As duty-cycled transmissions involve a large overhead such as the transmission of a long preamble before sending every packet [22], they make dissemination significantly more expensive in terms of both time and energy. This drives most of the dissemination protocols in the literature [14, 12, 5, 6, 10, 30] to keep the radio awake during dissemination as required in Splash. Therefore, energy consumption is directly proportional to the dissemination time. This means Splash reduces energy consumption by the same factor by which it reduces dissemination time.

5.2 Contribution of Individual Techniques

In order to achieve a high reliability, Splash incorporates four key techniques: (1) XOR coding; (2) transmission density diversity; (3) opportunistic overhearing; and (4) channel cycling. We now evaluate the contribution of these techniques together with local recovery.

XOR Coding. We employ XOR coding in the third round of dissemination. The goal of using XOR coding is to significantly increase the number of nodes that successfully receive the entire file so that local recovery will be much more efficient. We present the proportion of nodes that achieve a reliability of 100% before and after the third round of XORed dissemination on Indriya in Table 5. The largest improvement was observed for the fifth tree where the use of XOR coding increases the percentage of nodes having the full object from 9.42% to 81.88%. On average, the number of nodes with the full data object is more than doubled. Similar results were observed on the Twist testbed.

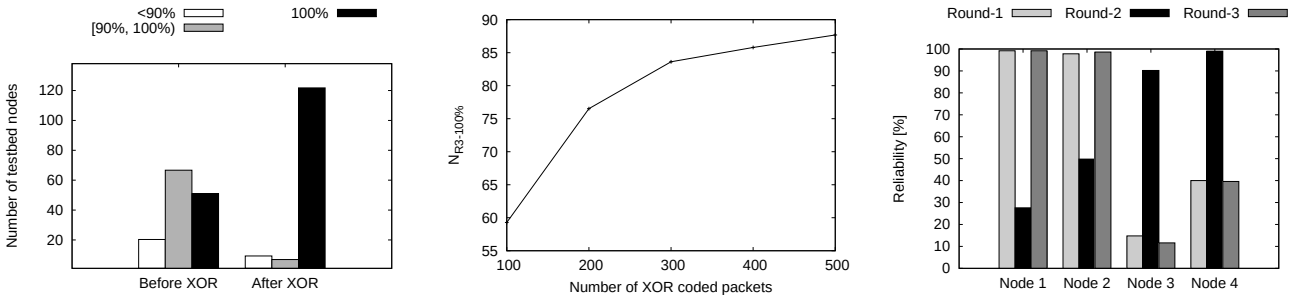
Table 5: Proportion of nodes with 100% reliability before and after the third round of XOR coding on Indriya.

Tree No.	Before XOR	After XOR
1	57.25	91.30
2	50.72	92.03
3	21.74	86.23
4	33.33	92.75
5	9.42	81.88
6	26.09	89.85
7	23.91	89.13
8	47.10	87.68
9	51.45	85.51
10	48.55	86.23
Avg.	36.96	88.26

To validate our hypothesis that XOR's effectiveness comes from helping the nodes that already have most of the packets, we plot in Fig. 5(a) the average number of nodes per tree found in the three different bins of reliability for Indriya, namely $<$90%, between 90% and 100%, and 100%. We see that before the third dissemination round, there are about 20 nodes in the first bin with reliability less than 90% and 67 nodes in the second bin with reliability between 90% and 100%. XOR coding is able to move most of these nodes in the first 2 bins into the third bin with 100% reliability. In particular, XOR coding can reduce the size of the second bin from 67 to 7, to give a total of 122 nodes in the 100% bin. Similar results were observed on the Twist testbed.

For the 32-kilobyte file that we used in our experiments, we XOR coded and transmitted each of the 500 packets (with a packet payload size of 64 bytes) constituting the file. One pertinent question is whether we can do with fewer packets since an XORed packet already contains the information of 20 packets. In Fig. 5(b), we present a plot of $N_{R3-100\%}$ against the number of XOR coded packets transmitted, averaged over five experimental runs on different dissemination trees. Note that only about 37% of the nodes have downloaded the whole file after the first two rounds of dissemination. It is clear from Fig. 5(b) that 100 packets is not enough, and that there is a significant improvement in $N_{R3-100\%}$ as we transmit more coded packets until about 400 packets. From 400 to 500 packets, we obtain only a small increase of about 2% in $N_{R3-100\%}$ (about 3 nodes). While the improvement is small, since local recovery over CSMA/CA can be expensive, we decide to transmit all the 500 coded packets for completeness since the extra 100 transmissions take only an extra 0.56 seconds.

Transmission Density Diversity. To understand the effectiveness of our attempt to exploit transmission density diversity, we disseminate a 32-kilobyte data object without the leaf nodes transmitting (*Round-1*). Imme-



(a) Distribution of avg. no. nodes across three reliability bins. (b) $N_{R3-100\%}$ Vs. no. of XOR coded transmissions. (c) Effectiveness of transmission density diversity.

Figure 5: Contributions of XOR coding and transmission density diversity.

Table 6: Performance of Splash with and without opportunistic overhearing.

	With overhearing			Without overhearing		
No.	N_{lrpkts}	$N_{R3-100\%}$	T_{Splash} [sec]	N_{lrpkts}	$N_{R3-100\%}$	T_{Splash} [sec]
1	1860	78.99	28.28	5536	79.71	44.07
2	1433	89.13	23.64	2415	84.06	36.19
3	1876	89.13	27.00	2531	85.51	34.98
4	420	93.48	21.94	1529	90.58	24.73
5	1356	90.58	22.68	1131	83.33	26.75
Avg.	1389	88.26	24.71	2628.4	84.64	33.34

diately after that, the object is disseminated again but with all the nodes transmitting (*Round-2*). Finally, we repeated the transmission without the leaf nodes transmitting (*Round-3*). This approach allows us to determine whether a node gains from a low transmission density or a node gains from a high transmission density. The same channel assignment is used for all three rounds.

We run this experiment five times on a dissemination tree. As an illustration, we present the reliability observed on four nodes in each of the three rounds of an experimental run in Fig. 5(c). Nodes 1 and 2 benefit from a low transmission density (without leaves) as the achieved reliability is higher in the first and third rounds of dissemination. On the other hand, nodes 3 and 4 benefit from a high transmission density with all nodes transmitting. On average, we found that 38.7% of the nodes benefit from a low transmission density and achieve higher reliability than that for the higher transmission density. The proportion of nodes that benefit from a high transmission density is lower, about 18.1% achieve higher reliability at the higher transmission density compared to that for the lower transmission density. Nevertheless, the key insight is that by varying the number of transmitters between transmission rounds, different sets of nodes will correctly decode packets over different transmission rounds.

Opportunistic Overhearing. Table 6 compares the performance of Splash with and without opportunistic overhearing on five dissemination trees on Indriya. The table shows the total number of packets to be recovered during local recovery (N_{lrpkts}) together with $N_{R3-100\%}$ and T_{Splash}. We found that T_{Splash} is increased by 8.6 seconds on average when opportunistic overhearing is not employed. Quite clearly, this is because the number of corrupted/missed packets N_{lrpkts} is typically larger when there is no overhearing, as observed on the first four of the five considered trees. In the case of the fifth tree, we found that overhearing did not lead to a smaller number of corrupted/missed packets N_{lrpkts}. However, Splash with overhearing is still faster because the proportion of nodes that have downloaded the full data object after 3 dissemination rounds ($N_{R3-100\%}$) is larger. In other words, overhearing helps not just by increasing the likelihood that packets are transmitted successfully, it also helps by ensuring that more nodes have downloaded the complete file.

Channel Cycling. In order to evaluate the effectiveness of channel cycling, we compare Splash with channel cycling against Splash without channel cycling i.e., by using the same channel assignment in all three dissemination rounds. We plot the resulting performance for five dissemination trees on Indriya in Table 7. Without channel cycling, there is a drop in both reliability (R3) and the percentage of nodes having the full data object after the third round of dissemination ($N_{R3-100\%}$). In addition to better average-case performance, we also see that channel cycling can significantly reduce the variance in performance. We see that T_{Splash} varies between 22.49 s and 28.39 s with channel cycling, while it varies between 26.24 s and 45.08 s without.

Local Recovery. After three rounds of dissemination, about 88% of the nodes would have successfully received the entire data object on average on both of the testbeds (see Column $N_{R3-100\%}$ in Tables 2 and 3). In Fig. 6, we plot the CDF of the reliability of those nodes that did not successfully receive the complete file after three rounds of dissemination. We see that among these nodes, only about 3% and 1% have less than 10% of the data on Indriya and Twist respectively. About 40% have at least 90% of the data object. In Fig. 7, we present the

Table 7: Performance of Splash with and without channel cycling.

No.	With cycling			Without cycling		
	R3	$N_{R3-100\%}$	T_{Splash} [sec]	R3	$N_{R3-100\%}$	T_{Splash} [sec]
1	96.98	89.13	26.25	92.33	76.81	45.08
2	98.16	89.86	25.98	95.56	86.23	26.24
3	96.69	81.88	23.86	92.15	73.19	34.79
4	98.47	91.30	22.49	91.86	79.71	34.58
5	96.70	86.23	28.39	95.61	85.51	31.51
Avg.	97.40	87.68	25.39	93.50	80.29	34.44

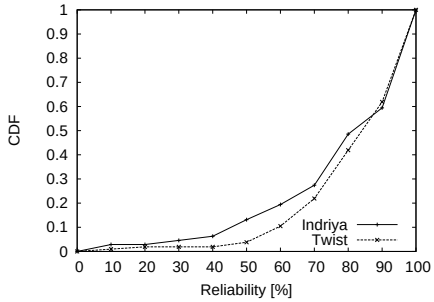


Figure 6: Distribution of the reliability of nodes with reliability less than 100%.

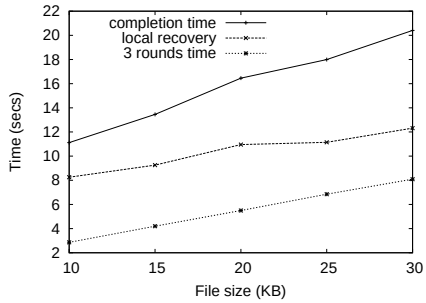


Figure 7: Breakdown of completion time for different file sizes.

time taken for local recovery for data objects of different sizes on Indriya. We also present the time taken for the first three rounds of dissemination and the completion time on the same graph. As expected, the time spent in the first three rounds increases linearly with the object size whereas time taken for local recovery is not strictly linear due to the variations in the number of packets to be recovered and the randomness involved in CSMA/CA.

5.3 Effect of Packet Size

It is well-known that the reliability of constructive interference decreases as packet size increases [9, 28]. To justify our choice of 64 bytes for the Splash payload, we compare the performance of Splash for the default payload size against the maximum possible payload size of 117 bytes (which results in a maximum-sized packet

Table 8: Performance of Splash for two different payload sizes.

64 bytes				117 bytes			
R1	R2	R3	$N_{R3-100\%}$	R1	R2	R3	$N_{R3-100\%}$
85.12	96.82	98.68	92.03	78.19	91.60	94.47	78.26
86.35	96.64	98.30	91.30	80.58	92.04	93.52	78.99
89.41	96.90	98.83	93.48	81.91	94.65	96.45	82.61
84.64	96.20	97.67	88.41	78.96	92.59	95.20	82.61
84.49	96.99	98.29	89.13	72.08	87.54	90.35	70.29
86.00	96.71	98.35	90.87	78.34	91.68	94.00	78.55

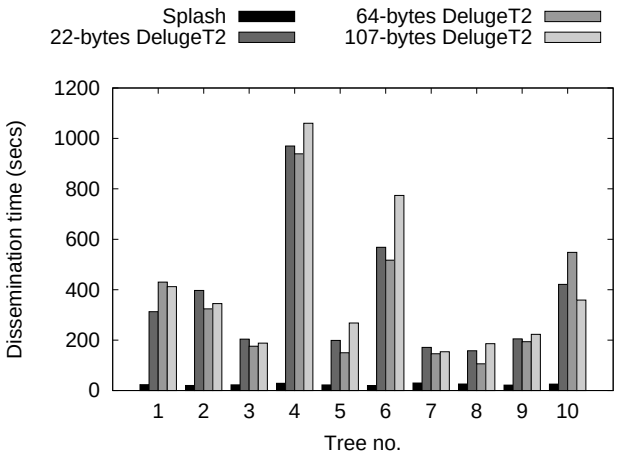


Figure 8: Comparison of Splash against DelugeT2 configured with different payload sizes on Indriya.

of 128 bytes) for five dissemination trees on Indriya in Table 8. As expected, reliability decreases with the larger payload size, so we set the default payload size for Splash to 64 bytes.

It is known that the performance of DelugeT2 varies with packet size [2], so in order to compare Splash fairly to DelugeT2, we also investigated the performance of DelugeT2 for different payload sizes. We constructed 10 random dissemination trees on Indriya, and on each of them we disseminated a 32-kilobyte object using Splash and DelugeT2 configured with payload sizes of 22 bytes (default), 64 bytes, and the maximum value of 107 bytes. We ensured that Splash and the three versions of DelugeT2 were executed back-to-back on each of the dissemination trees so as to minimize the temporal variations in channel conditions across these executions. The results are shown in Fig. 8. For DelugeT2, we found that while there was some variation in the average dissemination times depending on the payload size and the payload size that achieves the best performance depends on the actual network conditions, the differences in performance are not significant, at least not when compared to the dissemination times achieved by Splash.

6 Conclusion

We propose Splash, a fast and scalable dissemination protocol for wireless sensor networks, that exploits constructive interference and channel diversity to achieve speed and scalability. To achieve high reliability, Splash incorporates the use of transmission density diversity, opportunistic overhearing, channel-cycling, and XOR coding. We demonstrated with experiments on two large multihop sensor networks that Splash can achieve an order of magnitude reduction in dissemination time compared to state-of-the-art dissemination protocols.

Acknowledgements

We would like to thank the anonymous reviewers and our shepherd, Rodrigo Fonseca for their valuable comments and suggestions. This work was partially supported by the NRF Singapore through the SMART (R-252-002-430-592) program.

References

[1] ALIZAI, M. H., LANDSIEDEL, O., LINK, J. A. B., GOTZ, S., AND WEHRLE, K. Bursty Traffic over Bursty Links. In *Proceedings of SenSys* (2009).

[2] C., R. K., SUBRAMANIAN, V., ULUAGAC, A. S., AND BEYAH, R. SIMAGE: Secure and Link-Quality Cognizant Image Distribution for Wireless Sensor Networks. In *Proceedings of GLOBECOM* (2012).

[3] DODDAVENKATAPPA, M., CHAN, M. C., AND ANANDA, A. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *Proceedings of TRIDENTCOM* (2011).

[4] DODDAVENKATAPPA, M., CHAN, M. C., AND LEONG, B. Improving Link Quality by Exploiting Channel Diversity in Wireless Sensor Networks. In *Proceedings of RTSS* (2011).

[5] DONG, W., CHEN, C., LIU, X., BU, J., AND GA, Y. A Lightweight and Density-Aware Reprogramming Protocol for Wireless Sensor Networks. In *IEEE TRANSACTIONS ON MOBILE COMPUTING* (2011).

[6] DONG, W., LIU, Y., WANG, C., LIU, X., CHEN, C., AND BU, J. Link Quality Aware Code Dissemination in Wireless Sensor Networks. In *Proceedings of ICNP* (2011).

[7] DONG, W., LIU, Y., WU, X., GU, L., AND CHEN, C. Elon: Enabling Efficient and Long-Term Reprogramming for Wireless Sensor Networks. In *Proceedings of SIGMETRICS* (2010).

[8] DUQUENNOY, S., ÖSTERLIND, F., AND DUNKELS, A. Lossy Links, Low Power, High Throughput. In *Proceedings of SenSys* (2011).

[9] FERRARI, F., ZIMMERLING, M., THIELE, L., AND SAUKH, O. Efficient Network Flooding and Time Synchronization with Glossy. In *Proceedings of the IPSN* (2011).

[10] GAO, Y., BU, J., DONG, W., CHEN, C., RAO, L., , AND LIU, X. Exploiting Concurrency for Efficient Dissemination in Wireless Sensor Networks. In *Proceedings of DCOSS* (2011).

[11] GNAWALI, O., FONSECA, R., JAMIESON, K., MOSS, D., AND LEVIS, P. Collection Tree Protocol. In *Proceedings of SenSys* (2009).

[12] HAGEDRON, A., STAROBINSKI, D., AND TRACHTENBERG, A. Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks using Random Linear Codes. In *Proceedings of IPSN* (2008).

[13] HANDZISKI, V., KOPKE, A., WILLIG, A., AND WOLISZ, A. TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Network. In *Proceedings of REALMAN* (2006).

[14] HUI, J. W., AND CULLER, D. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *Proceedings of SenSys* (2004).

[15] KIM, S., PAKZAD, S., CULLER, D. E., DEMMEL, J., FENVES, G., GLASER, S., AND TURON, M. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In *Proceedings of IPSN* (2007).

[16] KO, J., ERIKSSON, J., TSIFTES, N., DAWSON-HAGGERTY, S., DURVY, M., VASSEUR, J., TERZIS, A., DUNKELS, A., AND CULLER, D. Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks. In *Proceedings of SenSys* (2011).

[17] KULKARNI, S. S., AND ARUMUGAM, M. INFUSE: A TDMA based Data Dissemination Protocol for Sensor Networks. Tech. rep., Michigan State University, 2004.

[18] KULKARNI, S. S., AND WANG, L. MNP: Multihop Network Reprogramming Service for Sensor Networks. In *Proceedings of ICDCS* (2005).

[19] LEVIS, P., AND CULLER, D. Mate: a Virtual Machine for Tiny Networked Sensors. In *Proceedings of ASPLOS* (2002).

[20] LEVIS, P., PATEL, N., CULLER, D., AND SHENKER, S. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proceedings of NSDI* (2004).

[21] LIANG, C.-J. M., PRIYANTHA, N. B., , LIU, J., AND TERZIS, A. Surviving Wi-Fi Interference in Low Power ZigBee Networks. In *Proceedings of SenSys* (2010).

[22] MOSS, D., AND LEVIS, P. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Tech. rep., Technical Report SING-08-00, Stanford University, 2008.

[23] OSTERLIND, F., AND DUNKELS, A. Approaching the Maximum 802.15.4 Multihop Throughput. In *Proceedings of HotEmNets* (2008).

[24] RAMAN, B., CHEBROLU, K., BIJWE, S., AND GABALE, V. PIP: A Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer. In *Proceedings of SenSys* (2010).

[25] ROSSI, M., BUI, N., ZANCA, G., STABELLINI, L., CREPALDI, R., AND ZORZI, M. SYNAPSE++: Code Dissemination in Wireless Sensor Networks Using Fountain Codes. In *IEEE TRANSACTIONS ON MOBILE COMPUTING* (2010).

[26] SRINIVASAN, K., JAIN, M., CHOI, J. I., AZIM, T., KIM, E. S., LEVIS, P., AND KRISHNAMACHARI, B. The K-Factor: Inferring Protocol Performance Using Inter-link Reception Correlation. In *Proceedings of Mobicom* (2010).

[27] WANG, Q., ZHU, Y., AND CHENG, L. Reprogramming Wireless Sensor Networks: Challenges and Approaches. In *IEEE Network Magazine* (2006).

[28] WANG, Y., HE, Y., MAO, X., LIU, Y., HUANG, Z., AND YANG LI, X. Exploiting Constructive Interference for Scalable Flooding in Wireless Networks. In *Proceedings of INFOCOM* (2012).

[29] WU, Y., STANKOVIC, J. A., HE, T., LU, J., AND LIN, S. Realistic and Efficient Multi-Channel Communications in Wireless Sensor Networks. In *Proceedings of INFOCOM* (2008).

[30] XIAO, W., AND STAROBINSKI, D. Poster Abstract: Exploiting Multi-Channel Diversity to Speed Up Over-the-Air Programming of Wireless Sensor Networks. In *Proceedings of SenSys* (2005).