SYREN: Synergistic Link Correlation-Aware and Network Coding-Based Dissemination in Wireless Sensor Networks

S. M. Iftekharul Alam, Salmin Sultana, Y. Charlie Hu, Sonia Fahmy
Purdue University
{alams, ssultana, ychu, fahmy}@purdue.edu

***Abstract*—Rapid flooding is necessary for code updates and routing tree formation in wireless sensor networks. Link correlation-aware collective flooding (CF) is a recently proposed technique that provides a substrate for efficiently disseminating a single packet. Applying CF to *multiple* packet dissemination poses several challenges, such as reliability degradation, redundant transmissions, and increased contention among node transmissions. The varying link correlation observed in real networks makes the problem harder. In this paper, we propose a multi-packet flooding protocol, SYREN, that exploits the synergy among link correlation and network coding. In particular, SYREN exploits link correlation to eliminate the overhead of explicit control packets in networks with high correlation, and uses network coding to pipeline transmission of multiple packets via a novel, single yet scalable timer per node. SYREN reduces the number of redundant transmissions while achieving near-perfect reliability, especially in networks with low link correlation. Testbed experiments and simulations show that SYREN reduces the average number of transmissions by 30% and dissemination delay by more than 60% while achieving the same reliability as state-of-the-art protocols.**

I. Introduction

In wireless sensor networks, *flooding* is used for code updates [1], [2], reconfiguration, and routing tree creation [3], [4]. Flooding involves the dissemination of multiple packets within an acceptable delay bound. Since these packets pertain to a single operation, flooding algorithms must be designed for efficiency and reliability.

Most current flooding protocols use an explicit ACK from each receiver, causing collisions. To avoid the heavy ACK load, Zhu *et al.* proposed a novel Collective Flooding (CF) protocol [5] to exploit the *link correlation* among neighboring nodes. The phenomenon of link correlation, which had previously been overlooked, allows a sender to infer the success of a transmission to a receiver based on ACKs from *correlated* neighbors. In other words, the traditional *direct* ACK per receiver is transformed into *collective* ACKs. With collective ACKs and an efficient forwarder selection mechanism, CF greatly reduces dissemination delay while achieving the same reliability as state-of-the-art solutions.

CF, however, was designed to flood a single packet at a time and cannot be easily extended to support multi-packet flooding. There are two obvious extensions to CF for the multi-packet scenario: (i) packet transmissions at fixed time intervals, or (ii) consecutive packet transmissions without significant delay. Since CF requires $\sim$0.9-2.34 seconds on average to disseminate a single packet, even for a small network of 19-36 nodes, the total dissemination delay for all packets will be exceedingly high in the first case. The second approach allows successive transmissions with minimal delay. However, it imposes these non-trivial challenges: (1) *Reliability degradation*: Due to varying link correlation in a neighborhood, different nodes may receive different subsets of transmitted packets; (2) *Redundant transmissions*: To achieve high reliability, the sender needs a per-packet reception record for each neighbor and needs to retransmit all missing packets, which leads to redundant transmissions; and (3) *Increased contention*: Performance further degrades when the receivers possessing different subsets of packets experience packet loss due to contention and collision.

To quantify the extent of link correlation in wireless sensor networks, we perform testbed experiments in different network settings and environments [6]. Experimental results show that the degree of link correlation varies in practical networks: around 50-65% of the link pairs exhibit high ($> 80\%$) correlation, about 25-30% pairs see moderate ($80 - 40\%$) correlation, and the rest are poorly correlated ($< 40\%$). Similar observations have been reported in a recent empirical study [7]. These results suggest that network coding protocols work well under low link correlation whereas CF [5] may not perform well in networks with low or medium correlation.

Network coding is a well-known technique for addressing the reliability, contention, and throughput concerns for multi-packet flooding, and a number of *network coding*-based dissemination protocols have been proposed in the literature [8], [9], [10]. However, these protocols are oblivious to spatial link correlation; they do not take advantage of such link correlation in estimating packet reception. Instead, they rely on explicit acknowledgments which can incur significant overhead.

In this paper, we observe that network coding and link correlation are synergistic techniques for effective multi-packet flooding, and propose SYREN, a multi-packet flooding (or **data dissemination**) protocol that exploits this synergy. While collective ACKs with link correlation reduce the number of transmissions, network coding coordinates the flooding of multiple packets in short dissemination time and near-perfect reliability. In addition, network coding provides robust performance under different network conditions, *e.g.,* when link correlation is low.

SYREN seamlessly integrates network coding with link correlation. It floods a large data object in the form of a *batch* of packets, where a sender transmits a random linear

TABLE I
COMPARISON AMONG SYREN AND EXISTING FLOODING APPROACHES.

Properties	Deluge	Rateless Deluge	ECD	CF	UFlood	Glossy	Splash	SYREN
Multi-packet flooding	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Exploits link correlation	No	No	No	Yes	No	No	No	Yes
Uses network/XOR coding	No	Yes	No	No	Yes	No	Yes	Yes
Control/ACK packet needed	Yes	Yes	Yes	No	Yes	No	Yes	No
Timer based sender selection	No	No	Yes (Feedback needed)	Yes	Yes (Feedback needed)	No	No	Yes
Pipelining in a batch	No	No	No	No	No	No	Yes	Yes
Hardware intervention	No	No	No	No	No	Yes	Yes	No

combination of packets (in a batch). A node is selected as a sender to transmit a linear combination only if it wins a self-organized competition among the neighbors. The competition for transmission is controlled in a distributed fashion by a per-node *forwarding timer*. The timer value reflects the broadcast effectiveness of a node which is locally estimated based on: (i) uncovered neighbors, *i.e.,* the neighbors that have not received the complete data, (ii) link quality, (iii) link correlation, and (iv) received packets. The most effective node will start to re-broadcast early to suppress transmissions from less effective nodes, consequently reducing redundancy.

Our contributions are summarized as follows: (1) We design SYREN, the first (to the best of our knowledge) multi-packet flooding protocol that demonstrates how link correlation and network coding can synergistically disseminate packets with near-perfect reliability and low delay. We propose two novel techniques in SYREN: (i) *Pipelining* the transmission of packets within a batch and across batches to improve reliability and reduce dissemination delay; (ii) Using a *single timer* to manage all transmissions by a node, which makes the protocol simple and scalable. The responsibilities of all nodes are homogeneous, and the protocol has low overhead since each node only needs to maintain per-batch per-neighbor state information rather than per-packet per-neighbor statistics.

(2) We implement SYREN in TinyOS and evaluate its performance through extensive TOSSIM simulations. The results demonstrate that SYREN reduces the average number of transmissions by 30% and dissemination delay by 60%, while achieving the same reliability as state-of-the-art protocols.

(3) We evaluate SYREN in a real-world testbed of 20 TelosB motes and observe performance gains similar to the simulation results. We have released the SYREN implementation for TinyOS on github (*https://github.com/smiftekhar/SYREN*).

II. RELATED WORK

Table I contrasts the data dissemination protocols in a wireless sensor network (WSN) proposed over the past few years. Among these, Deluge [11] is the *de facto* standard incorporated into TinyOS. Deluge divides a data object into multiple batches. Each node then periodically advertises the most recent version of the data object and the set of batches available for transmission, based on which nodes request (and receive) the needed batches.

To address the limitations of Deluge in sparse and lossy networks, coding-based dissemination protocols such as Rateless Deluge [8], SYNAPSE [10], and AdapCode [9] have been proposed. These protocols use network coding or erasure coding to enhance reliability. However, none of them exploits the link correlation present in the network, which can reduce the overhead of control messages.

CF [5] was the first flooding protocol to exploit *link correlation* in WSNs. It uses a *Collective ACK* to infer a successful transmission, and dynamically selects the most effective node as the next sender. However, CF is designed to flood a single packet, and suffers from reliability degradation, redundant transmissions, and increased contention when directly applied to multi-packet flooding (details in Sec. III).

Like SYREN, ECD [12] uses a link quality-based forwarder selection algorithm to mitigate contention over lossy links, but it does not exploit spatial link correlation to estimate the packet reception at neighboring nodes. Instead, it uses explicit ADV (advertisement) and REQ (request) messages which incur high overhead. It does not use network coding and hence suffers from redundant packet transmissions.

The work closest to our work is UFlood [13], a recently proposed data dissemination protocol for mesh networks. Like SYREN, it exploits network coding and link quality-based sender selection. SYREN differs from UFlood in two major ways. First, UFlood uses *explicit* feedback messaging to update the reception status of receivers whereas SYREN uses link correlation-based *implicit* collective ACKs. A typical feedback message in UFlood requires 80 bytes of payload which incurs significant overhead in an energy-constrained sensor network. Second, the timer design of UFlood does not allow pipelined transmissions of packets in a batch, which we will show to be a key advantage of SYREN.

Splash [14] is a data dissemination protocol built on Glossy [15] for flooding a single packet. Splash exploits constructive interference which allows concurrent transmissions of the same packet if the temporal displacement among these transmissions is less than 0.5 μs. However, when correlated loss is prevalent in the network, constructive interference cannot work, rendering Splash unusable [14]. In contrast, SYREN can work under different network conditions, exploiting link correlation information (in the form of implicit ACKs) to estimate packet reception. Further, to support pipelining a batch of packets, Splash requires dynamic channel switching and tight synchronization across nodes, requiring hardware changes, whereas SYREN uses standard networking primitives and supports pipelining through intelligent timer design.

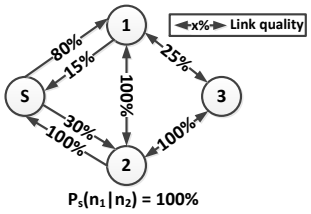
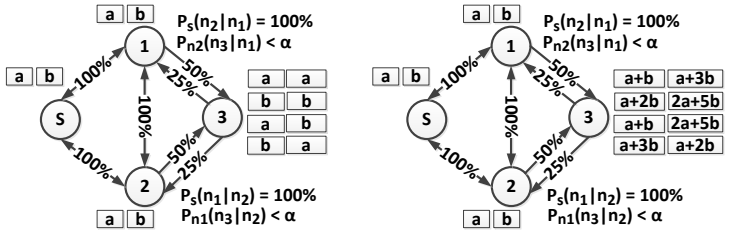


Fig. 1. Collective Flooding in an example sensor network.



(a) Link correlation based (b) Link correlation and network coding based

Fig. 2. Reliability in flooding a batch (size = 2) of packets.

III. Background

A. Link Correlation and Collective ACKs

In traditional explicit ACK-based flooding in WSNs, each receiver sending a separate ACK causes contention and explosion of ACK messages. To mitigate these problems, CF [5] introduces the concept of *collective ACKs* that utilize link correlation. Correlation among links is represented by the *Conditional Packet Reception Probability (CPRP)*. The CPRP – denoted as $P_S(X|Y)$ – is the probability that node X receives a packet from sender node S, given that the same packet is received by node Y. We discuss a practical method to estimate CPRP values in Section V-F as part of SYREN.

To understand how this CPRP information allows collective ACKs, consider the example network shown in Fig. 1. When S broadcasts a packet and node 2 receives it, this transmission serves to node 2 as: (i) a direct ACK that S has the packet already, and (ii) a collective ACK that node 1 has received the packet assuming probability $P_S(1|2) = 100\%$. Again, if node 2 re-broadcasts the packet and S overhears that re-broadcast, S has a direct ACK from node 2, as well as an implicit ACK for the reception at node 1 based on the assumption that $P_S(1|2) = 100\%$. Since S considers all its neighboring nodes as *covered* (*i.e.,* all the neighbors have received the packet), it decides to terminate its transmission. In contrast, in traditional ACK-based solutions, S may redundantly transmit since ACKs from node 1 to S are likely to be lost due the poor link quality (15%) between them.

CF dynamically selects a node as forwarder based on the number of uncovered neighbors and link quality among them. For the example shown in Fig. 1, selecting node 1 as a fixed forwarder creates unnecessary transmissions due to poor link quality with its neighbors. In CF, if nodes 1 and 2 both receive a packet, they compete to be a forwarder and node 2 wins the competition due to better link quality with its neighbors.

CF nodes are initially in a "maintenance state" and periodically exchange *hello* messages with their neighbors. The record of all hello messages received from neighbors is used to discover a node's 1-hop neighbors, and to calculate link quality and CPRP values with these neighbors. Thus, every node u maintains the following information: (i) set of neighbors, $N(u)$, (ii) link quality with node v, $L(u, v)$, $\forall_v \in N(u)$, and (iii) CPRP value, $P_v(k|u)$, $\forall_{v,k} \in N(u)$.

The CPRP information enables collective ACKs in a cumulative manner. Using these collective ACKs, each node maintains the probability (called *coverage probability*) of a neighboring node being covered in a broadcast that is performed or overheard by that node. When a node u receives or overhears a broadcast from node v, it enters the "receiver state" and updates the coverage probability, $CP_u(k), \forall_k \in \{N(u) - v\}$ as follows: $CP_u(k) = 1 - (1 - CP_u(k)).(1 - P_v(k|u))$.

When the coverage probability $CP_u(k)$ reaches a certain threshold (α), u considers k as covered. If u still has uncovered neighbors, it joins the competition to be the next forwarder by setting its back-off timer. The timer is calculated based on the number of uncovered neighbors and link quality with them.

By winning the forwarder competition, node u enters the "sender state" to send out the packet, and updates the coverage probabilities of every uncovered neighbor k as follows: $CP_u(k) = 1 - (1 - CP_u(k)).(1 - L(u, k))$.

B. Link Correlation in Real Networks

Recently, several measurement studies [7], [14] quantified the link correlation present in wireless sensor networks. While link correlation varies across networks and channels, there are link pairs that have highly correlated reception in all cases. Srinivasan et al. [7] presented results from two testbed experiments where about 35-60% link pairs in one testbed exhibited high correlation and about 20% link pairs in another testbed showed similar correlation. The results of our own testbed experiments [6] in different environments show that the extent of link correlation varies in practical networks. Around 50-65% of the link pairs exhibit high ($> 80\%$) correlation, about 25-30% pairs see moderate ($80 - 40\%$) correlation, and the remainder are poorly correlated ($< 40\%$). With such variation in link correlation, it becomes challenging to exploit link correlation-based ACKs for multi-packet dissemination, as discussed next.

IV. Challenges and Opportunities

Directly applying CF to flood multiple packets causes reliability degradation, redundant transmissions, and increased contention. These problems can be addressed by exploiting the synergy between link correlation and network coding.

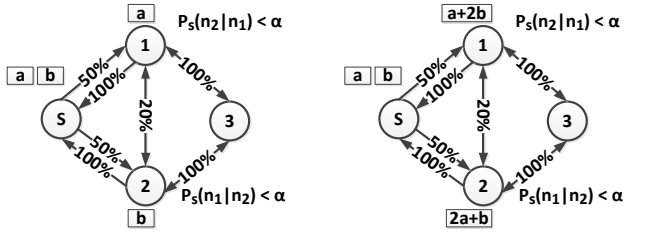
1) Reliability Degradation: Consider the scenario in Fig. 2(a) where the source node S floods a batch (size = 2) of packets $\{a, b\}$ using the CF protocol. Since S has good link quality with its neighbors, it estimates that nodes 1 and 2 have both received the transmitted packets. Upon receiving each packet, nodes 1 and 2 estimate that the packet has been

received by each other since $P_S(1|2) = P_S(2|1) = 100\%$. Then, they compete to be the next forwarder for each of the received packets by setting their respective per-packet back-off timers. Since both nodes 1 and 2 have one uncovered neighbor and the same link quality with that neighbor, they are equally likely to win the competition.

Without loss of generality, we assume that node 1 is selected as forwarder and broadcasts both packets $\{a, b\}$ for which it updates the coverage probability of node 3. However, due to poor link quality between node 1 and 3, node 1's estimated coverage probability for node 3 does not exceed the specified threshold (α). Again, as node 2 overhears the transmissions, it updates the coverage probability of node 3 based on CPRP value, $P_1(3|2)$, which also does not meet the threshold. Thus, nodes 1 and 2 both consider node 3 as uncovered and compete to be the forwarder. This time, we assume that node 2 wins the competition and updates the coverage probability for node 3 after forwarding the packets $\{a, b\}$. Upon overhearing the transmissions from node 2, node 1 also updates the coverage probabilities of node 3. Since the coverage probabilities are updated cumulatively, nodes 1 and 2 find that the coverage probabilities of node 3 for both packets exceed the threshold. However, only two transmissions (from nodes 1 and 2) of two packets $\{a, b\}$ result in four possible combinations of packet receptions at node 3: $\{\langle a, a\rangle, \langle b, b\rangle, \langle a, b\rangle, \langle b, a\rangle\}$. Thus CF achieves only 50% reliability here, since a node should be considered as covered only if it receives the complete batch of packets.

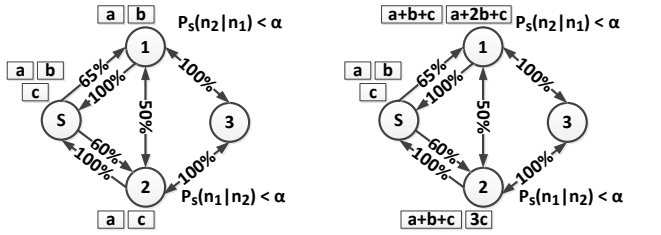
Network coding can improve the reliability to 100% in the above case. As shown in Fig. 2(b), after receiving two independent combinations computed over packets $\{a, b\}$ from S, nodes 1 and 2 are always able to decode the original packets and also to broadcast linearly independent combinations of these packets. Assume that the combinations broadcast by node 1 and node 2 are $\{a + b, a + 2b\}$ and $\{a + 3b, 2a + 5b\}$, respectively. This yields four possible combinations of packets received at node 3: $\{\langle a + b, a + 3b\rangle, \langle a + 2b, 2a + 5b\rangle, \langle a + b, 2a + 5b\rangle, \langle a + 3b, a + 2b\rangle\}$. This achieves 100% reliability since the complete batch of packets can be recovered from any of the combinations.

2) Redundant Transmissions: Consider the example in Fig. 3 where source node S broadcasts a batch of packets, $\{a, b\}$. Consider the case when node 1 only receives packet a, and node 2 only receives packet b as illustrated in Fig. 3(a). Both nodes 1 and 2 estimate each other as uncovered for their received packets, since $P_S(2|1) < \alpha$ and $P_S(1|2) < \alpha$. Now, they compete to be forwarder by starting their respective back-off timers. Assume that node 1 wins, *i.e.*, re-broadcasts first, which acts as a direct ACK to S that node 1 has packet a. S also estimates the probability of node 2's reception of packet a and considers node 2 as uncovered for the packet a, since $P_1(2|S) < \alpha$. When node 2 re-broadcasts its received packet, S decides that node 2 has packet b. This time S considers node 1 as uncovered for the packet b, because $P_2(1|S) < \alpha$. Since each neighbor misses a packet, S is required to re-transmit both $\{a, b\}$. CF may thus cause redundant transmissions, increasing the overall dissemination delay.



(a) Link correlation based flooding (b) Link correlation and network coding based flooding

Fig. 3. Redundant transmissions in flooding a batch (size = 2) of packets.



(a) Link correlation based flooding (b) Link correlation and network coding based flooding

Fig. 4. Contention issues while flooding a batch (size = 3) of packets.

With network coding (depicted in Fig. 3(b)), S transmits two linearly independent combinations, $a+2b$ and $2a+b$ computed over packets $\{a, b\}$. After overhearing re-broadcasting from nodes 1 and 2, S decides that both nodes have received at least one of the two linear combinations. Thus, it suffices for S to successfully transmit one more linear combination of the original packets.

3) Timer Intricacies and Increased Contention: In the presence of links with little correlation, transmission of multiple packets may result in the reception of different subsets of packets at different neighboring nodes. A node that overhears re-broadcasts from different nodes is required to maintain per-packet per-neighbor reception statistics. If the node calculates different back-off timer values for different packets (since the number of uncovered neighbors may be different for each packet), several timers will be required to ensure timely transmission of these packets. Aggregation of different transmissions using a single forwarding timer mitigates this problem but introduces intricacies to the protocol.

The situation is aggravated by contention among neighboring nodes. Consider the example in Fig. 4(a) where S is transmitting a batch (size = 3) of packets, $\{a, b, c\}$. With mediocre link quality from S to nodes 1 and 2, the nodes receive two different sets of packets, $\{a, b\}$ and $\{a, c\}$, respectively. Now, both nodes 1 and 2 become eligible to be forwarders and start back-off timers to transmit the received packets. Assume that node 1 wins the competition and starts transmitting packet a. Even if node 2 overhears that transmission of packet a, it does not stop the timer to transmit packet c. Moreover, node 2

cannot know that node 1 is going to transmit one more packet, b. Thus, if the timer of node 2 fires before node 1 completes the transmission of packet b, one of these two transmissions will likely experience contention delay.

With network coding, each packet is a linear combination of the original packets and a node considers one of its neighbors as covered once it estimates that the neighboring node has received sufficient linear combinations to decode the data. This eliminates the need to track the per-packet reception records of neighbors. Consider Fig. 4(b) where S transmits three independent linear combinations, $\{a+b+c, a+2b+c, 3c\}$. Nodes 1 and 2 receive two different sets of linear combinations, $\{a+b+c, a+2b+c\}$ and $\{a+b+c, 3c\}$, respectively. Now, when node 2 overhears the transmission of a linear combination from node 1, it resets its own back-off timer after updating coverage probabilities for the neighbors. Thus, the transmission of the next linear combination from node 1 is likely to pass through without experiencing collision with transmissions from node 2.

V. SYREN: A Data Dissemination Protocol

SYREN exploits link correlation-based implicit ACKs and network coding to reduce redundant transmissions and improve reliability of data dissemination. A node in SYREN takes one of three roles: a source, a receiver, or a forwarder. The source divides a data object into fixed-size *batches*, generates network-coded packets per batch, and broadcasts the coded packets. When a node overhears a broadcast, it updates its per-neighbor reception records consisting of a coded packet counter and coverage probability. The coded packet counter represents the node's estimate of packet receptions at a particular neighbor, and the coverage probability indicates the estimated reception probability of a coded packet at that neighbor. The node considers a neighbor covered for a batch when it estimates that the neighbor has received sufficient packets to decode the original packets in that batch. If the node has uncovered neighbors, it competes to be the next forwarder by setting a local timer according to its broadcast effectiveness. When the timer fires, the node sends out a coded packet and updates the neighbors' reception status. The node repeats the process until all its neighbors are covered for the current batch.

A. Network Coding

In order to transmit multiple packets reliably, each node in the network needs to track which packets have been received by which neighboring nodes. By incorporating network coding, SYREN avoids maintaining such per-packet per-neighbor reception statistics. Since all coded packets have similar significance, it is sufficient for a node to receive any coded packet instead of receiving a particular packet. In addition, network coding helps SYREN reduce the number of redundant transmissions, especially in networks with low link correlation as discussed in Sec. IV-2.

To control the coding costs, SYREN divides a large data object into batches of M packets, where M does not need to be a fixed value. We refer to these M packets as *original packets* that are mixed using network coding to produce *coded packets* for a batch. Every packet transmitted by a node is a coded packet, $C = \sum_i a_i p_i$, where the a_i's are coefficients chosen randomly by the node, and the p_i's are packets belonging to a batch. The packet header contains four fields: (i) batch ID, g, (ii) number of packets, M, in that batch, (iii) number of linearly independent received packets, M_g, and (iv) a unique identifier representing the coding coefficients. The batch ID helps distinguish coded packets from two different batches.

B. State Management

Each node in our protocol maintains information about link quality and link correlation among its neighboring nodes. A node u maintains a list of its neighbors, $N(u)$ and link quality, $L(u,k)$ for every $k \in N(u)$. It also keeps information about the conditional reception rate of a packet, $P_v(k|u)$, where $v \in N(u)$ is the sender of the packet and $k \in \{N(u)-v\}$. $L(u,k)$ and $P_v(k|u)$ are calculated using periodic hello messages as discussed in Sec. V-F.

The node also maintains information regarding transmission and reception of coded packets in a batch: (1) A *packet buffer* which stores the received coded packets that are linearly independent. (2) A *vector of unique identifiers*, each representing the set of coding coefficients for a received packet. (3) A *vector of coded packet counters*, where counter $C(u,k)(\leq M$, which maintains the node's estimate of the number of packets received at the neighboring node $k \in N(u)$. (4) A *coverage probability vector*, where coverage probability $CP_u(k)$ indicates estimated probability that the neighbor $k \in N(u)$ has received a coded packet. (5) A *forwarding timer* value which indicates the amount of time node u waits before transmitting the next packet.

C. Exploiting Link Quality and Correlation

When a node transmits or receives a coded packet, it exploits link quality and correlation information to estimate the reception of the packet at its neighboring nodes. Fig. 5 depicts the processing steps of data transmission and reception at a node for a particular batch.

Receiver: Whenever a node receives a broadcast coded packet, it checks whether the packet is *innovative* with respect to its packet buffer. Regardless of the innovativeness of the received packet, the receiver updates the coverage probabilities and coded packet counters of its neighbors. Assume u receives a packet from v. Initially, for any node $k \in N(u)$, $CP_u(k) = 0$ and $C(u,k) = 0$. Node u retrieves M_g^v from the received packet, where M_g^v refers to the number of linearly independent packets received so far by v. Then, u sets $C(u,v) = M_g^v$ and updates the coverage probability for each neighbor $k \in \{N(u) - v\}$. $CP_u(k)$ is computed as in CF (Sec. III-A).

When $CP_u(k)$ reaches the threshold α, the node u estimates that k has received one coded packet and resets $CP_u(k)$ to 0 after incrementing the counter $C(u,k)$. When $C(u,k) = M$, node u considers k covered for the current batch and resets $C(u,k)$ to 0. Node u joins the competition to be a forwarder by setting its timer until all of its neighbors are covered.

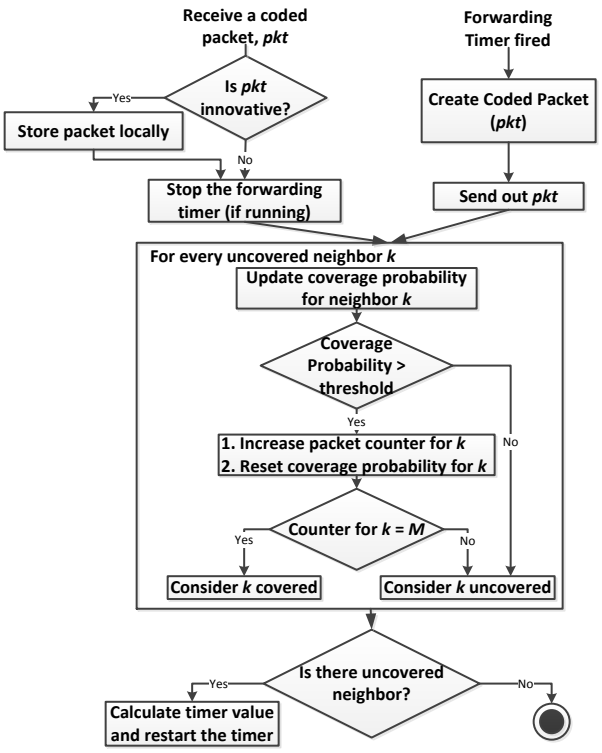


Fig. 5. Control flow of SYREN.

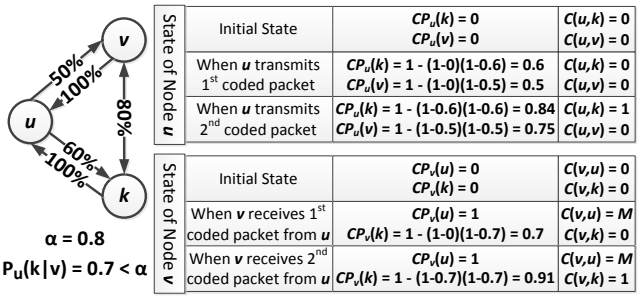


Fig. 6. Cumulative updates of coverage probabilities in SYREN.

Forwarder: The node whose timer fires first wins the competition to be the next forwarder. The forwarder creates a coded packet as a random linear combination of the packets in the packet buffer and then broadcasts it. Assume that node u becomes the forwarder. After broadcasting a coded packet, u updates the coverage probability of each neighbor k, using the same equation as in CF (Sec. III-A).

As in the receiver's case, when $CP_u(k)$ exceeds a pre-specified threshold α, the node u estimates that k has received one coded packet and resets $CP_u(k)$ to 0 after incrementing the coded packet counter for k, $C(u,k)$. If $C(u,k)$ reaches M, node u considers k as covered for the current batch and resets $C(u,k)$ to 0. Node u joins the competition to be a forwarder by setting its timer as long as it has uncovered neighbors.

In CF, coverage probability is updated cumulatively for successive transmissions of a particular original packet whereas in SYREN, coverage probability is updated cumulatively for any coded packet belonging to a particular batch. Considering a small neighborhood consisting of 3 nodes u, v, and k, Fig. 6 shows how cumulative updates of coverage probabilities take place at nodes u and v under SYREN.

D. Forwarder Selection

To reduce redundant transmissions, we aim at selecting the best forwarder in terms of *broadcast effectiveness*. Thus, a node having more uncovered neighbors with good link quality is highly desirable as a forwarder. However, it is also important to select a node that has the ability to generate more innovative packets. A node with more packets is likely to generate more innovative packets and thus offers higher *innovation impact*. Considering both factors, a forwarder in SYREN is selected in a distributed manner via a self-organized competition among neighbors. Since the competition is governed by the locally managed forwarding timers (a.k.a. back-off timers), we compute the forwarding timer value at a node using its *broadcast effectiveness* and *innovation impact*.

Broadcast effectiveness of a node u, $BE(u)$, is a function of its neighborhood size, link quality with neighbors, and the probability of neighboring nodes being uncovered.

$$BE(u) \propto \sum_{\substack{k \in N(u) \\ k \text{ is uncovered}}} L(u,k).(1 - CP_u^*(k))$$

$$\text{where, } CP_u^*(k) = \begin{cases} 0 & \text{if } C(u,k) < M-1 \\ CP_u(k) & \text{if } C(u,k) = M-1 \end{cases}$$

Since a node is considered uncovered until it receives M coded packets, the coverage probability for k, $CP_u^*(k)$ is 0 for $C(u,k) < M-1$. When u estimates that neighbor k has received $M-1$ packets, *i.e.,* $C(u,k) = M-1$, one more successful transmission is sufficient to cover k. In that case, the coverage probability for node k is the probability that k receives one coded packet.

Innovation impact, $II(u)$ is calculated as $II(u) \propto \frac{M_g^u}{M}$, where M_g^u denotes the number of packets available in the packet buffer of u. Then, the timer value is calculated based on $BE(u)$ and $II(u)$ such that a node with higher broadcast effectiveness and innovation impact has a lower timer value. Timer(u) = $\frac{c}{(w_1 BE(u) + w_2 II(u))}$, where c is some constant and w_1, w_2 are weighting factors.

E. Pipelining in a Batch

Our timer design allows every node to be an eligible forwarder as soon as it receives a coded packet. This enables pipelined transmission of packets in a batch, thus reducing the total dissemination delay. The traditional flooding protocols do not support pipelining within a batch, and only allow a node to forward once it receives the complete batch.

Fig. 7 illustrates the pipelining feature of SYREN, where the transmission and reception of packets at different nodes are marked along the timeline. The network considered is shown inside the box in the figure. Assume that node S is the source, which is disseminating a batch of packets. Upon receiving a coded packet from S, node 1 can compete to be the

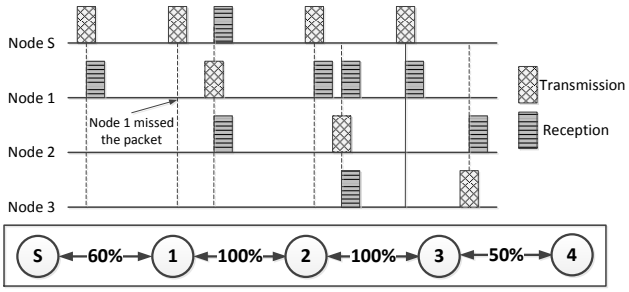


Fig. 7. Illustration of pipelining in a batch.

forwarder and start packet transmission when its forwarding timer fires. Transmission from node 1 helps S compensate any overestimation of the number of packets received at node 1. Additionally, node 2 becomes an eligible forwarder as soon as it receives packets from node 1. When node 3 starts receiving packets from node 2, further transmissions from S and node 3 continue concurrently which speeds up data dissemination. However, our forwarding timer design prioritizes nodes that have more coded packets over other neighbors in order to reduce the redundant transmissions from the nodes having an incomplete batch. In our example, if node 1 has two coded packets and node 2 has one coded packet at a particular moment, node 1 has a higher chance of winning the forwarder competition.

F. Practical Considerations

1) Estimation of Link Quality and Correlation: In order to estimate and exchange the link quality and link correlation information, we implement the algorithm used in CF as a separate library compliant with TinyOS Link Estimation Exchange Protocol (LEEP) [16]. Since LEEP only provides interfaces for link quality estimation, our implementation adds another interface to estimate link correlation. In our implementation, each node sends out hello messages every T seconds and maintains statistics of the hello messages received from its neighboring nodes. The record of hello messages received so far is translated into a bit vector and always piggybacked onto the next hello message. To compute the link quality ($L(u,k)$) and link correlation ($P_v(k|u)$) for a neighbor k, node u compares its own hello message reception records with the reception records received in the last piggybacked hello message from k. The metrics representing $L(u,k)$ and $P_v(k|u)$ are:

$$L(u,k) = \frac{\text{Number of packets } k \text{ received from } u}{\text{Number of packets } u \text{ sent to } k}$$

$$P_v(k|u) = \frac{\text{Number of packets both } u \text{ and } k \text{ received from } v}{\text{Number of packets } u \text{ alone received from } v}$$

where v is present as a sender in the reception records of both u and k.

Due to the changing nature of links, this estimation process must be performed periodically to keep link related information up-to-date. Interestingly, standard routing protocols exploit link quality information to choose next hop [4] and since they are an essential part of a sensor network, data dissemination and routing operations can share such information. Thus, the costs of estimating link quality and link correlation are amortized over the data and control planes.

2) Coding Overhead: To reduce the size of the coded packet header, we do not include the coefficients of linear combinations in the header. Instead, we include a unique identifier representing coding coefficients which are generated based on pre-defined pseudo-random functions. SYREN performs network coding arithmetic in the finite Galois field F_q, where q is the size of the field. We choose $q = 2^8$ which has a very low probability of decoding failure for a reasonable batch size (M) of 24. To speed up the decoding process, we use a pre-computed multiplication/inverse lookup table which takes 256 bytes of memory. We also only allow generation of coded packets out of innovative packets since coding non-innovative packets is not useful as they do not add any information content [17].

VI. Simulations

We implement SYREN in TinyOS 2.x. We first conduct TOSSIM [18] simulations under different network settings. We compare the performance of SYREN with the following flooding protocols: **CF** [5], **Deluge** [11], and **Rateless Deluge** (R_Deluge) [8]. We do not compare with two other state-of-the-art protocols (UFlood [13] and Splash [14]) as their TinyOS-compatible implementations are unavailable.

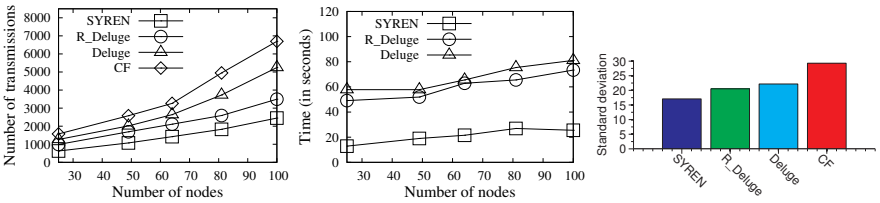
The source sends out a single batch consisting of 24 data packets. Each data packet contains a 22-byte payload. The reliability threshold α is set to 0.85 and 0.95 for SYREN and CF, respectively. The results are averaged over 50 runs.

A. Performance Metrics

We evaluate: (1) *Reliability:* The percentage of nodes in a network that receive the entire data object. (2) *Dissemination Delay:* The time needed to disseminate the (entire data to individual nodes or the time by which every node stops (data transmission, relative to the first data or control packet (transmission made by the source. For a fair comparison, we do not consider the dissemination delay of CF since CF is concerned with flooding a single packet and transmits a packet every 10 seconds. (3) *Number of Transmissions:* The total number of data and control packets transmitted by the time every node stops data transmission. (4) *Load Balance:* The standard deviation of the number of data and control packets transmitted by a node.

B. Grid Network Topology

We vary the grid dimensions from 5×5 to 10×10, keeping the nodes spaced 2.1 meters apart and the source at coordinate (0,0). Since TOSSIM cannot simulate correlated links, we implement support for link correlation. We configure each node with two parameters: *transmission loss* and *reception loss* probabilities. A sender drops packets with transmission loss



(a) Number of transmissions in grids (b) Dissemination delay in grids (c) Load balance in 10 × 10 grid

Fig. 8. Performance comparison of grid network experiments.

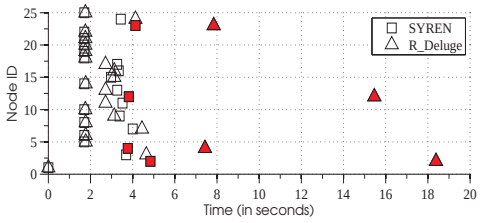


Fig. 9. Forwarder selection in a 25-node network.

probability to ensure correlated reception or loss in its neighborhood. A receiver drops received packets with reception loss probability and adds some variability to the link correlation introduced by the sender. Keeping the average link quality (measured from the TOSSIM topology) within a neighborhood intact, we make 60% of link pairs have correlation >85% and the rest have moderate correlation according to the traces collected from our testbed experiments [6].

1) Reliability: SYREN achieves complete data reception and decoding at each node in the network whereas CF only achieves 60-65% reliability even when its reliability threshold α is set to 0.95. In CF, every packet in a batch is individually disseminated and earlier packets have no effect on the coverage probability for subsequent packets. Based on the α value, if the sender overestimates the coverage probability for a particular packet, the receiving nodes will not receive that packet. Hence CF cannot ensure 99-100% reliability. In SYREN, all packets are coded and overestimation for one packet is corrected by subsequent packet transmissions. In addition, a node can be an eligible forwarder upon receiving a coded packet from a sender and if the node re-broadcasts the packet, the sender can update the coverage counter of the forwarding node based on its reception record (M_g) embedded in the packet. Thus, the coverage probability and counter estimations are more accurate in SYREN, which ensures perfect reliability. Deluge and R_Deluge also provide 100% reliability.

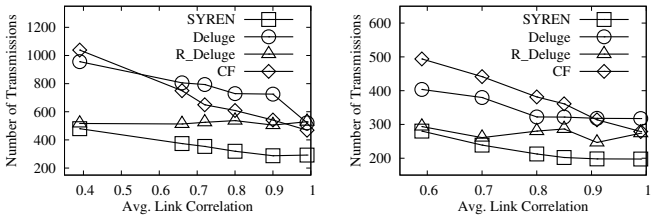
2) Number of Transmissions: As shown in Fig. 8(a), the number of transmissions required in SYREN is 30-36% lower than R_Deluge. Both Deluge and R_Deluge transmit data based on explicit advertisements and requests, which increase the overhead of control packet exchanges. However, unlike Deluge and CF, both SYREN and R_Deluge transmit *network coded* data packets which reduces redundant data packet transmissions by eliminating the need for transmitting different sets of missing packets for different nodes. In addition, SYREN eliminates advertisement/request message overhead since the presence of correlated links in a neighborhood allows a node to estimate reception of packets at other neighbors.

3) Dissemination Delay: Compared to R_Deluge, the dissemination delay in SYREN is more than 60% lower, as shown in Fig. 8(b). In both Deluge and R_Deluge, forwarder nodes require reception of request packets before starting data transmission. As a consequence, losses of such control packets make the propagation of packets slower. By eliminating control packets and by selecting the most effective sender in a neighborhood, SYREN reduces the overall dissemination delay. In SYREN, data dissemination is further expedited since a node is allowed to start forwarding upon receiving a packet. In both Deluge and R_Deluge, nodes wait until the reception of a complete batch of packets to be a forwarder.

4) Load Balance: Fig. 8(c) shows the standard deviation in transmissions per node for a 10×10 network. The standard deviation of SYREN is slightly better than for Deluge and R_Deluge. This is because the presence of asymmetry in the network causes some nodes to transmit a large number of packets in case of Deluge and R_Deluge.

C. Forwarder Selection

Our timer design ensures the most effective forwarder selection in a neighborhood, which reduces the number of transmissions and dissemination delay. Fig. 9 shows the timeline when a node, in a network of 25 randomly deployed nodes, receives a full batch of packets for SYREN and R_Deluge. Before 2 s, the same set of nodes receives a complete batch from the source (node 1) in both SYREN and R_Deluge. Between 2 s and 4 s, 10 nodes are covered in SYREN whereas only 6 nodes are covered in R_Deluge. Further, SYREN covers 4 nodes (red boxes in the figure) much earlier than R_Deluge (red triangles in the figure). This is because SYREN selects nodes with more neighbors with good link quality, thus covering more neighboring nodes with a lower number of transmissions and reduced delay. In case of R_Deluge, when any other node is selected as sender, several rounds of DATA and REQ transmissions among the neighboring nodes are required.



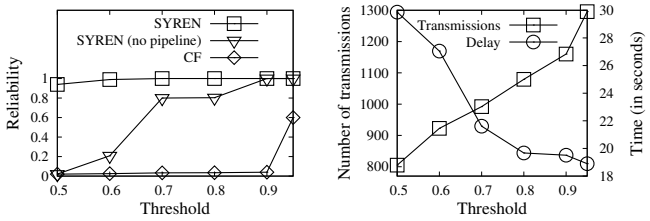
(a) PRR = 0.4 (b) PRR = 0.6

Fig. 10. Evaluation with varying PRR and link correlation.

D. Varying Link Quality and Correlation

To understand the synergy between link correlation and network coding in SYREN, we vary the packet reception ratio (PRR) and the average link correlation in a controlled manner. We perform this experiment in a 4×4 grid and vary average link correlation from 0.35 to 1 for PRR values of 0.4 and 0.6.

As shown in Fig. 10(a), for a low PRR network with low link correlation, both SYREN and R_Deluge perform better than Deluge and CF. Due to low link correlation, different nodes in a neighborhood lose different sets of packets, causing redundant transmissions for Deluge and CF. By using network coding, both R_Deluge and SYREN reduce these redundant transmissions. As link correlation increases, SYREN performs better than R_Deluge and CF performs better than Deluge. Since SYREN utilizes knowledge of link correlations, a node can estimate packet receptions at its neighbors with higher accuracy and avoid redundant transmissions. It is interesting to note that when link correlation is high, R_Deluge matches Deluge in terms of packet transmissions. The reason is that network coding offers nothing extra when packet losses are correlated across different links in the neighborhood. By exploiting link correlation information, SYREN outperforms both Deluge and R_Deluge in this case. Though CF improves in terms of number of transmissions after a certain correlation value, it never achieves 100% reliability. The number of transmissions for CF, Deluge, R_Deluge, and SYREN exhibit similar trends in networks with PRR 0.6 as shown in the Fig. 10(b).



(a) Reliability (b) Transmissions vs. delay in SYREN

Fig. 11. Varying threshold in 7 × 7 grid.

E. Varying Reliability Threshold

To observe the impact of pipelining, we devise a non-pipelined version of SYREN where a node becomes an eligible sender only after receiving a complete batch. In a 7×7 grid network, we vary the reliability threshold α from 0.55 to 0.95 and compare SYREN, SYREN without pipelining, and CF. As shown in Fig. 11(a), SYREN achieves 100% reliability beyond α = 0.6 whereas SYREN without pipelining achieves 100% reliability at α = 0.9. CF ensures 60% reliability even with α set to 0.95. For lower α, sender nodes frequently overestimate the reception of packets at their neighbors. Pipelined transmissions from a node with an incomplete batch of packets in SYREN allow senders to correct overestimation. In SYREN without pipelining and in CF, it is impossible to correct such overestimation. Though non-pipelined SYREN improves reliability over CF, it still needs to set α to a higher value to ensure 100% reliability, which increases the number of transmissions. Interestingly, with low α, SYREN's perfect reliability comes at the expense of increased delay. Nodes having an incomplete batch of packets have lower transmission priority and so larger forwarding timer value. Thus sender nodes have to wait longer to correct their overestimation which increases overall dissemination delay. Fig. 11(b) shows that the number of transmissions increases and dissemination delay decreases as α increases for SYREN.

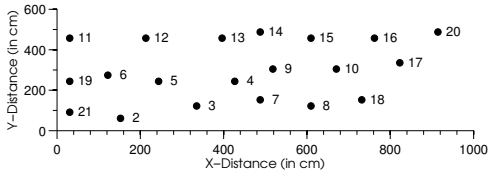
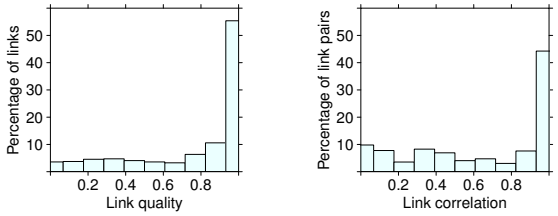


Fig. 12. Placement of nodes in indoor multi-hop network.



(a) Histogram of link quality (b) Histogram of link correlation

Fig. 13. Distribution of link quality and correlation present in the network.

VII. Prototype and Evaluation

We port the implementation of SYREN to the TelosB platform. Our motes have an 8 MHz TI MSP430 micro controller, 2.4 GHz radio, 10 kB RAM, and 1 MB flash for data logging. Implementations of **Deluge**, **Rateless Deluge**, and **CF** are also ported to this platform. We consider the same performance metrics as in Sec. VI-A.

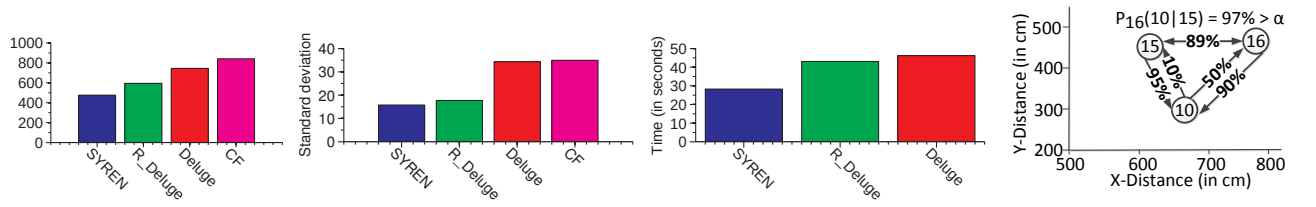
We place battery-powered TelosB motes in an indoor environment and control transmission power to ensure multi-hop communication. The source node sends out a single batch of 24 data packets, each packet with a 22-byte payload. The thresholds in SYREN and CF are set to 0.85 and 0.9, respectively. The results are averaged over 10 runs.

A. Multi-hop Indoor Experiments

We construct a 9 m × 5 m topology consisting of 20 TelosB sensors deployed randomly in a Purdue University classroom. In order to ensure multi-hop communication, we used the lowest power level 1. Fig. 12 shows the coordinates of the nodes in the testbed, where nodes are labeled 2 to 21. Node 2 is the source. We observed varying link quality and correlations in the network as shown in Fig. 13.

Our experimental results show that CF cannot ensure 100% reliability. The reason is that the sender sometimes overestimates packet reception at neighbors. Though SYREN improves reliability with network coding, there is a low probability that its reliability is affected by its threshold-based estimation. In this case, pipelined transmissions from nodes with incomplete batches of packets help their neighbors correct any overestimation. Thus SYREN achieves 100% reliability for the given threshold. Deluge and R_Deluge achieve 100% reliability as well.

Fig. 14(a) shows that SYREN reduces the number of transmissions with respect to R_Deluge, Deluge, and CF by 25%,



(a) Number of transmissions (b) Load balancing (c) Dissemination delay (d) With asymmetric and correlated links

Fig. 14. Performance evaluation in an indoor multi-hop network.

36% and 43%, respectively. Due to varying link quality, CF and Deluge require redundant transmissions. Network coding helps SYREN and R_Deluge reduce such transmissions. In addition, SYREN reduces transmissions by selecting the most effective sender. SYREN also avoids control/ACK packet overhead by exploiting link correlation-based implicit ACKs.

Fig. 14(b) shows that SYREN and R_Deluge have lower standard deviation than CF and Deluge. For Deluge, asymmetric bottleneck links (as discussed below) cause some extra packet transmissions, increasing the standard deviation. Some CF nodes also require a large number of transmissions due to their poor link quality with neighboring nodes.

As shown in Fig. 14(c), SYREN reduces dissemination delay by 35% and 40% compared to R_Deluge and Deluge, respectively. Though R_Deluge requires fewer transmissions than Deluge, it still incurs a significant dissemination delay due to explicit REQ-ADV message exchanges and checking independence of coded packets. SYREN achieves a low dissemination delay due to the selection of the most effective sender and the pipelined transmission of packets. Further, the time required to check independence of coded packets is amortized over pipelined transmissions.

B. Effect of Asymmetric and Correlated Links

Fig. 14(d) shows the link quality among a partial set of nodes in the testbed network. It is seen that node 10 has asymmetric links with nodes 15 and 16. In both Deluge and R_Deluge, request messages from node 10 towards nodes 15 and 16 are lost, which increases overall dissemination delay. However, in SYREN, node 15 estimates the reception of packets at node 10, based on its knowledge of the quality of the link between 10 and itself. Further, node 16's transmission helps node 15 estimate packet reception at node 10 since $P_{16}(10|15) = 97\% > \alpha$. Without exploiting this link correlation information, any forwarder selection algorithm (*e.g.*, MNP, ECD) that uses explicit control messages to select the most effective sender will suffer from repetitive control messages and increased dissemination delay.

VIII. Conclusions and Future Work

In this paper, we present the design and implementation of a multi-packet flooding protocol, SYREN, that exploits the synergy among link correlation and network coding to provide an efficient, reliable data dissemination service with low complexity. We implement SYREN in TinyOS, and conduct simulations and testbed experiments to compare SYREN with other link correlation-based and network coding-based flooding protocols. Results show that SYREN is scalable with low overhead, low dissemination delay, high reliability. We plan to extend SYREN to flood multiple batches of packets.

Acknowledgments. This work was supported in part by NSF grants CNS-0905331 and CNS-0964294.

References

[1] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in *Proc. of ASPLOS X*, 2002.

[2] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. of USENIX Symposium on NSDI*, 2004.

[3] R. Fonseca, S. Ratnasamy, D. Culler, S. Shenker, and I. Stoica, "Beacon vector routing: Scalable point-to-point in wireless sensornets," in *Proc. of USENIX Symposium on NSDI*, 2005.

[4] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. of ACM Sensys*, 2009.

[5] T. Zhu, Z. Zhong, T. He, and Z. li Zhang, "Exploring link correlation for efficient flooding in wireless sensor networks," in *Proc. of USENIX Symposium on NSDI*, 2010.

[6] S. M. I. Alam, S. Sultana, Y. C. Hu, and S. Fahmy, "Link correlation and network coding in broadcast protocols for wireless sensor networks," in *Proc. of SECON (poster)*, 2012.

[7] K. Srinivasan, M. Jain, J. I. Choi, T. Azim, E. S. Kim, P. Levis, and B. Krishnamachari, "The k factor: inferring protocol performance using inter-link reception correlation," in *Proc. of ACM MOBICOM*, 2010.

[8] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless deluge: Over-the-air programming of wireless sensor networks using random linear codes," in *Proc. of IPSN*, 2008.

[9] I.-H. Hou, Y.-E. Tsai, T. F. Abdelzaher, and I. Gupta, "Adapcode: Adaptive network coding for code updates," in *Proc. of IEEE INFOCOM*, 2008.

[10] L. M. Ni, Y. Liu, and Y. Zhu, "Synapse++: Code dissemination in wireless sensor networks using fountain codes," *IEEE Trans on Mobile Computing*, vol. 9, pp. 1749–1765, 2010.

[11] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. of ACM Sensys*, 2004.

[12] W. Dong, Y. Liu, C. Wang, X. Liu, C. Chen, and J. Bu, "Link quality aware code dissemination in wireless sensor networks," in *Proc. of IEEE ICNP*, 2011.

[13] J. Subramanian, R. Morris, and H. Balakrishnan, "Uflood: High-throughput flooding over wireless mesh networks," in *Proc. of IEEE INFOCOM*, 2012.

[14] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast data dissemination with constructive interference in wireless sensor networks," in *Proc. of USENIX Symposium on NSDI*, 2013.

[15] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proc. of IPSN*, 2011.

[16] O. Gnawali, "The link estimation exchange protocol (LEEP)," 2007, online at http://www.tinyos.net/tinyos-2.x/doc/html/tep124.html.

[17] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *Proc. of ACM SIGCOMM*, 2007.

[18] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. of ACM Sensys*, 2003.