Reliable Communication and Latency Bound Generation in Wireless Cyber-Physical Systems

Article *in* ACM Transactions on Cyber-Physical Systems · November 2019

DOI: 10.1145/3354917

CITATIONS
0

READS
53

8 authors, including:

Sirajum Munir
University of Virginia
30 PUBLICATIONS **495** CITATIONS
SEE PROFILE

Hao-Tsung Yang
Stony Brook University
8 PUBLICATIONS **22** CITATIONS
SEE PROFILE

S. M. Shahriar Nirjon
University of Virginia
24 PUBLICATIONS **416** CITATIONS
SEE PROFILE

Enamul Hoque
Samsung
25 PUBLICATIONS **483** CITATIONS
SEE PROFILE

Reliable Stream Scheduling with Minimum Latency for Wireless Sensor Networks

Hao-Tsung Yang*, Kin Sum Liu*, Jie Gao*, Shan Lin†, Sirajum Munir§, Kamin Whitehouse‡, and John Stankovic‡
*Department of Computer Science, Stony Brook University
†Department of Electrical and Computer Engineering, Stony Brook University
‡Computer Science Department, University of Virginia
§Bosch Research and Technology Center, Pittsburgh, PA

***Abstract*—As sensor networks are increasingly deployed for critical applications, reliability and latency guarantee become more important than ever to meet industrial requirements. In this paper, we investigated the impact of link burstiness on stream scheduling using a data trace of 3,600,000 packets collected from an indoor testbed. We demonstrate that a good tradeoff between reliability and latency can be achieved by allocating certain time slots on each link for stream transmissions based on its burst length and frequency distributions. With this observation, we design transmission scheduling and routing algorithms for data streams to meet a specified reliability requirement while minimizing end-to-end latency. For the multi-stream scheduling problem, we prove its NP-hardness and design an algorithm that achieves the reliability guarantee and an $O(\log n)$ approximation of minimizing the maximum end-to-end latency for any stream. Trace-driven simulations show that our solution meets specified end-to-end reliability requirements with latency up to 9.18 times less than existing solutions.**

I. Introduction

As sensing and wireless technologies become mature, wireless sensor networks are increasingly deployed for applications including industrial plant monitoring [22], building occupancy sensing [15], and manufacturing pipeline monitoring [19]. These applications usually require sensing data to be collected periodically for real-time monitoring and diagnosis purposes. Therefore, one or more real-time data streams need to be transferred among the sensors and controllers via multi-hop with high reliability to detect events of interest. However, the problem of achieving specified reliability requirement in multi-hop low power sensor networks is very challenging due to the unpredictable packet loss caused by various interferences and many other factors.

There has been extensive research on link quality modeling and characterization. Srinivasan et al. [23] developed a link model for measuring the bursty packet loss. Munir et al. [17] focused on capturing the worst-case packet loss on each link. These valuable results show that in a closed indoor environment with limited external interferences, certain links have more predictable reliability than others. Based on this observation, a few wireless sensor network designs have successfully achieved 100% reliability in deployed systems [7], [20]. On the other hand, such high reliability is usually obtained at the cost of high latency and energy consumption. There are many applications require high reliability but not necessarily 100% while the low end-to-end delay is also preferred [12], [15], [19]. For instance, in a home health monitoring system, 95% reliability is sufficient to monitor the health condition of the user and a few packet loss does not affect the monitoring quality significantly. Therefore, it is desirable to find a good tradeoff between reliability and latency in stream data transmission.

To address this problem, we first conduct an analysis of a data trace collected from a real indoor sensor network testbed to learn the patterns of packet loss on different links. We show that most bursty links with low burst frequencies have consistent loss patterns over a long time period. Therefore, by allocating certain time slots on such links, i.e. restricting the maximum number of retransmissions, a specified level of link reliability and corresponding latency bound can be achieved. Our solution is based on the following insight: most of the links do not show their worst case burst behaviors frequently. Hence, instead of allocating time slots based on maximum burst length, if we allocate slots based on frequency distribution of bursts subject to link interferences and utilize load-balancing algorithm, the solution not only offers expected reliability but reduces end-to-end latency significantly.

Thus, we design a cross-layer solution to schedule streams with different reliability requirements. At link layer, a link reliability table stores the numbers of time slots required to achieve different levels of packet delivery ratio for this link. At network layer, our algorithm decides the number of allocated slots for each stream on each link to meet the specified reliability requirement and minimize the maximum end-to-end latency bound of all streams.When scheduling single stream, our algorithm meets the reliability requirement with the minimum latency bound. For multiple data streams in the network, the interferences among streams need to be considered. We prove the hardness of this problem and design an approximation algorithm with reliability guarantee and bounding the maximum latency. Differ from previous researches which consider 100% reliability only [17], [20], our algorithms prove rigorous guarantees on the maximum delay with any given reliability requirements.

Extensive trace-driven simulations with 300 streams show that our solution successfully meets all the specified reliability requirements: given the stream reliability requirements 95%, 97%, 99%, the average end-to-end delivery rates achieved are 95.7%, 97.76%, and 99.16%.

The contributions of this paper are summarized as follows:

- We reveal the impacts of link burstiness on the reliability and latency of stream transmissions with data-driven analysis.

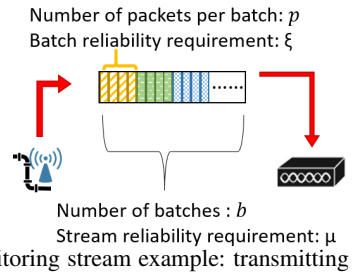


Fig. 1: A monitoring stream example: transmitting multiple samples to the sink node

- We provide a cross-layer solution to meet the specified reliability requirement and minimize end-to-end latency bound.We prove the stream scheduling problem is NP-hard and design an algorithm that achieves the specified reliability guarantee and an $O(\log n)$ approximation of minimizing the maximum end-to-end latency bound of all streams. Note that the end-to-end latency can not be infinity in such a case as the number of retransmissions are restricted to a finite limit.
- Our solution successfully meets the end-to-end stream reliability requirements while reducing end-to-end latency by up to 9.18 times comparing to existing algorithms in extensive trace-driven evaluations.

The rest of the paper is organized as follows, Section II states the definition of the problem. In Section III, we discuss how to build tables for trading reliability with delay at link level and analyzes the steadiness of the tables. The design of our algorithms is described in Section IV. We conduct simulations for the proposed algorithms and report the results in Section V. Finally, SectionVII is the conclusion.

II. Definition

In many monitoring applications, periodic sensing data streams are transmitted from the sensors to the sink via a multi-hop network. A typical data stream instance (e.g. vibration, acoustic, ECG) consists of a number of continuous or discrete samples, which are transmitted with a batch, i.e. a set of related packets. This scenario is summarized as follows:

1) Each stream generates a batch of multiple packets periodically.
2) For each stream, there is a stream reliability requirement: a specified percentage of batches is required to be delivered for the continuous monitoring purpose.
3) For each batch of packets, there is a batch reliability requirement: a minimal percentage of packets is required to be delivered reliably for each batch, to achieve the quality of a sample.

This scenario is shown at Figure 1. Generally, each data stream is represented by a sequence of batches of packets from source s to sink t. The batch rate at the source is denoted by b, which represents the number of batches per hour. The stream reliability requirement is μ. Each batch has a set of p packets and the batch reliability requirement is ξ. We formally describe this stream scheduling problem as a 6-tuple (s, t, b, μ, p, ξ).

The formal description of our problem is as the following; given a set of streams needs to be scheduled, each stream i is represented by 6-tuple $(s_i, t_i, b_i, \mu_i, p_i, \xi_i)$, how to find routes and transmission schedules to meet the stream and batch reliability requirement μ_i, ξ_i of all streams while minimizing maximum latency? In this paper, we use *time slots* as the measurement of latency. A time slot is a scheduling time unit such that any node can transmit one packet to another node and receive an acknowledgment if the packet is delivered.

III. Link Reliability Table Construction

A. Packet trace collection

We conducted a 21-day experiment on a testbed with 48 Tmote nodes running TinyOS. For each pair of two nodes, a total of 3,600,000 packets is transmitted to evaluate packet delivery traces. The transmission rate is approximately 200 packets per second for all links. The transmissions of different links are scheduled at different time slots to avoid a collision. The receiving status of each packet at every node is recorded as a sequence of 0 and 1. More specifically, a packet delivery trace D contains a 1 at position d means the d-th packet is successfully received, which we denoted by $D[d] = 1$. So for each link, we get a trace D with length 3,600,000.

B. Transmission for a batch

Consider a case of sending a batch of packets with the retransmissions at the link layer, the transmitter keeps retransmitting the packets to the receiver until it receives the full batch. Each transmission takes exact one time slot. In our analysis, we set the number of allocated time slots l per batch of each link. The transmitter will immediately stop retransmitting the rest of packets in the batch if it already uses l time slots. We change the number of the allocated time slots l on different links to show how it affects reliability requirements ξ, μ and the latency.

As the previous definition, a batch is treated as successfully transmitted if and only if there are more than $p * \xi$ received packets, where p is the number of packets per batch and ξ is the batch reliability requirement. Therefore, given a trace D, we can calculate the delivery rate, which is the rate of successfully transmitted batches, if we assume the starting time of the transmission is uniformly distributed in D.

In detail, denote $|\cdot|$ is the number of received packets in a length of the trace. Given parameters p, l, for an arbitrary starting time slot d at trace D, the number of received packets during the time slot d to $d + l - 1$ is

$$\min(|D[d, d + l - 1]|, p). \tag{1}$$

Since we assume the starting time is uniformly distributed on the trace, we calculate the empirical distribution function F_l respected to the number of received packets among all possible starting time d. Therefore, the delivery rate of a batch is actually the value of $1 - F_l(x \leq p * \xi)$.

Figure 2 is such an example with batch size $p = 5$ and batch requirement $\xi = 80\%$. It shows the cumulative distribution of received packets in a batch under different allocated time slots l. For simplicity, we only show allocated time slots l being 5, 15, 100, or 1000.

Now, by drawing a vertical line as the threshold of specific batch requirement ξ with size p, each distribution gives the percentage of batches that satisfies ξ. For example, allocating 5 time slots for each batch gives you 91.5% delivery rate for $\xi = 80\%$ $(1 - F_5(x \leq 5 * 80\%) = 91.5\%)$. Therefore, we can record this delivery rate under different allocated time slots l.

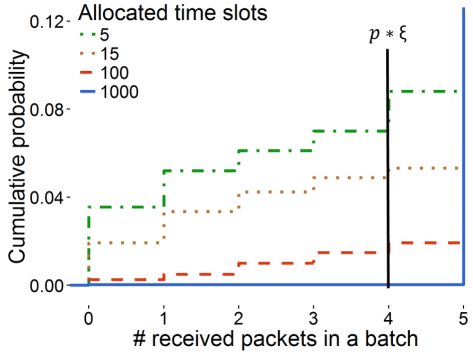


Fig. 2: Empirical distribution for different values of l (allocated time slots)

$\xi = 80\%, p = 5$							
Delivery rate	91.5%	...	95.5%	...	97.5%	...	100%
Allocated time slot	5	...	15	...	100	...	1000

TABLE I: Link reliability table record the batch delivery rate with corresponding allocated time slots at trace D

Notice that these tables might cannot reflect link qualities for the long term in reality. However, we can also easily update tables in real time system. By taking the transmission results as an additional training set, we can use a weight parameter to combine the old and the new training set for updating tables.

C. Link reliability table

Based on our empirical analysis, we design the link reliability table as a 2-column data frame recording (batch) delivery rates and allocated time slots on a link. It treats delivery rate as the key such that our algorithm can query the minimum allocated time slots for the link to meet specific delivery rate. The allocated time slots for different delivery rates are calculated based on the trace analysis. The example is shown at Table I.

We found that the tables of different links are highly correlated with the link burstiness, so we classify links according to different patterns of burstiness and discuss table steadiness among all classes as follows.

With many complex factors affecting bursty packet loss distribution, each link might show a totally different characteristic of the tradeoff between delivery rate and latency. If the link usually suffers from long consecutive packet losses, then increasing allocated time slots does not improve delivery rate significantly. On the other hand, if the link suffers from short bursty loss, increasing the allocated time slots might enhance reliability. Here a bursty packet loss is represented by a substring with consecutive 0's in the delivery trace.

Based on this intuition, we look at the bursty packet loss frequency and length on different links. We define average burst length as the average length of burstiness and burst frequency as the average number of burstiness within one hour of the delivery trace. Next, we classify links into 4 classes by their burst length and burst frequency. We pick links with 70% PRR as available links and use the median values as the threshold to divide these classes, i.e. *burst frequency per hour* ≥ 1157 is high-frequency links and *average burst length* ≥ 2.57 as long burst length links. Table II is the distribution of these classes.

HFLB : High-frequency with long-burst length.
HFSB : High-frequency with short-burst length.
LFLB : Low-frequency with long-burst length.
LFSB : Low-frequency with short-burst length.

HFLB	HFSB	LFLB	LFSB
29.7%	21.3%	21.3%	29.7%

TABLE II: Link class distribution on our testbed

If burst distribution is stable, the link reliability tables should be stable too, given that the measurement is taken for a sufficient period to capture the distribution of burst. To check table steadiness of each class, we calculate the standard deviation of delivery rate over different measurement days with the numbers of allocated time slots. The standard deviation is calculated from a set of reliability tables with same measuring length but different measuring days. For consistency, we use 3 days of a delivery trace as measuring length (our analysis shows the tables change little with more than 3 days of a delivery trace.) Therefore, for each link, there are 18 reliability tables and we calculate the standard deviation among them.

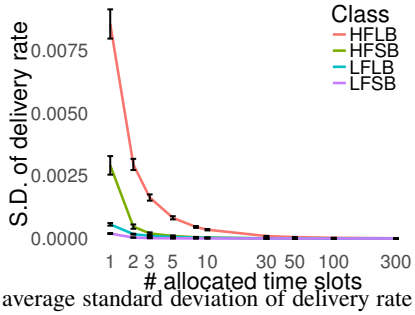


Fig. 3: The average standard deviation of delivery rate over different measuring days among all classes

Figure 3 shows the standard deviation given different number of allocated time slots in each class. The x-axis is the number of allocated time slots and y-axis is the average standard deviation of each class. Error bars showed the standard errors among links at the same class. Generally, the delivery rate changes little with a high number of allocated time slots(> 30). However, for HFLB and HFSB classes, the delivery rate is not stable with a low number of allocated time slots. We believe it is because links in these classes suffer burstiness frequently, which affect burst distribution hugely and make link reliability diverse among different measuring days. In sum, the standard deviation of the delivery rate among all classes is lower than 1%, which we believe is stable enough for most applications. For other applications with hard real-time guarantees, we suggest using LFLB and LFSB links since these links show very low standard deviation($< 0.1\%$), which imply high consistency of the tradeoff between reliability and delay.

In long-term, the classes of links need to be updated according to table changes. The update frequency is based on the environment of sensor network. According to our analysis, it is suitable to update classes once per month in our testbed. Such overhead is manageable for real world application.

IV. Scheduling Algorithm Design

In this section, we will describe how to use the link reliability tables to achieve *end-to-end* stream reliability requirements

with minimum delay. We start from three different application scenarios and we will discuss the algorithms for each scenario separately.

Notice that our scheduling algorithms are focused on choosing routes for each stream rather than transmission time slots. Therefore, for the flexibility, we don't consider internal interference in second and third scenarios because it is also relevant to transmission schedule and the protocol of sensor network. However, we also provide a scheduling algorithm which avoids internal interference in section IV-D3.

A. Three streaming scenarios

To quantify the end-to-end latency, recall that one packet transmission will take a time slot. The exact length of a time slot depends on the implementation of the protocol and here we count the number of slots taken from source to destination. Depending on applications, we may have three streaming scenarios, single stream with a single batch, single stream with multiple batches and multiple streams with multiple batches.

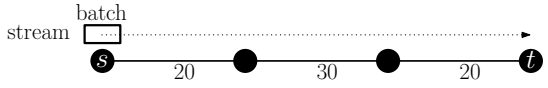


Fig. 4: One stream transmitting one batch with latency bound = 20+30+20 = 70. In the worst case, the only batch may require 70 packet transmissions along the route to reach node t.

Single Stream with a Single Batch. In this scenario, we assume that there is only a single data stream in the network and the stream has only a single batch of packets. Figure 4 illustrates one example. Suppose in this example we need to allocate 20, 30, and 20 time slots respectively along the route to achieving the reliability requirement. Therefore the worst case end-to-end latency for the single batch to successfully arrive from s to t with the given reliability requirement is 70, by adding up the allocated slots along the route.

In general, each link may have multiple entries in the link reliability table. Thus we want to find among all routes that satisfy the given reliability requirement the one with minimum total latency. This problem has similarity to finding the shortest path when the weight of a link is taken as the number of allocated time slots of the link. But the challenge is to consider the reliability of the route as well. Later we show how to use a dynamic programming algorithm to solve it optimally.

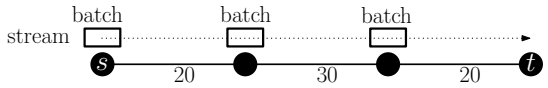


Fig. 5: One stream transmitting multiple batches with latency bound = 30. In the worst case, the node t may need to wait for 30 packet transmissions to receive a new batch.

Single Stream with Multiple Batches. In this case, we have a single data stream with multiple (possibly infinite) batches of packets. For the example in Figure 5, the node s generates and sends out batches continuously. Now we have a pipeline of packets traveling along the route. While the first batch is being transmitted on the second link the second batch can be transmitted along the first link (here we assume it is the full-duplex sensor network for simplicity). It will still take 70 time slots for the first batch to arrive at the destination. But after that, every 30 slots the destination receives another batch. In this case, the natural latency measurement is the throughput of the flow – the number of time slots for receiving an additional batch, which is the bottleneck latency along the route. Again we find the route with maximum throughput (minimum bottleneck latency) among all routes that meet the reliability requirement.

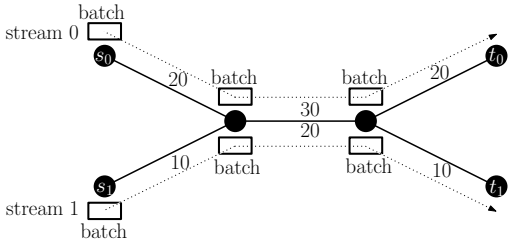


Fig. 6: Two streams transmitting multiple batches with latency bound = 30+20 = 50. In the worst case, the node t_0 (t_1) may need to wait for 50 packet transmissions to receive a new batch from source s_0 (s_1) since the two streams take turn to use the shared link.

Multiple Streams with Multiple Batches. In Figure 6, one of the links is shared between two streams, requesting 30 and 20 slots respectively for each batch. To meet the reliability guarantee for the shared link we run round-robin on all the data streams through it. Therefore, the latency for a batch to get through the shared link in this example is $50 = 30 + 20$ as the link is shared by two streams. For each data stream, we measure its throughput as the bottleneck latency among the links on its route. We look for routing algorithms such that the maximum throughput for all streams is minimized.

B. Single Stream with a Single Batch

As explained earlier, finding a route to minimize latency for a single stream with a single batch is similar to finding the shortest path. We adapt and modify the Floyd-Warshall algorithm[6] which uses a dynamic programming approach. The main challenge is to handle the reliability requirement. In particular, each link has multiple entries in its reliability table and we need to select and combine the link reliability to meet the end-to-end reliability requirement. This difference is reflected in how we combine the subproblems in the DP formulation. For a high-level idea, we are storing and updating a table for each pair of nodes x, y to record the tradeoff between reliability and allocated time slot for source s and destination t.

The details are provided in the Algorithm 1. For every pair of source x and destination y, we maintain a table $T_{xy}(S)$ which records the reliability tradeoff of a path from x to y using intermediate nodes only from the set S. Initially $T_{xy}(\emptyset)$ is obtained directly from the data traces [Line 4]. The final solution can be found in table $T_{st}(V)$, where V is the set of all vertices. The program builds upon subproblems: each time we introduce a new node into the set S at a time [Line 6], we incrementally update the tables [Line 9] until $S = V$ [Line 12]. The value l of the entry $(\mu, l) \in T_{st}(V)$ is the minimum end-to-end latency to achieve reliability requirement for the

Algorithm 1 Find-minimum-time-slot-path

procedure FIND-FASTEST-PATH(s, t, p, ξ)
  $S \leftarrow \emptyset$
  for any two vertices x, y **do**
    initialize($T_{xy}(S, p, \xi)$)
  end for
  repeat
    pick a vertex v in V/S
    for any two vertices x, y **do**
      update($T_{xy}(S \cup \{v\}, p, \xi)$)
    end for
    $S \leftarrow S \cup \{v\}$
  until $S = V$ ▷ If all vertices added, done
  return $T_{st}(V, p, \xi)$
end procedure

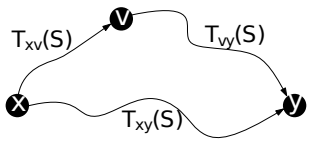


Fig. 7: Two operations to update node-to-node tables. After concatenating $T_{xy}(S)$ and $T_{vy}(S)$, we can compare it with $T_{xy}(S)$.

stream. Note that the relative frequency of transmission is not applicable to the single stream case so the parameter b_i is not used in the algorithm.

Since we are only working on the tables parameterized by p and ξ. The notation for $T_{xy}(S, p, \xi)$ is abbreviated to $T_{xy}(S)$ whenever the context is clear. The remaining question is to update the table. For doing so, we need to introduce two operations, compare and concatenate.

Concatenate. The *Concatenate* operation is to combine the tradeoff information from two tables $T_{xv}(S)$ and $T_{vy}(S)$ for some set S ($v \notin S$) (i.e., the end node of the first table is the start node of the other table). We can concatenate the tables for xv and vy to form a new table denoted as T^v_{xy} which considers paths going through v. Taking Figure 7 as an example.

For each entry $(g_j, l_j) \in T^v_{xy}$, the allocated time slot l_j is equal to the minimum sum of allocated time slots from all the combinations between (g_p, l_p) and (g_q, l_q) satisfying g_j. That is,

$$(g_j, l_j) \leftarrow (g_j, \min_{g_p \cdot g_q \geq g_j} (l_p + l_q)), \tag{2}$$

where $(g_p, l_p) \in T_{xv}(S)$, $(g_q, l_q) \in T_{vy}(S)$. Naively, to calculate a single new entry (g_j, l_j), it takes $O(k^2)$ where k is the number of entries in each table, to evaluate all the combinations of g_p and g_q. However, notice that l_p is monotonically increasing when g_p is increasing, so as l_q. Therefore, if we fix the entry's index at $\dot{p}$ in $T_{xv}(S)$ when calculating the entry j in $T_{xy}(S)$, the minimum value of $(l_p + l_q)$ must be at the entry with the minimum index q such that $g_{\dot{p}} \cdot g_q \geq g_j$. In addition, for each entry $\dot{p}$, this corresponding q is pre-computable and we can store them in a $k \times k$ matrix. Hence, to calculate all the combinations of g_p and g_q, we can always look up corresponding g_q from the matrix. The time complexity of computing an entry (g_j, l_j) reduces to $O(k)$ in this way. With pre-computation, deriving all entries in T^v_{xy} takes only $O(k^2)$.

Compare. The *Compare* operation is used to compare the tradeoff of two different paths. After we derive T^v_{xy} by the *concatenate* operation, we need to revise the current estimate in $T_{xy}(S)$ with this new information to form the new table $T_{xy}(S \cup \{v\})$. We perform the following procedure. For each entry $(g_j, l_j) \in T_{xy}(S \cup \{v\})$,

$$(g_j, l_j) \leftarrow (g_j, \min(l^p_j, l^q_j)), \tag{3}$$

where $(g_j, l^p_j) \in T_{xy}(S)$ and $(g_j, l^q_j) \in T^v_{xy}$. This operation takes time $O(k)$.

The update operation is defined as:

$$Update(T_{xy}(S \cup \{v\})) := Compare \begin{cases} Conc.(T_{xv}(S), T_{vy}(S)) \\ T_{xy}(S) \end{cases}$$

The time complexity of *update* takes $O(k^2 + k)$, where k is the number of entries in each table. In sum, the total time complexity is $O(|V|^3 k^2)$ for Algorithm 1. This algorithm only records the optimal tradeoff from s to t. However, it is easy to also record the path by inserting one more column in every table. This column reports the path and allocated time slot setting at each edge from x to y and then we can get both time slot allocation and its corresponding path.

C. Single Stream with Multiple Batches

When the single stream has multiple batches we look for the path with minimum bottleneck delay. Algorithm 1 can be slightly modified to solve this problem by adjusting the *concatenate* operation. Specifically, considering a stream sending multiple batches, the end-to-end delay is the largest allocated time slot among all edges (weight of the bottleneck link) in the route. In this case, we can retain most parts of the algorithm in Section IV-B but modify *concatenate* operation from summing allocated time slots to taking the maximum.

Concatenate for Multiple Batches For each entry $(g_j, l_j) \in T^v_{xy}$, the allocated time slot l_j is equal to the minimum of maximum allocated time slots from all the combinations between (g_p, l_p) and (g_q, l_q). That is,

$$(g_j, l_j) \leftarrow (g_j, \min_{g_p \cdot g_q \geq g_j} (max(l_p, l_q)), \tag{4}$$

where $(g_p, l_p) \in T_{xv}(S)$, $(g_q, l_q) \in T_{vy}(S)$. The analysis of time complexity is the same as Section IV-B. Therefore the running time for single stream with multiple batches routing is also $O(|V|^3 k^2)$, k is the number of entries for each table.

D. Multiple Streams with Multiple Batches

For multiple streams, the latency bound is the maximum latency (bottleneck) on a link to fulfill the reliability requirement of all streams. In general, multiple streams may share edges. Since an edge sends only one batch at a time, the total latency on an edge per batch is the sum of slots we allocate for each stream on this edge. For the notations, we denote r^j_e

is the allocated time slots for stream j on edge e and r_e as the sum of slot for stream using edge e, which is,

$$r_e = \sum_j r_e^j. \tag{5}$$

The maximum latency per batch is denoted as δ which is the maximum of r_e in the network, which is,

$$\delta = \max_{e \in E} r_e. \tag{6}$$

In addition, different streams transmit at different rates which are characterized by the relative transmission frequency parameter b. We must consider its effect on the allocated time slots in determining δ.

We analyze and solve this problem in two parts. In the first part, we prove that finding the smallest possible δ is NP-hard. In the second part, we provide a $O(\log |V|)$ online approximation algorithm for δ where V is the node set.

1) NP-hardness Proof: To prove hardness, we reduce the edge-disjoint path problem (EDP) to multiple streams with multiple batches routing problem. In a classic EDP, given a graph $G = (V, E)$ and a set of source-sink pair vertices $\{(s_1, t_1), (s_2, t_2), ..., (s_k, t_k)\}$, we need to return whether there exists a set of edge-disjoint paths for all the pairs. Given an EDP instance, we turn it into an instance of our problem.

In the setting of our routing problem, we set the sensor network topology to be graph G and each source-sink pair (s_i, t_i) in EDP is convert to a stream $(s_i, t_i, b, 100\%, 1, 100\%)$, where the relative frequency b is an arbitrary integer greater than the number of total edges $|E|$. In addition, link reliability table at each edge e has only one entry $(100\%, 1)$, which means every edge is perfectly reliable. Now, if there is an edge-disjoint solution for all pairs in the EDP instance, all streams can be satisfied with per batch latency $\delta \leq b$. On the other hand, if there does not exist edge-disjoint paths for all the pairs, two streams will share an edge. The transmission time on that edge must be at least $2b$ time slots.

Therefore, the routing problem is an NP-hard problem. In addition, the same reduction shows that it is NP-hard to find a 2-approximation on the maximum latency of all streams.

2) Approximation Algorithm: In this section, we describe an algorithm that provides $O(\log |V|)$ approximation on the maximum latency, where $|V|$ is the number of nodes in graph G. The algorithm is online. That is, we assume the streams come one at a time and we update the latency bound δ for each time we arrange a new stream. We prove that every time after we schedule a new stream, the latency δ is no greater than $\beta\delta^*$, where $\beta = O(\log |V|)$ and δ^* is the latency of optimal scheduling for the same set of data streams. For clarity, we put the proof at Appendix A and give the main idea in this section.

Our algorithm is inspired by the online algorithm for load balancing problem, which is to minimize the maximum traffic load while satisfying all the requests [1]. The main idea is to introduce an exponential weight function on each edge and run shortest path algorithm. In our case, we choose routes only among those that meet the given reliability requirements. The good thing is that we can again use dynamic programming to handle this part. For convenience, we first define some table operations. For the notations, T_e is the reliability table of edge e and a is a constant.

- $T_e \cdot a$: For each entry $(g, l) \in T_e$, $(g, a \cdot l) \in T_e \cdot a$
- $T_e + a$: For each entry $(g, l) \in T_e$, $(g, a + l) \in T_e + a$
- a^{T_e}: For each entry $(g, l) \in T_e$, $(g, a^l) \in a^{T_e}$

The Assign-route algorithm is shown in Algorithm 2. The input is a new stream that is not assigned yet. G is the topology of the sensor network. Λ, β are the parameters used to ensure the approximation factor. After receiving the inputs of a new stream, Assign-Route constructs new tables by including the exponential function of the currently allocated time slots on each edge. Then it runs Find-minimum-time-slot-path Algorithm on these modified tables $\widehat{T}$ to assign the route. This is to make sure the new route avoids the already congested edges and achieves load balancing [Line 4 to 7]. a is a chosen constant bigger than 1.

Notice that the allocated time slot of each edge [Line 8] is calculated from the exponential reliability tables. Therefore, we need to run Find-minimum-time-slot-path again with respect to original link reliability tables [Line 9] to get the right allocated time slot setting with the correct scale [Line 10]. We record the assignment of time slots in each edge for this new stream at [Line 11]. And update latency bound δ [Line 12]. In the end, we test whether the current estimate of latency bound δ^{new} is still within the approximation parameter β or not [Line 13 to 17]. We show $\beta = O(\log |V|)$ in Appendix A.

The time complexity for Algorithm 2 is the one of transforming each entry of tables to exponential value and the one of running Algorithm 1. Therefore, the total complexity is $O(|E|k + |V|^3 k^2)$.

3) Scheduling Algorithm: In the previous subsections, we describe how we choose routes with an online fashion based on applications' reliability needs. For our scheduling protocol of each hop, we use TSCH which inherits from WirelessHART and ISA100.11a [2]. Our scheduling algorithm is sender-based slots sharing which is inspired by Simon et al [4]. The algorithm is to determine the specific time slots for packet transmission and avoid the internal interference.

To determine the transmission schedule of all the nodes, we randomly pick an unscheduled node and let it pick the earliest time slots greedily without causing interference with the existing schedule. We iteratively pick a new node to schedule until all nodes decide their transmission schedule.

For instance, assume we have 4 nodes with node x & y having interference and node z & w having interference. Each node needs 3 time slots to transmit. According to our sender-based algorithm, we may assign node x & z using time slots 1 to 3 and node y & w using time slots 4 to 6 for transmissions.

Note that this transmission scheduling algorithm is a heuristic greedy algorithm. However, if the optimal algorithm uses the same way of avoiding internal interference as us, the objective function remains the same and the approximation factor $O(\log |V|)$ still holds.

Algorithm 2 Assign-route

procedure ASSIGN-ROUTE($s, t, b, \mu, p, \xi, G, \Lambda, \beta, r$)
  $/*$ Λ: upper bound of latency
  $/*$ β: designed performance guarantee
  $/*$ r: current allocated time slots for previous streams
  $\forall e \in E, \tilde{r}_e \leftarrow r_e/\Lambda$
  $\forall e \in E, \tilde{T}_e \leftarrow T_e \cdot b/\Lambda$
  $\forall e \in E, \widehat{T}_e \leftarrow a^{\tilde{T}_e+\tilde{r}_e} - a^{\tilde{r}_e}$
  $\widehat{T}_{st}(V)$ = Find-minimum-time-slot-path(s, t, p, ξ) w.r.t. link reliability tables $\widehat{T}$. Schedule stream(s, t, μ) with $\widehat{T}_{st}(V)$ to get the route P.
  $\forall e \in P, \acute{T}_e \leftarrow T_e$ ▷ If $e \notin P$, $\acute{T}_e$ is empty
  $\acute{T}_{st}(V)$ = Find-minimum-time-slot-path(s, t, p, ξ) w.r.t. link reliability tables $\acute{T}$. Schedule stream(s, t, μ) with $\acute{T}_{st}(V)$ to get the allocated time slot setting $\acute{l}_e$ in P
  $\forall e \in P$, $r_e \leftarrow r_e + \acute{l}_e \cdot b$
  $\delta^{new} \leftarrow \max_{e \in E} r_e$
  if $\delta^{new} > \beta\Lambda$ **then**
    $return: \ fail$
  else
    $return: \ success, r$
  end if
end procedure

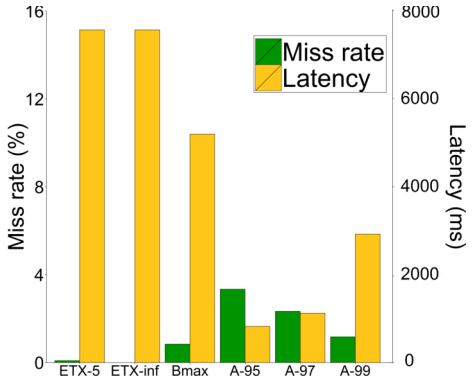


Fig. 8: Comparison on reliability of different algorithms. Our algorithm A-95 (A-97,A-99) satisfies stream requirements of 95% (97%,99%) and the latency of our algorithm is only 10.8% (14.8%,38.6%) of ETX and 15.8% (21.6%,56.2%) of Bmax.

V. EVALUATION

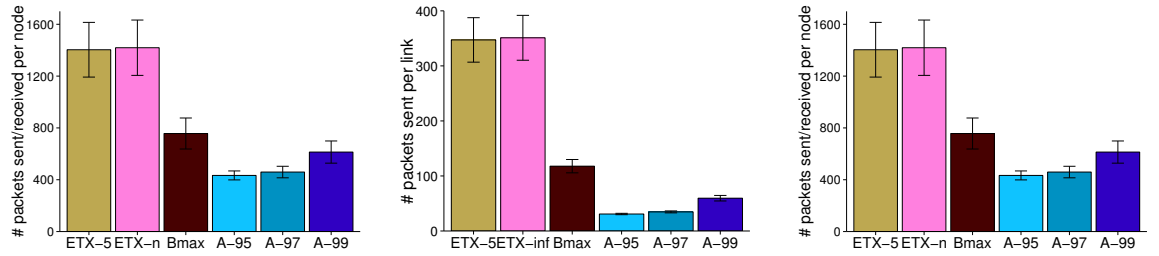
For the evaluation, we collected execution trace of 17 days from the testbed. We divide traces into two separate set, 12 days for training and 5 days for testing. The training part of the historical information is used to build the link reliability tables of each link in the network. The granularity of the tables is 0.5%. Then we generate 300 streams with random source and destination pair. For each stream, the batch requirement (BPRR) is fixed at 80% and the batch size is 5 packets. Both tables and information of 300 streams are the inputs to our Assign-route algorithm. After the algorithm selects the route and allocated time slots of each stream, we simulated the stream transmissions with these settings using the testing set to determine whether a packet transmission is successful.

To compare the performance of our algorithm and the baseline ones, we evaluate the empirical latency, miss rate, and energy consumption under the setting of 300 streams between random source and destination. For the baseline algorithms, we perform shortest path routing with the weight being Bmax [17] and ETX [3]. After route selection, the allocated time slots on each edge of a Bmax shortest path is naturally the Bmax value (Bmax). For paths based on ETX, the number of retransmissions is 5 (ETX-5) and infinity (ETX-inf). For our algorithms, we run A(ssign-route) Algorithm with different stream reliability requirements at 95%, 97% and 99% (A-95, A-97, A-99). After we determine the route and allocated time slots in each algorithm, we can empirically study their performance.

In Figure 8, the miss rate from the experiment shows that our algorithms all are able to satisfy the required stream reliability requirement (miss rate $\leq 1 - \mu$). The miss rates of the baseline algorithms are lower. The reasons are two-folded. First, Bmax and ETX-inf use much more re-transmissions than our algorithm because Bmax considers worst-case burstiness and ETX-inf allocates time slots until the batches are transmitted successfully. This will be reflected in the energy consumption. Second, both ETX algorithms over-utilize the most reliable links in the network and will cause congestion in them. This will be reflected in both latency and load balancing in a heavily used network. From the simulation of 300 streams, our algorithms achieve a better tradeoff between reliability (miss rate) and latency. While our algorithms meet all the required reliability, latency is significantly reduced.

The energy consumption of different algorithms to fulfill all the streams is computed based on the total number of packet transmissions. We assume a single packet transmission over any direct wireless link will consume the same amount of energy. Figure 9a shows the average number of packet transmissions per stream. Since each stream needs to transmit 3 ($b = 3$) batches of 5 packets ($p = 5$), the minimum number of packet transmissions in each course of time per stream is 15. The energy consumption of Bmax, ETX based routing is higher because they choose long paths that are as reliable as possible and they re-transmit for a longer period. While our algorithms satisfy all the reliability requirement, they only need to transmit $\frac{1}{2}$ or even $\frac{1}{3}$ amount of packets compared to the baseline algorithms. As our algorithms try to minimize the bottleneck, they will avoid using the same set of edges for different streams. This results in more disjoint paths. In Figure 9b and Figure 9c, our algorithms achieve a good load balancing of energy among nodes and transmitting edges. This is a particularly important feature if the network is battery-powered.

In Figure 10, we report the how latency observed empirically from the experiments changes when the number of streams increases. From the figure, A(ssign-route) algorithms achieve low latency compared to other methods since we effectively minimize bottleneck while maintaining reliability requirement. Moreover, we can get a lower latency with a lower reliability requirement. This practical result matches the theoretical latency bound. On the other hand, the latency



(a) Comparison on average energy consumption per stream of different algorithms

(b) Comparison on average packets sent per link of different algorithms

(c) Comparison on average packets sent and received per node of different algorithms

Fig. 9: Energy consumption comparison in different aspects between ETX-based, Bmax-based, and our algorithm

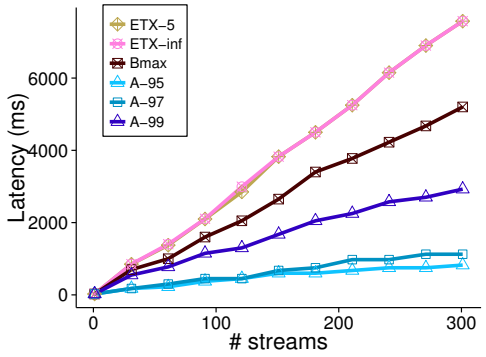


Fig. 10: Latency vs. Traffic Load. The growth rate of latency for our algorithms is much lower than Bmax and ETX.

increases less than linearly with the number of streams for our algorithms. This shows the scalability of A-route algorithms to avoid congestion in a network with a heavy workload.

VI. Related Work

Providing reliable real-time transmission in wireless sensor network has been explored in many early [5], [24], and recent works [16], [21], [11]. In those research, estimating link reliability plays an important role, which is challenging due to unpredictable noise and interference. There are many proposed metrics trying to model link quality such as β-factor, [23], Bmax [17], or Gilbert-based models [13]. Meanwhile, other research builds real-time systems by combining these metrics with routing or schedule algorithms to provide reliable transmission among all streams. For example, Le et al. [14] use a physical model to predict packet loss on each link and further provided an energy efficient transmission with end-to-end reliability guarantee. Suriyachai [24] and Potter [21] use Bmax to measure link quality and develop heuristic scheduling algorithm to ensure end-to-end transmission reliability. These works focus on packet scheduling to reduce interference and also improve reliability.

Different from scheduling algorithms, routing algorithms for reliable communication are usually focus on latency and energy issues. Hammoudeh et al. [10] maintain reliability in clustering routing by rotating cluster heads periodically to balance energy consumption. In addition, there are other papers that consider the balance between latency and energy consumption. Guha et al. [8] observe the tradeoff between energy, latency, and link capacity. They aim to optimize the unicast latency and capacity while maintaining energy conservation at a certain level. Naveen et al. [18] provide a favorable tradeoff between end-to-end packet delay and the number of nodes in the forwarding path. Lastly, some of the works focus on wireless sensor networks with unreliable links. For example, Guo et al. [9] use Opportunistic Flooding to predetermine routing and working schedule. Their works have a shorter delay while consuming only 20%-60% of the transmission energy compared to other traditional designs. However, the scenario is for broadcast communication, which cannot apply to unicast cases.

Compared to their works, we focus on unicast streaming scenarios with varied end-to-end reliability requirements. By exploring empirical data trace, we use data-driven methods to identify link quality and build tables for balancing reliability and delay at each link. We provide new routing algorithms for reliable transmission with load-balancing and minimum latency.

VII. Conclusion

In this paper, we proposed algorithms to achieve reliable communication and minimize end-to-end latency bound. Our algorithms are motivated by the empirical study of link quality characteristics and have demonstrated superior performance in simulations.

Acknowledgments

This research is supported by National Science Foundation (CNS-1618391, CNS 1553272, CNS 1536086, CNS 1463722, IIS 1460370, DE-EE0007682) and Air Force Office of Scientific Research (FA9550-14-1-0193.)

Appendix

Theorem 1: Algorithm Assign-route can be used to achieve $O(\log |V|)$ competitive ratio with respect to latency bound δ

To prove theorem 1, we firstly define the notations of our scenario and prove lemma 1. Followed by lemma 1 we acquire theorem 1.

Let $P = \{P_1, P_2, ..., P_h\}$ be the routes assigned to stream 1 through h by the online algorithm, and let $P^* = \{P_1^*, P_2^*, ..., P_h^*\}$ be the routes assigned by the optimal algorithm. Given a set of routes P, define the sum of allocated time slots at edge e after the first j streams are satisfied by

$$r_e(j) = \sum_{i: e \in P_i,\ i \le j} r_e^i$$

and let $\delta(j) = \max_{e \in E} r_e(j)$. Similarly, define $r_e^*(j)$ and $\delta^*(j)$ to be corresponding quantities for the routes produced by optimal algorithm. Our goal is to produce a set of route P that minimized $\delta(h)/\delta^*(h)$, we denoted as δ/δ^* for simplicity.

For convenience, we define the notion of a designed performance guarantee β as follows: the algorithm accepts a parameter Λ and never creates the load that exceeds $\beta\Lambda$. The algorithm is allowed to return "fail" and refuses to route a circuit if $\Lambda < \delta^*$, otherwise, it has to route all streams. Notice that we can use a doubling technique to overcome the problem that Λ is unknown.

Lemma 1: If $\delta^* \leq \Lambda$, then there exists $\beta = O(log\ |V|)$ such that algorithm Assign-route never fails. Thus, the sum of allocated time slots for all streams at any edge never exceeds $\beta\Lambda$

Proof: To simplify the formula, we will use tilde to denote normalization by Λ, for example $\tilde{r}_e(j) = r_e(j)/\Lambda$. Define the potential function:

$$\phi(j) = \sum_{e \in E} a^{\tilde{r}_e(j)}(\tau - \tilde{r}^*_e(j)),$$

where $a, \tau > 1$ are constants. Note that ϕ is a function of both $\tilde{r}_e$ and $\tilde{r}^*_e$. If the online algorithm satisfies the $(j+1)$st stream with route P_{j+1} and the optimal algorithm satisfies it with route P^*_{j+1}, we get the following change in the potential function:

$$\begin{aligned}
\phi(j+1) - \phi(j) &\leq \sum_{e \in P_{j+1}} (\tau - \tilde{r}_e(j))(a^{\tilde{r}_e(j) + \tilde{r}_e^{j+1} \cdot p_{j+1}} - a^{\tilde{r}_e(j)}) \\
&\quad - \sum_{e \in P^*_{j+1}} a^{\tilde{r}_e(j)} \cdot \tilde{r}_e^{j+1^*} \cdot p_{j+1} \\
&\leq \sum_{e \in P^*_{j+1}} (\tau - \tilde{r}_e(j))(a^{\tilde{r}_e(j) + \tilde{r}_e^{j+1} \cdot p_{j+1}} - a^{\tilde{r}_e(j)}) - a^{\tilde{r}_e(j)} \cdot \tilde{r}_e^{j+1^*} \cdot p_{j+1} \\
&\leq \sum_{e \in P^*_{j+1}} a^{\tilde{r}_e(j)}(\tau(a^{\tilde{r}_e^{j+1} \cdot p_{j+1}} - 1) - \tilde{r}_e^{j+1} \cdot p_{j+1}).
\end{aligned}$$

The first inequality follows because $r_e(j+1) \leq r_e(j) + r_e^{j+1} \cdot p_{j+1}$, where p_{j+1} is the parameter of batch size in stream $j+1$. The second inequality holds from the fact that P_{j+1} is the path with minimum sum of allocated time slots for stream $j+1$ with respect to the tables $a^{\tilde{T}_e + \tilde{r}_e} - a^{\tilde{r}_e}$. The last inequality follows because $\tilde{r}_e^{j+1^*} \leq \tilde{r}_e^{j+1}$ and we also ignores the value of $\tilde{r}_e(j)$.

Since the $j+1$st stream is satisfied by the optimal algorithm by assigning it the route P^*_{j+1}, which means that $e \in P^*_{j+1}$: $0 \leq \tilde{r}_e(j+1) \leq \delta^*/\Lambda \leq 1$. Therefore, in order to show that the potential function does not increase, it is sufficient to show that $x \in [0,1] : \tau(a^x - 1) \leq x$, which is true for $a = 1 + 1/\tau$.

Initially, $\phi(0) \leq \tau|E|$. Since ϕ does not increase, and since $r^*(j) \leq 1$, then after satisfying h requests, we have

$$\sum_e (\tau - 1)a^{r_e(|E|)/\Lambda} \leq \tau|E|.$$

The last inequality, and the fact that $\tau > 1$ implies

$$\max_{e \in E} r_e(k) \leq \Lambda \log_a(\frac{\tau|E|}{\tau - 1}) = O(\Lambda \log |V|).$$

■

REFERENCES

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3):486–504, 1997.

[2] I. S. Association et al. Ieee standard for local and metropolitan area networks-part 15.6: wireless body area networks. *IEEE Standard for Information Technology, IEEE*, 802(6):1–271, 2012.

[3] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11(4):419–434, 2005.

[4] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 337–350. ACM, 2015.

[5] S. Ehsan and B. Hamdaoui. A survey on energy-efficient routing techniques with qos assurances for wireless multimedia sensor networks. *Communications Surveys & Tutorials, IEEE*, 14(2):265–278, 2012.

[6] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[7] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, 2009.

[8] S. Guha, P. Basu, C.-K. Chau, and R. Gibbens. Green wave sleep scheduling: optimizing latency and throughput in duty cycling wireless networks. *Selected Areas in Communications, IEEE Journal on*, 29(8):1595–1604, 2011.

[9] S. Guo, L. He, Y. Gu, B. Jiang, and T. He. Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links. *Computers, IEEE Transactions on*, 63(11):2787–2802, 2014.

[10] M. Hammoudeh and R. Newman. Adaptive routing in wireless sensor networks: Qos optimisation for enhanced application performance. *Information Fusion*, 22:3–15, 2015.

[11] S. Jang, H. Jo, S. Cho, K. Mechitov, J. A. Rice, S.-H. Sim, H.-J. Jung, C.-B. Yun, B. F. Spencer Jr, and G. Agha. Structural health monitoring of a cable-stayed bridge using smart sensor technology: deployment and evaluation. *Smart Structures and Systems*, 6(5-6):439–459, 2010.

[12] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *2007 6th International Symposium on Information Processing in Sensor Networks*, pages 254–263. IEEE, 2007.

[13] S. M. Kim, S. Wang, and T. He. cetx: Incorporating spatiotemporal correlation for better wireless networking. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 323–336. ACM, 2015.

[14] T. Le, W. Hu, P. Corke, and S. Jha. Ertp: Energy-efficient and reliable transport protocol for data streaming in wireless sensor networks. *Computer Communications*, 32(7):1154–1171, 2009.

[15] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The smart thermostat: Using occupancy sensors to save energy in homes. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, 2010.

[16] M. A. Mahmood, W. K. Seah, and I. Welch. Reliability in wireless sensor networks: A survey and challenges ahead. *Computer Networks*, 79:166–187, 2015.

[17] S. Munir, S. Lin, E. Hoque, S. Nirjon, J. A. Stankovic, and K. Whitehouse. Addressing burstiness for reliable communication and latency bound generation in wireless sensor networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 303–314. ACM, 2010.

[18] K. P. Naveen and A. Kumar. Tunable locally-optimal geographical forwarding in wireless sensor networks with sleep-wake cycling nodes. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[19] M. Pajic, S. Sundaram, J. Le Ny, G. J. Pappas, and R. Mangharam. Closing the loop: A simple distributed method for control over wireless networks. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*, IPSN '12, 2012.

[20] W.-B. Pöttner, H. Seidel, J. Brown, U. Roedig, and L. Wolf. Constructing schedules for time-critical data delivery in wireless sensor networks. *ACM Trans. Sen. Netw.*, 2014.

[21] W.-B. Pöttner, H. Seidel, J. Brown, U. Roedig, and L. Wolf. Constructing schedules for time-critical data delivery in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 10(3):44, 2014.

[22] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Real-time scheduling for wirelesshart networks. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, RTSS '10, 2010.

[23] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The beta-factor: Measuring wireless link burstiness. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, 2008.

[24] P. Suriyachai, J. Brown, and U. Roedig. Time-critical data delivery in wireless sensor networks. In *Distributed Computing in Sensor Systems*, pages 216–229. Springer, 2010.