Reliable and Real-time Communication in Industrial Wireless Mesh Networks

Song Han, Xiuming Zhu, Aloysius K. Mok
University of Texas at Austin
{*shan, xmzhu, mok*}*@cs.utexas.edu*

Deji Chen, Mark Nixon
Emerson Process Management
{*deji.chen, mark.nixon*}*@emerson.com*

***Abstract*— Industrial wireless mesh networks are deployed in harsh and noisy environments for process measurement and control applications. Compared with wireless community networks, they have more stringent requirements on communication reliability and real-time performance. Missing or delaying of the process data by the network may severely degrade the overall control performance. In this paper, we abstract the primary reliability requirements in typical wireless industrial process control applications and define three types of reliable routing graphs for different communication purposes. We present efficient algorithms to construct them and describe the recovery mechanisms. Data link layer communication schedules between devices are further generated based on these graphs to achieve end-to-end real-time performance. We have built a complete WirelessHART communication system and integrated our solutions into its Network Manager. We demonstrate through extensive experiment results that our algorithms can achieve highly reliable routing, improved communication latency and stable real-time communication in large-scale networks at the cost of modest overheads in device configuration.**

I. Introduction

Wireless process control has been increasingly recognized as an important technology in the field of industrial process management [1], [2], [3], [4], [5], [6], [7], [8]. Several industrial organizations such as ISA [9], HART [10] and ZigBee [11] have been actively pushing the application of wireless technologies in industrial automation. However, compared with wireless community networks, the industrial control environment is harsher and noisier and thus has more stringent requirements on reliable and real-time communication. Missing or delaying the process data may severely degrade the quality of control. The shifting wireless signal strength with time and location, the mobility of the control devices and power limitation due to battery usage make the problem even worse. Accordingly, network management techniques adapted for industrial wireless mesh are critical.

WirelessHART [12] is the first global wireless communication standard approved by IEC, and it is specifically designed for process measurement and control applications. Unlike the decentralized control adopted by wireless ad-hoc or peer-to-peer networks, it advocates explicit and centralized network management. The standard pushes the complexity of ensuring reliable and real-time communication to a centralized Network Manager, but it provides little guidance on how to meet the demanding design goals. This paper attempts to bridge the gap and shall explore efficient approaches for forming a WirelessHART network, managing reliable graph routing, allocating network resources and constructing data link layer communication schedules.

In a typical WirelessHART network, each device has a designated sample rate to publish its process data to the Gateway through multi-hop transmissions. In the other direction, the Network Manager sends the control data back to the devices periodically. To help relay different types of data, the standard defines three types of communication graphs. The network shares one broadcast graph for propagating common control messages and one uplink graph for devices to publish process data. If needed, each device further has a unique downlink graph from the Network Manager for forwarding specific control messages to itself. Although several research works have been devoted on the design of data link layer scheduling in WirelessHART networks [8], [13], [14], [15], how to satisfy the enforced strict reliability requirements on the routing graphs and construct data link layer communication schedules on top of them is still a challenging problem and has not received sufficient attentions.

In this paper, we abstract the reliability requirements for packet routing defined in WirelessHART standard. We present efficient algorithms to construct these reliable graphs and describe the recovery mechanisms. These algorithms are designed to maintain the maximum number of reliable nodes in the graphs while achieving good network latency. To improve the scalability of the downlink graphs in large-scale networks, we further propose an extension on the standard to replace the single downlink graph with a sequence of ordered local graphs. These local graphs work as reusable building blocks in constructing downlink graphs for different destinations thus greatly reduce the overall overhead in device configuration.

Based on these routing graphs, the data link layer communication schedule is further constructed. Our approach allows multiple devices to compete for the retry links to the same device, and split the traffic from one device among all its successors, thus reduces the bandwidth allocation on each of them. By designing the communication schedules on the successors so that their combination has the same communication pattern as the original device, the global communication schedule is then spliced into sub-schedules and distributed to the corresponding devices. These sub-schedules work together and guarantee that the periodic process/control data between devices and the Gateway can be forwarded through multi-hops in a timely manner.

We have conducted extensive experiments to evaluate the performance of the proposed algorithms. We have also built a complete WirelessHART communication system, and integrated our network management solutions into the Network Manager. We are deploying this system to a large-scale manufacturing factory to achieve factory automation.

The remainder of this paper is organized as follows. Section II briefly describes the WirelessHART network architecture. Section II reviews the previous works on reliable routing and real-time scheduling in WirelessHART networks. Section IV presents the fundamental synchronization mechanism applied in WirelessHART networks. The details of reliable graph routing and communication schedule construction in WirelessHART are described in Section V and Section VI. Section VII presents our design and implementation of the complete WirelessHART communication system. Section VIII summarizes our experiment results. We conclude the paper and discuss the future works in Section IX.

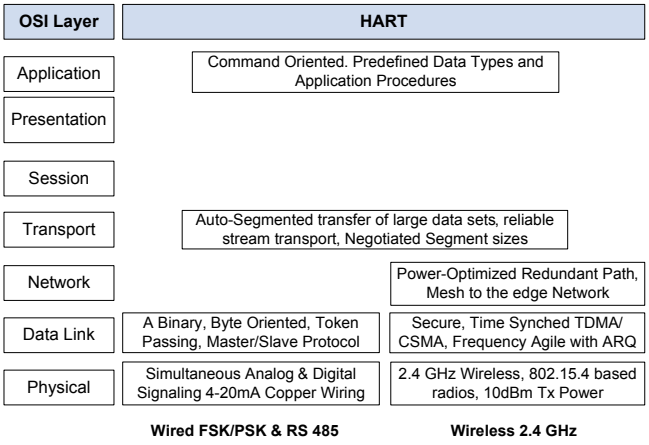

Fig. 1. The architecture of HART communication protocol

II. WirelessHART Architecture

Traditional wireless standards for office and manufacturing automation such as ZigBee [11] and Bluetooth [16] are not designed to meet the stringent timing and security requirements of industrial control. The WirelessHART standard is specifically targeted to solve these problems and provide a complete solution for real-time process control applications.

Figure 1 illustrates the architecture of the HART protocol according to the OSI 7-layer communication model. As a part of the HART protocol, the architecture of WirelessHART protocol is shown on the right side of Fig. 1. At the bottom of its communication stack, WirelessHART adopts IEEE 802.15.4-2006 [17] as the physical layer. On top of that, WirelessHART defines its own time-synchronized data link layer. Some notable features of WirelessHART data link layer include strict 10 ms timeslot, network-wide time synchronization, channel hopping, channel blacklisting, and industry-standard AES-128 ciphers and keys. The network layer supports self-organizing and self-healing mesh networking techniques and uses source routing and graph routing. In this way, messages can be routed around interferences and obstacles and greatly improve the network performance in noisy and harsh environments. WirelessHART distinguishes itself from other public standards by maintaining a central Network Manager. The Network Manager is responsible for maintaining up-to-date routes and communication schedules for the network, thus guaranteeing the reliable and real-time network communications.

Fig. 2 shows a typical topology of a WirelessHART mesh network. All WirelessHART nodes support the basic mesh node functionalities, including routing capability. The basic node types of a WirelessHART network are:

- **Network Manager:** It is responsible for configuring the network, scheduling and managing communication among WirelessHART devices. It is implemented in software that resides in the Gateway or the Host.
- **Gateway:** It connects Host applications with field devices. It is responsible for data caching and query processing.
- **Access Point:** It is attached to the Gateway and provides redundant paths between the wireless network and the Gateway.
- **Router:** It is deployed in the network to improve network coverage and connectivity.
- **Field Device:** It is attached to the process plant and collects data. It could be a sensor or an actuator.
- **Handheld:** It is a portable WirelessHART-enabled computer used to configure devices, run diagnostics, and perform calibrations.
- **Adapter:** It is a bridge device between the wireless mesh network and traditional wired HART devices.

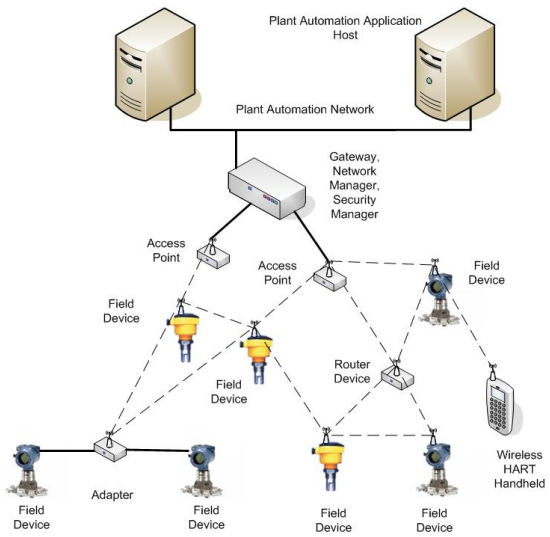

Fig. 2. A typical topology of a WirelessHART mesh network

III. Related Works

In this section, we summarize previous works in the literature on achieving reliable routing in wireless networks, and describe recent works on link and channel scheduling in WirelessHART networks to achieve end-to-end real-time communication.

A. Reliable Routing in Wireless Networks

The reliable graph routing defined in WirelessHART standard is essentially a multipath routing approach which has been extensively studied in wireless networks, and recognized as an efficient approach for improving the routing reliability [18], [19], [20], [21], [22], [23]. In [20], node-disjoint and braided multipath schemes are proposed to provide energy efficiency and resilience against node failures. SMR [21] is a multipath version of DSR. It is designed to utilize multipath concurrently by splitting traffic onto two maximally disjoint routes. AOMDV [22] is a multipath, loop-free extension to AODV. It ensures that alternate paths at every node are disjoint, therefore achieves path disjointness without using source routing. AODVM [23] is another extension to AODV for finding multiple node-disjoint paths. It also proposes an infrastructure to include deployment of reliable nodes which can route on multiple paths. This infrastructure can increase the number of node-disjoint paths between the source and the destination especially when they are far apart.

Most of these works focus on identifying multiple node or edge-disjoint paths to improve the routing reliability. However, to deal with much harsher and noisier industrial control environments, the WirelessHART standard defines more stringent requirements on routing reliability. Each intermediate node on the routing graph must have at least two neighbors to forward the traffic to the destination. For this reason, the works in the literature cannot be directly applied in WirelessHART networks, and new routing algorithms have to be designed.

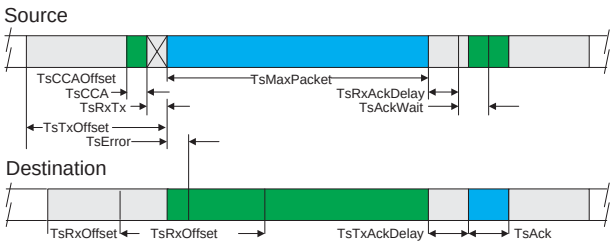

Fig. 3. Timing of a WirelessHART timeslot

B. Real-time Scheduling in WirelessHART Networks

Since the standard was ratified in 2007, several research works have been devoted to the link scheduling and channel assignment problems in WirelessHART networks to achieve end-to-end real-time communication [13], [14], [15]. In [14], [15], the convergecast scheduling problem is studied in linear network topologies. They formulate the problem as a mixed integer linear programming problem, and design algorithms based on different assumptions on devices' buffering capability. [13] considers a more general WirelessHART network model including arbitrary network topology and multi-path routing. It formulates the sensor-to-actuator real-time flow scheduling problem and proves that it is NP-hard. Based on a necessary condition for schedulability in WirelessHART networks, it proposes an optimal scheduling algorithm based on a branch-and-bound technique. A practical scheduling policy called Conflict-aware Least Laxity First algorithm is also proposed to achieve better scalability and handle network dynamics.

However, all these aforementioned works assume that the network layer routes have already been provided and focus on data link layer scheduling. The relationship between the routes and the data link layer schedules are not thoroughly studied. In our work, we present a general framework for network management in WirelessHART networks. We shall study how to achieve reliable graph routing for different communication paradigms in WirelessHART network and further construct a data link layer communication schedule based on them. Our solution can be easily integrated into the Network Manager, so that the setup of an operational WirelessHART network is simple and prompt.

IV. Time Synchronization in WirelessHART Networks

WirelessHART is a TDMA-based network protocol and every communication in it is time-synchronized. The basic time unit of communication activity is a fixed-length timeslot that is commonly shared by all network devices. The timeslot provides the time base for scheduling process data transmission. The duration of a timeslot defined in WirelessHART is 10 ms which is sufficient for sending or receiving one packet per channel and the accompanying acknowledgement, including guard-band times for network-wide synchronization. The specific timing requirement inside a WirelessHART timeslot is depicted in Figure 3. Precise time synchronization is critical to the operation of networks based on time division multiplexing. Since all communication happens in timeslots, the network devices must have the same notion of when each timeslot begins and ends, with minimal variation. Several mechanisms are applied in WirelessHART for time synchronization. In a WirelessHART network, time propagates outwards from the Gateway.

When a new device joins a WirelessHART network initially, it has no idea what the current time is. For each incoming MAC layer packet, the device records T_a, the time when the packet's first bit arrives. Because of the strict timeslot structure, the device can derive the start of the next timeslot, T, from the packet's arrival time according to the following formula where TsTxOffset is the offset in the slot to start the preamble transmission.

$$T = T_a + 10\text{ms} - \text{TsTxOffset}$$

Synchronization happens not only in the device join process, but also during a node's normal operation. A receiving node always compares the start time of the incoming MAC layer packet and the expected arrival time measured on its own clock. The difference is the drift between their clocks. The receiver includes the difference in the time adjustment field of the corresponding ACK packet. Each node is designated a time source node. Whenever a node receives an ACK from its time source, it will adjust its clock based on the time adjustment field. If the sender is the time source of the receiver, the receiver adjusts its clock directly from the time difference value. Together, these adjustments provide the network-wide time synchronization in WirelessHART mesh networks.

V. Reliable Graph Routing

In this section, we present the details how we define and achieve the reliable routing in a typical wireless industrial mesh network like WirelessHART. We first describe the primary routing approaches adopted in WirelessHART in Section V-A. Section V-C abstracts the reliability requirements on packet routing, defines three types of reliable graphs for different communication purposes, and describes their properties. We discuss the difficulties in achieving completely reliable routing in Section V-D. The algorithmic details to construct these routing graphs are presented in Section V-E, Section V-F, Section V-G and Section V-H. We describe the recovery mechanisms in Section V-I.

A. Source Routing and Graph Routing

Two primary routing approaches are defined in the WirelessHART standard: graph routing and source routing. When using graph routing, a network device sends packets with a graph id in the network layer header along a path to the destination. All devices on the way to the destination must be pre-configured with graph information that specifies the neighbors to which the packets may be forwarded.

With source routing, to send a packet to its destination, the source includes in the network layer header an ordered list of devices through which the packet must travel. As the packet is routed, each routing device utilizes the next network device address from the packet header to determine the next hop to use. Since packets may go to a destination without explicit setup of intermediate devices, source routing requires knowledge of the complete network topology.

Since the source routing approach only establishes a fixed single path between the source and destination, any link or node failure will cut off their communication. For this reason, source routing is mostly used for diagnostics purposes in industrial wireless networks. In this paper, we will focus on the graph routing approach and investigate how to achieve reliable routing in the network. Based on different communication purposes, there are three types of routing graphs defined in a WirelessHART network, and Figure 4 illustrates an example.

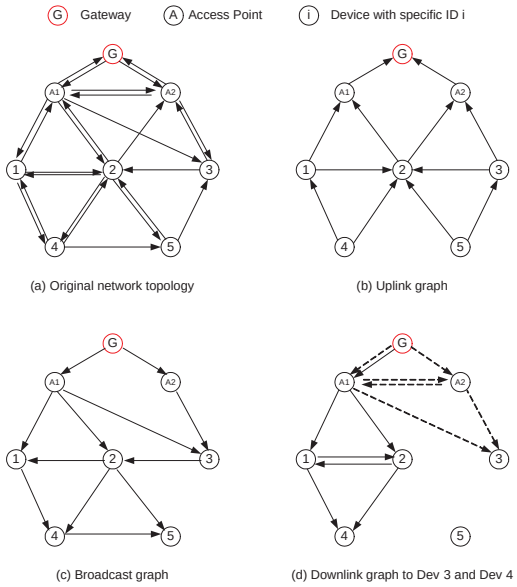

Fig. 4. Three types of routing graphs

Uplink graph: It is a graph connecting all devices upward to the Gateway. It is used to propagate devices' process data periodically to the Gateway. Different devices may have different sample rates.

Broadcast graph: It is a graph connecting the Gateway downward to all devices. It is used to broadcast common configuration and control messages to the entire network.

Downlink graph: It is one per device. It is the graph from the Gateway to each individual device. The unicast messages from the Gateway and the Network Manager to each device traverse through this graph.

Based on these graphs, the Network Manager can further generate the corresponding sub-routes for each device. Only after the routes are constructed and downloaded to every device, can the network communication schedule be generated, which we shall elaborate in Section VI. When devices initially join into the network, they carry with them a list of neighbor entries including the received signal strength information. The Network Manager uses this information and the periodic health reports from the devices to construct and maintain the global network topology.

B. Notations

This section summarizes the notations to be used throughout the paper. Given the original network topology $G(V, E)$, we use g to denote the Gateway, V_{AP} to denote the set of Access Points and V_D to denote the set of devices. We have $\{g\} \cup V_{AP} \cup V_D = V$. For each node $i \in V_D \cup V_{AP}$, we use e_i^+ and e_i^- to denote its set of outgoing edges and incoming edges. We use δ_i^+ and δ_i^- to denote its outgoing and incoming degree. $G_B(V_B, E_B)$ and $G_U(V_U, E_U)$ are used to represent G's reliable broadcast graph and uplink graph. The reliable downlink graph for node $v \in V$ is denoted by $G_v(V_v, E_v)$.

C. Reliability Requirements and Reliable Graphs

Compared with wireless community networks, industrial wireless mesh networks have a much higher demand on the routing reliability to tolerate node and link failures. In this section, we abstract the reliability requirements defined in WirelessHART standard using the concept of (k, m)-*reliability* in packet routing. Notice that here we assume that the Gateway and Access Points are all connected through wire and reliable, so in the following of the paper, the reliability requirements only apply to wireless devices.

Definition V.1: Given a directed graph $G(V, E)$, a node $v \in V$ satisfies the (k, m)-*reliability* if and only if $\delta_v^- \geq k$ and $\delta_v^+ \geq m$. There is no constraint on δ_v^- or δ_v^+ if $k = 0$ or $m = 0$.

Based on this definition, we now give the definitions of the aforementioned three reliable routing graphs and present their important properties.

Definition V.2: Given a directed graph $G(V, E)$, a directed acyclic graph $G_B(V_B, E_B)$ ($V_B = V$ and $E_B \subseteq E$), is a reliable broadcast graph if the $(2, 0)$-reliability holds on every node in $V - \{g\} - V_{AP}$.

G_B requires that each device has at least two parents from which it can receive the broadcast messages. This significantly increases the chance for the broadcast messages to be propagated to the entire network. G_B has the following property.

Property V.1: Each device in G_B has at least two paths from g.

Proof: According to the definition of G_B, $\forall v, v \in V - \{g\} - V_{AP}$, it has two different parent nodes. There are two cases on v's parent node u. In the first case, u is an Access Point. It is obvious that there exists one path $g \rightarrow u \rightarrow v$. In the second case, u is a device. We perform the same analysis on u and find its parents. As G_B is acyclic, this process can be repeated and terminates when it reaches an Access Point. Thus there exists a path $g \rightarrow \cdots \rightarrow u \rightarrow v$. Because v has two different parent nodes, there are at least two paths from g to v in G_B. □

Different from the broadcast graph, the uplink graph is used by the devices to forward their process data to the Gateway with a required sample rate. It is considered reliable if and only if for each device in the network except the Access Points, it has two children to forward its packet to the Gateway. In cases where the communication between the device and one of its children is broken, the process data can still be delivered to the Gateway through the alternative child.

Definition V.3: Given a directed graph $G(V, E)$, a directed acyclic graph $G_U(V_U, E_U)$ ($V_U = V$ and $E_U \subseteq E$), is a reliable uplink graph if the $(0, 2)$-reliability holds on every node in $V - \{g\} - V_{AP}$.

Property V.2: Each device in G_U has at least two paths to g.

Proof: The proof is similar to the proof for Property V.1.□

Property V.3: G_B and G_U both have no less than 2 Access Points.

Proof: Assume that there is only one Access Point p in G_B. and v is a node with an incoming edge from p in G_B. As p is the only Access Point, node u, the other parent node of v is a device. We repeat this analysis on u and it is obvious that at least one of u's parents is still a device. This process will be repeated until a cycle is formed because the number of devices in the network is finite. This is a contradiction with the definition of G_B. So G_B has no less than 2 Access Points. The proof for G_U is similar. □

The broadcast graph and uplink graph are global graphs shared by the entire network. However, to support the transmission of configuration and control messages to a specific device v, a unique downlink graph $G_v(V_v, E_v)$ from the Gateway and Network Manager to v is also required. G_v is defined to be reliable only if $(0, 2)$-reliability holds on each intermediate node.

Property V.4: $G_v(V_v, E_v)$ contains at least one directed cycle.

Proof: Assume there is no cycle in G_v. Consider the node u which has a direct edge to v in G_v. According to the definition

of G_v, intermediate node u has another non-v child w. Repeat this analysis on w, and w also has a non-v child. This process can be repeated and finally form a cycle. □

Property V.4 states the existence of directed cycles in G_v. To guarantee the prompt delivery of the downlink messages, we must avoid arbitrary cycles in G_v which will generate infinite loops in packet forwarding. Thus in its definition, we restrict that there is only one cycle of length 2 in G_v and require that every node on the cycle must be the destination's parent. Once the packet reaches such nodes, it will be directly forwarded to the destination which is required by the standard. This will avoid any cyclic transmission and unnecessary delay.

Definition V.4: Given a directed graph $G(V, E)$, a directed graph $G_v(V_v, E_v)$ ($V_v \subseteq V$ and $E_v \subseteq E$), is a reliable downlink graph from g to v if 1) v is the only sink and g is the only source in G_v; 2) $(0, 2)$-reliability holds on each intermediate node in G_v; and 3) there is only one cycle of length 2 in G_v, and each node on the cycle has a direct edge to v.

D. Difficulties in Achieving Completely Reliable Graphs

The major barrier to construct reliable routing graphs is the underlying network connectivity. Better network connectivity will obviously lead to a higher chance for constructing completely reliable graphs. In this section, we evaluate the relationship between the network connectivity and the success ratio to construct these reliable graphs. In our experiments, we vary the network connectivity by changing the edge success probability p and Figure. 5 summarizes our results. We observe that with 150 devices in the experiments, the success ratio drops quickly along with the decrease of p. When p is 0.8, the success ratio is around 80% for downlink graphs and above 95% for both G_B and G_U. However, when p drops to 0.5, we can barely construct reliable downlink graphs and the success ratios for G_B and G_U are only around 40%.

Under the same experiment settings, Figure. 6 shows the percentage of reliable nodes in the incomplete reliable graphs. We observe that the percentage of reliable nodes in incomplete G_U and G_B are always above 95% and this percentage for downlink graphs is also larger than 75% even when the edge success probability drops to 0.5. Figure. 7 further evaluates the impact of the network density on the success ratio. We vary the size of the network from 75 to 150 and fix the edge success probability at 0.8. As expected, The results show that the network density has a great impact on network connectivity, and lower network density will lead to poor success ratio.

Based on these results, we conclude that the success ratio for constructing reliable routing graphs is closely related to the underlying network connectivity. In many scenarios, it is impossible to achieve the completely reliable graphs. For this reason, we shall allow violations of the reliability requirements in the routing graphs and instead focus on designing algorithms to construct graphs with the maximum number of reliable nodes. In the following of the paper, we will still use G_B, G_U and U_v to represent the broadcast graph, uplink graph and downlink graph for node v even though they may not be completely reliable.

E. Constructing Reliable Broadcast Graph

In a broadcast graph, we say that a node i is reliable if and only if $\delta_i^- \geq 2$. Let $S_B = \{i \,|\, \delta_i^- \geq 2, i \in V\}$, and we want to maximize $|S_B|$ when we construct the reliable broadcast graph G_B. Furthermore, to reduce the energy consumption in propagating broadcast messages to the entire network and improve network latency, we also hope to minimize the average number of hops from the Gateway to each node. For node i, we denote its average number of hops from the Gateway by $\bar{h}_i$ and use P_i to represent its parents in G_B. We have:

$$\bar{h}_i = \frac{\sum_{k \in P_i} \bar{h}_k}{|P_i|} + 1 \tag{1}$$

Alg 1 Constructing Reliable Broadcast Graph $G_B(V_B, E_B)$

```
1: // G(V, E) is the original graph
2: Initially V_B = g ∪ V_AP and E_B contains all links from g to V_AP.
3:
4: while V_B ≠ V do
5:    Find S' ⊆ V − V_B: ∀v ∈ S', v has at least two edges from V_B
6:    if S' ≠ ∅ then
7:       for all node v ∈ S' do
8:          Sort its edges e_{u,v} from V_B according to h̄_u
9:          Choose the first two edges e_{u1,v} and e_{u2,v}
10:         h̄_v = (h̄_{u1} + h̄_{u2})/2 + 1
11:      end for
12:      Choose the node v with min h̄_v
13:      Add v to V_B and add e_{u1,v} and e_{u2,v} to E_B
14:   else
15:      Find S'' ⊆ V − V_B: ∀v ∈ S'', v has one edge e_{u,v} from V_B
16:      if S'' ≠ ∅ then
17:         for all node v ∈ S'' do
18:            h̄_v = h̄_u + 1
19:            Calculate n_v, the # of its outgoing edges to V − V_B
20:         end for
21:         Choose the node v with maximum n_v, break tie using h̄_v
22:      end if
23:   else
24:      return FAIL;
25:   end if
26: end while
27: return SUCCESS;
```

We present a greedy algorithm (Alg. 1) to achieve these two goals in constructing G_B. In our approach, we maintain a set V_B to record the explored nodes and V_B is initialized as $\{g\} \cup V_{AP}$. The explored edges are maintained in E_B which is initialized to include the edges from g to each Access Point. In the algorithm, we incrementally select one node v from $V - V_B$. In each loop, we first find S', the set of reliable nodes in $V - V_B$ (Line 5). For each node v in S', we sort its incoming edges from V_B according to their averaged number of hops from the Gateway in ascending order. We choose the first two edges and calculate $\bar{h}_v$ according to Eq. 1. We choose the node in S' with the minimum $\bar{h}_v$ and add it to V_B. If there is no reliable node available in $V - V_B$, we will instead find S'', the set of nodes with exact one edge from V_B (Line 15). We choose the node in S'' with the maximum number of outgoing edges to $V - V_B$ to maximize the chance to find reliable nodes in the next round. This process continues until all nodes in V are explored. Otherwise an error will be reported (Line 24). This will trigger the Network Manager to execute appropriate recovery actions. The *worst-case* complexity of the algorithm is

$$\sum_{k=|V_{AP}|}^{|V|} (|E| + (|V| - k) \cdot \lg(|V| - k)) = O(|V|^3)$$

F. Constructing Reliable Uplink Graph

The construction of a reliable uplink graph $G_U(V_U, E_U)$ is similar to that of $G_B(V_B, E_B)$. Essentially, we only need to reverse all edges in the original graph $G(V, E)$, construct G_B

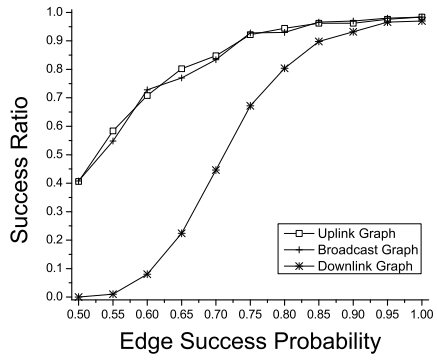

Fig. 5. Success ratio vs. Edge success probability

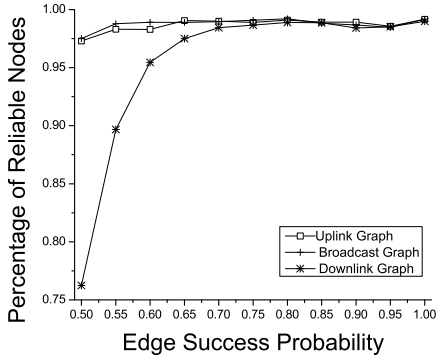

Fig. 6. Percentage of reliable nodes

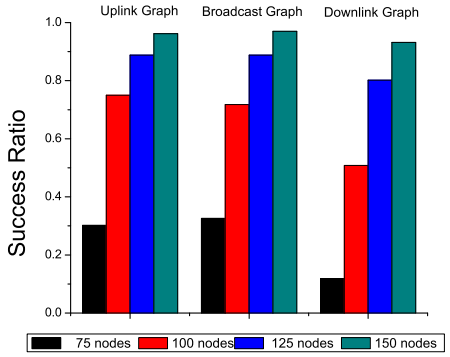

Fig. 7. Success ratio vs. Network density

and then reverse all its edges back. We define $G^R(V, E^R)$ to be the reversed graph of $G(V, E)$, and the greedy algorithm to construct $G_U(V_U, E_U)$ is summarized in Alg. 2 and its *worst-case* complexity is $O(|V|^3)$.

Alg 2 Constructing Reliable Uplink Graph $G_U(V_U, E_U)$

```
1: // G(V,E) is the original graph, G^R(V,E^R) is the reversed graph
2: Construct G^R(V,E^R)
3: Construct G_B(V_B,E_B) from G^R(V,E^R) by applying Alg. 1
4:
5: if V_B = V then
6:     // Construct G_U by reversing all edges in G_B
7:     G_U(V_U,E_U) = G_B^R(V_B,E_B^R)
8: else
9:     // the network topology is disconnected
10:    return FAIL;
11: end if
12: return SUCCESS;
```

G. Constructing Reliable Downlink Graph

The construction of the reliable downlink graph $G_v(V_v, E_v)$ for a given node v in $G(V, E)$ only involves part of the nodes in $G(V, E)$ and it is more complicated because of the existence of cycles as shown in Property V.4. Furthermore, according to Definition V.4, we want to have exactly one cycle in G_v of length 2 and restrict it to be between the two parents of v. Our optimization goals in constructing G_v are similar to that of G_B and G_U. We hope to maximize the number of nodes in the network to have reliable downlink graphs and for each downlink graph, we want to minimize its average number of hops from the Gateway.

Alg. 3 summarizes the framework of our approach. In the algorithm, we construct the reliable downlink graph for each node in the network. For the Access Point, its downlink graph consists of the Gateway g, itself and the edge from g to itself. We maintain S, a set of nodes whose reliable downlink graphs have already been constructed (Line 1). We incrementally find an eligible node v in $V - S$ to construct G_v where three constraints in Table I are applied and v has the minimum $\bar{h}_v$ as calculated in Line 17. Constraint C1 and C2 are to satisfy the reliability requirements in G_v and Constraint C3 is to make sure that we can remove the internal cycles in the constructed G_v. If such an eligible node cannot be found, we will instead choose the node that has two parents from S with the minimum average latency to the Gateway (Line 20). If every node in $V - S$ only has one parent from S, we choose the one with the minimum average latency (Line 27 - 37).

C1: v has at least two parents u_1 and u_2 in S
C2: u_1 and u_2 form a directed cycle
C3: u_2 (u_1) has at least one parent from the cycle in G_{u_1} (G_{u_2})

TABLE I
Three constraints in constructing reliable downlink graphs

Alg. 4 describes how we construct G_v based on its parents (u_1 and u_2)' reliable downlink graphs G_{u_1} and G_{u_2}. We first merge G_{u_1}, G_{u_2}, v and edges among u_1, u_2 and v together (Line 4). We maintain S, the set of explored nodes in G_v and initialize it as $\{g, v, u_1, u_2\}$. We construct G_v in a bottom-up manner by incrementally selecting a node $i \in V_v - S$ which has two outgoing edges to S in G and has the minimum $\bar{h}_i$ (Line 6-30). This process continues until either all nodes in V_v are explored or V_{AP} has two outgoing edges to S (Line 7 - 10). Finally, we remove all nodes in $V_v - S$ and their corresponding edges from G_v (Line 32 - 34). If there is no node available to have two outgoing edges to S in G, we choose the node with the minimum $\bar{h}_i$ (Line 20 - 29).

H. Constructing Scalable Reliable Downlink Graph

The algorithms proposed in Section V-G strictly comply to the WirelessHART standard and construct one downlink graph for each individual node. However, this approach is not scalable. When a device is multi-hop away from the Gateway, its downlink graph has to traverse multiple intermediate devices but cannot reuse their downlink graph information. This will introduce unnecessarily high configuration overhead in the network. To achieve reliable downlink graph routing in large-scale wireless networks, in this section we propose to extend the current downlink route from a single graph to a sequence of ordered local graphs, and we call this approach Sequential Reliable Downlink Routing (SRDR). Instead of constructing a completely new graph from Gateway to device v, SRDR lets each node only keep a small local graph to maintain the reliable routing from its parents. The reliable downlink graph to a given node can be constructed by assembling the intermediate nodes' local graphs together based on a given order. These local graphs can be taken as building blocks in constructing downlink graphs for different destinations, thus existing device configurations can be reused. This will significantly reduce the overall configuration overhead and improve the downlink routing scalability.

Extension: To support sequential reliable downlink routing, we need two extensions in the current WirelessHART standard. First, as depicted in Figure 8, we use the reserved bits (Bits 4-3) of the control byte in the network layer header to indicate, when set, the presence of the sequential downlink routing fields, and we

Alg 3 Constructing Reliable Downlink Graphs in $G(V, E)$

Let S be the set of nodes with downlink graphs constructed
Initially $S = g \cup V_{AP}$ and $G_g = (\{g\}, \varnothing)$
Initially for each AP i in S, set $G_i = (\{g \cup i\}, \{e_{g,i}\})$

while $S \neq V$ **do**
  Find $S' \subseteq V - S$: $\forall v \in S'$, v has at least two edges from S
  // S_r is the reliable node set in S', initially $S_r = \varnothing$
  if $S' \neq \varnothing$ **then**
    for all node $v \in S'$ **do**
      for all edge pair $(e_{u_1,v}, e_{u_2,v})$ from S **do**
        if C 1 $\wedge$ C 2 $\wedge$ C 3 **then**
          $S_r = S_r \cup \{v\}$
        end if
        $\bar{h}_{u_1,u_2} = (\bar{h}_{u_1} + \bar{h}_{u_2})/2$
      end for
      Choose the edge pair $(e_{u_1,v}, e_{u_2,v})$ with min $\bar{h}_{u_1,u_2}$
      $\bar{h}_v = \bar{h}_{u_1,u_2} + 1$
    end for
    if $S_r \neq \varnothing$ **then**
      Add node v in S_r with min $\bar{h}_v$ to S
    else
      Add node v in S' with min $\bar{h}_v$ to S
    end if
    // construct G_v: $\bar{h}_{u_1,u_2}$ is the min among all edge pairs to v
    ConstructDG(G, G_{u_1}, G_{u_2}, v);
  else
    Find $S'' \subseteq V - S$: $\forall v \in S''$, v has one edge $e_{u,v}$ from S
    if $S'' \neq \varnothing$ **then**
      for all node $v \in S''$ **do**
        $\bar{h}_v = \bar{h}_u + 1$
        Calculate n_v, the # of v's outgoing edges to $V - S$
      end for
      Add v to S with maximum n_v, break tie using $\bar{h}_v$
      ConstructDG(G, G_{u_1}, null, v);
    else
      return FAIL;
    end if
  end if
end while
return SUCCESS;

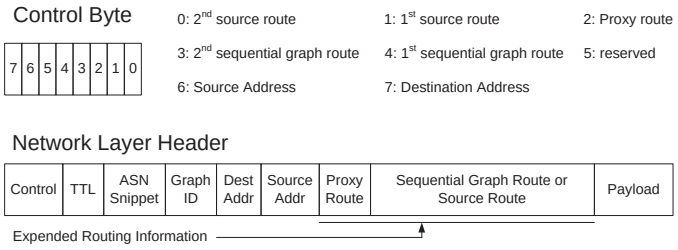

Fig. 8. The extension of the network layer header in WirelessHART to support sequential reliable downlink routing

use the source routing option field to store the ordered graph list; Second, the routing module is enhanced to support SRDR. When the packet arrives at the intermediate node, the routing module will retrieve the earliest graph ID in the graph list and verify if the current node is the sink of this specific graph. If it is, we remove this graph ID from the graph list and route this packet on the next earliest graph. This process continues until we reach the final destination or the routing fails. In the latter case, we will remove this graph ID and try the next earliest graph ID if it has the corresponding edges. Otherwise, alarm messages will be sent to the Network Manager and appropriate actions shall be taken.

Alg. 5 summarizes the framework of SRDR. In the algorithm, given the original graph G, we construct the reliable downlink

Alg 4 ConstructDG ($G(V, E)$, $G_{u_1}(V_{u_1}, E_{u_1})$, $G_{u_2}(V_{u_2}, E_{u_2})$, v)

Let S contain explored nodes in $G_v(V_v, E_v)$: $S = \{g, v, u_1, u_2\}$

// Construct G_v: Merging G_{u_1}, G_{u_2}, v, and edges among v, u_1, u_2
$G_v(V_v, E_v) = G_v(V_{u_1} \cup V_{u_2} \cup \{v\}, E_{u_1} \cup E_{u_2} \cup \{e_{u_1,v}, e_{u_2,v}, e_{u_1,u_2}, e_{u_2,u_1}\})$

while $S \neq V_v$ **do**
  if V_{AP} has two outgoing edges to S in G **then**
    $S = S \cup V_{AP}$
    break;
  end if
  for all node $i \in V_v - S$ **do**
    Sort i's outgoing edges to S in descending order of $\bar{h}_i$
  end for

  Find $S' \subseteq V_v - S$: $\forall v \in S'$, v has at least two edges to S
  if $S' \neq \varnothing$ **then**
    Add node i with min $\bar{h}_i$ to S
    Add first two edges from i to S to G_v if they don't exist
    Remove all other edges from E_v
  else
    Find $S'' \subseteq V_v - S$: $\forall v \in S''$, v has one edge to S
    if $S'' \neq \varnothing$ **then**
      Add i with min $\bar{h}_i$ to S
      Add the edge from i to S to G_v if it doesn't exist
    else
      return FAIL;
    end if
  end if
end while

for all node $i \in V_v - S$ **do**
  Remove i from V_v and corresponding edges from E_v
end for
return SUCCESS;

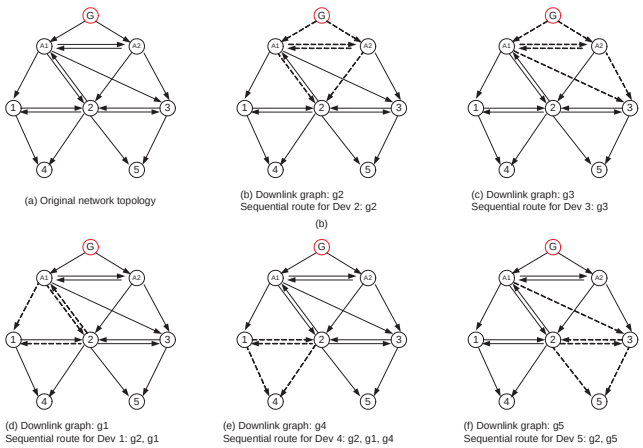

Fig. 9. Examples of the sequential reliable downlink routes

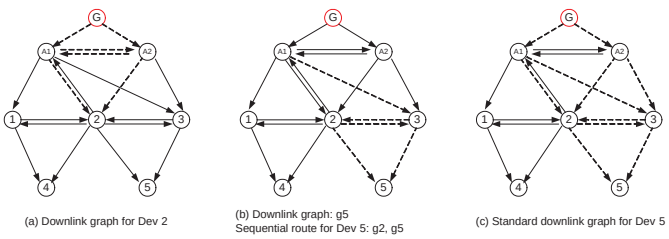

Fig. 10. Standard approach vs. Sequential reliable downlink routing (SRDR)

C1: v has at least two parents u_1, u_2, and they form a cycle.
C2: u_1 is u_2's parent in u_2's local downlink graph.
C3: u_2 (u_1) has at least one parent from the cycle in G_{u_1} (G_{u_2})

TABLE II

Three constraints in constructing scalable reliable downlink graphs

route (an ordered graph list) for each node in the network. For the Access Point, its downlink route contains only one local graph which consists of the Gateway g, itself and the edge between them. We maintain S, a set of nodes whose downlink routes have already been constructed (Line 1). We incrementally find an eligible node v in $V-S$ to construct its downlink route R_v where three constraints in Table II are applied and v has the minimum $\bar{h}_v$ as calculated in Lines 14-26. Constraint C1 is to find v's local downlink graph $g_v = (\{u_1 \cup u_2 \cup v\}, \{e_{u_1,u_2}, e_{u_2,u_1}, e_{u_1,v}, e_{u_2,v}\})$; If constraint C2 is satisfied, v's downlink route R_v can be simply derived as $R_v = R_{u_2} \rightarrow g_v$; Constraint C3 presents another way to construct the reliable downlink route for v if u_1 and u_2 are independent. If an extra edge e can be found from the cycle in G_{u_1} to u_2 or from the cycle in G_{u_2} to u_1, it will be added into g_v, and R_v can be derived as $R_{u_1} \rightarrow g_v$ or $R_{u_2} \rightarrow g_v$. If such an eligible node cannot be found, we will instead choose the node that has two parents from S with the minimum average latency to the Gateway (Line 18). If every node in $V-S$ has only one parent from S, the one with minimum average latency will be chosen (Line 28 - 40). Alg. 6 gives the details how we construct R_v.

Example V.1: Figure 9 illustrates an example for constructing the reliable downlink routes for devices in a WirelessHART network. Figure 9(a) gives the original topology of the network. We first include node 2 and node 3 into the explored node set S. The dotted lines in Figure 9(b) and Figure 9(c) show their local downlink graphs. When adding node 1 into S, as $A1$ and node 2 are already in S and they satisfy the constraints C1 $\wedge$ C2, R_1 is derived as $g_2 \rightarrow g_1$. We have the similar operations when adding node 4 into S and $R_4 = g_2 \rightarrow g_1 \rightarrow g_4$. However, when we add node 5 into S, node 2 and node 3 are independent. As we have a link between $A1$ and node 3, constraints C1 $\wedge$ C3 are satisfied. The dotted links in Figure 9(f) shows g_5, and the downlink route of node 5, R_5 is $g_2 \rightarrow g_5$.

The next example compares the standard approach in WirelessHART with sequential reliable downlink routing (SRDR).

Example V.2: Figure 10 compares SRDR with the standard approach in WirelessHART. The downlink graphs for node 2 under both approaches are the same (Figure 10(a)). The downlink route for node 5 in our approach is $R_5 = g_2 \rightarrow g_5$, and g_5 is shown in Figure 10(b). In SRDR, the downlink routing from the Gateway to node 5 can leverage the local routing graph in intermediate node (node 2) while only a local graph in node 5 is needed. However, the standard approach has to construct a completely new graph from the Gateway to node 5 which is shown in Figure 10(c). Comparing Figure 10(b) and Figure 10(c), the standard approach requires 3 extra links to achieve the reliable downlink routing. This overhead will increase dramatically when the destination is far away from the Gateway.

Optimization: In the basic SRDR, the routing is performed strictly according to the sequence in the ordered graph list. However, as each node can keep graph information to multiple destinations, we can take advantage of the "shortcut" to further improve the network latency. We call this approach SRDR-OPT. When a packet arrives at an intermediate node i, instead of using the earliest graph ID, SRDR-OPT searches the ordered graph list *backward* and finds the first graph ID that is stored in its routing table. The packet then will take the "shortcut" and be forwarded on this graph. If this forwarding is successful, at the destination of this selected graph, all the preceding graph IDs in the ordered graph list including the current ID will be removed. Otherwise, node i will choose the next available graph ID *backward* in the ordered graph list and repeat this process. The following example shows the advantage of SRDR-OPT.

Alg 5 Constructing Sequential Reliable Downlink Routes

```
1: Let S be the set of explored nodes with downlink route constructed
2: Initially S = g ∪ V_AP
3: Initially for each AP i in S, set G_i = ({g ∪ i}, {e_{g,i}}) and R_i = G_i
4:
5: while S ≠ V do
6:    Find S' ⊆ V − S: ∀v ∈ S', v has at least two edges from S
7:    // S_r is the reliable node set in S', initially S_r = ∅
8:    if S' ≠ ∅ then
9:       for all node v ∈ S' do
10:         for all edge pair (e_{u1,v}, e_{u2,v}) from S do
11:            h̄_{u1,u2} = (h̄_{u1} + h̄_{u2})/2
12:         end for
13:         Find P_v, set of edge pairs of v satisfying C1 ∧ (C2 ∪ C3)
14:         if P_v ≠ ∅ then
15:            S_r = S_r ∪ {v}
16:            Choose (e_{u1,v}, e_{u2,v}) from P_v with min h̄_{u1,u2}
17:         else
18:            Choose (e_{u1,v}, e_{u2,v}) from S with min h̄_{u1,u2}
19:         end if
20:         h̄_v = h̄_{u1,u2} + 1
21:      end for
22:      if S_r ≠ ∅ then
23:         Add v in S_r with min h̄_v to S
24:      else
25:         Add v in S' with min h̄_v to S
26:      end if
27:      ConstructDG(G, u1, u2, v);
28:   else
29:      Find S'' ⊆ V − S and ∀v ∈ S'', v has one edge e_{u,v} from S
30:      if S'' ≠ ∅ then
31:         for all node v ∈ S'' do
32:            h̄_v = h̄_u + 1
33:         end for
34:         Add v to S with min h̄_v
35:         G_v = ({u ∪ v}, {e_{u,v}})
36:         R_v = R_u → G_v
37:      else
38:         return FAIL;
39:      end if
40:   end if
41: end while
42: return SUCCESS;
```

Example V.3: In Figure 11, we are routing packets from node s to node 4 and R_4 is $g_2 \rightarrow g_3 \rightarrow g_4$. In node 2, it contains the routing information for both graph g_3 and g_4. It contains edges $2 \rightarrow 3$ and $2 \rightarrow 1$ on g_3 and edges $2 \rightarrow 4$ and $2 \rightarrow 3$ on g_4. When a packet arrives at node 2 with an ordered graph list $g_3 \rightarrow g_4$ in the network layer header (g_2 is removed at node 2), node 2 will take the "shortcut" and try to forward the packet on graph g_4 to node 4. Only if both edges on graph g_4 are broken, node 2 will forward the packet on graph g_3 and try the edge $2 \rightarrow 1$ instead. Under this worse-case scenario, the packet will forwarded to node 4 through $s \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 4$.

I. Maintaining Reliable Routing Graphs with Network Dynamics

The algorithms presented in the previous subsections construct the reliable routing graphs in ideal scenarios where network

Alg 6 ConstructDG (G, u_1, u_2, v)

Let E_δ be the set of edges among u_1, u_2 and v
if u_1, u_2 satisfy C1 $\wedge$ C2 **then**
  $G_v = G(\{u_1, u_2, v\}, E_\delta)$
  if u_1 is u_2's parent in G_{u_2} **then**
    $R_v = R_{u_2} \rightarrow G_v$
  else
    $R_v = R_{u_1} \rightarrow G_v$
  end if
else if u_1, u_2 satisfy C1 $\wedge$ C3 **then**
  if u_1 has an edge e from u_2's parents in G_{u_2} **then**
    $G_v = G(\{u_1, u_2, v\}, E_\delta \cup e)$
    $R_v = R_{u_2} \rightarrow G_v$
  end if
  if u_2 has an edge e from u_1's parents in $G_{u_1} \wedge (h_{u_2} < h_{u_1})$ **then**
    $G_v = G(\{u_1, u_2, v\}, E_\delta \cup e)$
    $R_v = R_{u_1} \rightarrow G_v$
  end if
else
  if e_{u_1,u_2} and e_{u_2,u_1} both exist **then**
    $G_v = G(\{u_1, u_2, v\}, E_\delta)$
    $R_v = (h_{u_1} < h_{u_2})\,?\,R_{u_1} \rightarrow G_v : R_{u_2} \rightarrow G_v$
  else if there is neither e_{u_1,u_2} nor e_{u_2,u_1} **then**
    $G_v = (h_{u_1} < h_{u_2})\,?\,G(\{u_1, v\}, \{e_{u_1,v}\}) : G(\{u_2, v\}, \{e_{u_2,v}\})$
    $R_v = (h_{u_1} < h_{u_2})\,?\,R_{u_1} \rightarrow G_v : R_{u_2} \rightarrow G_v$
  else if e_{u_1,u_2} exists **then**
    $G_v = G(\{u_1, u_2, v\}, E_\delta)$
    $R_v = R_{u_1} \rightarrow G_v$
  else
    $G_v = G(\{u_1, u_2, v\}, E_\delta)$
    $R_v = R_{u_2} \rightarrow G_v$
  end if
end if

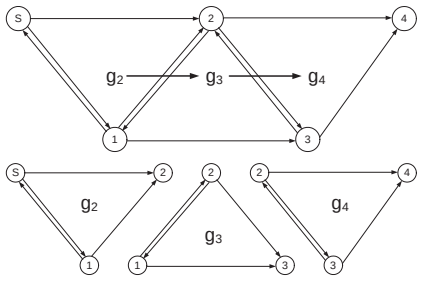

Fig. 11. An example of the SRDR optimization

devices work properly after joining the network. Although the industrial wireless mesh is usually quite stable after deployment, network devices may experience various failures and need to be reset. Wireless links can also be blocked by interference and become temporarily or permanently unavailable. All these scenarios require the Network Manager to recover the routing graphs to maintain the reliability requirements. Furthermore, corresponding adjustments on the communication schedules are also necessary along with these routing graph modifications.

In WirelessHART networks, network abnormalities and statistics are reported to the Network Manager through a set of network maintenance commands. These commands are summarized in Table III. Command 779 summaries the communication statistics of a specific device; Command 780 and 787 report the signal strengths of a device's neighbors; Command 788, 789 and 790 are triggered once a path failure or routing failure is detected in the network. These commands are carried in normal messages and published to the Network Manager. Based on this information, the Network Manager will update the network topology, adjust the routing graphs and communication schedules if necessary to reach a good balance between the reliability and recovery cost.

Our current heuristics to recover G_B consists of two steps. We first find $G'_B(V'_B, E'_B)$, the sub graph of G_B where all nodes in V'_B are reliable after the topology changes. In the second step, we replace G_B with G'_B and repeat Alg. 1 to incrementally add nodes to G_B. This process repeats until either all the nodes are included in G_B or disconnected nodes are identified. The mechanism to reconstruct G_U is similar to that of G_B. Designing efficient algorithms to reconstruct G_v to each node v is more challenging and will be addressed in our future works.

VI. Communication Schedule and Channel Management

Typical wireless industrial process control applications take the approach that devices specify their requirements in communication bandwidth and the Network Manager allocates necessary resources such as timeslots, to maintain the periodic sensing-control loop between the Network Manger and devices. In the sensing phase, the devices publish their process data to the Gateway through the uplink graph based on their specific sample rates; In the control phase, the Network Manager generates control messages and sends them back to each individual device on its downlink graph. The Network Manager maintains a global communication schedule for transmitting these process and control data and distributes the sub-schedule to each effected device.

The construction of the communication schedule is subject to several practical constraints in WirelessHART networks:

- The maximum number of concurrent active channels is 16.
- Each device can only be scheduled to TX/RX once in a slot.
- Multiple devices can compete to transmit to the same device simultaneously (in shared timeslot).
- On a multi-hop path, early hops must be scheduled first.
- The practical sample rates are defined as 2^n sec ($-2 \leq n \leq 9$) from 250 ms (2^{-2} sec) to 8 min and 32 sec (2^9 sec).

Our design philosophy for constructing the communication schedule is to spread out the channel usage in the network as much as possible and to apply the Fastest Sample Rate First policy (*FSRF*) to schedule the devices' periodic publishing and control data.

We use the concept of superframe to group a sequence of consecutive timeslots and represent the communication pattern for a given sample rate. We define two types of superframes: data superframe and management superframe. The data superframe is used to support data transmissions between the devices and the Gateway while the management superframe is used to support exchanging network management messages. The number of data superframes is decided by the number of different sample rates existing in the network. Notice that there can be multiple devices having the same sample rate, thus a data superframe will represent the periodic behavior of multiple devices.

We maintain a global matrix $\mathcal{M}$ to keep track of the current slot/channel usage in the network. Each entry in the matrix, $\mathcal{M}_{i,j}$ represents the slot usage at timeslot i on channel j, and it has four types: unused, exclusive, shared and reserved. An unused entry can be allocated to any pair of devices if there is no communication conflict; An exclusive entry is one occupied by two devices for dedicated communication; Reserved entries are managed by the Gateway or the Network Manager for maintenance purposes; Finally a shared entry allows multiple devices to compete for transmitting to the same device simultaneously. For instance, in our system, we allow 5 simultaneous transmissions on a shared timeslot. We also maintain several other important data structures for constructing the communication schedule. They include one

Command	Functionality
Command 779	Report device communication statistics
Command 780	Report neighbor health list
Command 787	Report neighbor signal levels
Command 788	Path down alarm
Command 789	Source route failure alarm
Command 790	Graph route failure alarm

TABLE III

Summary of network maintenance commands

data superframe $\mathcal{F}_i$ per sample rate r_i and a global management superframe $\mathcal{F}_m$. Here we use l_i to denote the length of $\mathcal{F}_i$. For each node v, we maintain a schedule $\mathcal{S}_v$ to record its own slot/channel usage. The length of $\mathcal{M}$ and $\mathcal{S}_v$ are both equal to the maximum length among the existing superframes. These schedules will be distributed to the devices to achieve end-to-end real-time communication.

Alg 7 Constructing Data Communication Schedule

```
1: Sort device sample rates in ascending order: r_1 < r_2 < ... < r_k.
2: Identify the set of nodes with each sample rate: N_1, N_2, ..., N_k.
3: Initialize the schedule for each node as ∅
4:
5: for all r_i from r_1 to r_k do
6:    Generate the data superframe F_i
7:    for all node v ∈ N_i do
8:       // Schedule primary and retry links for publishing data
9:       ScheduleLinks(v, g, G_U, F_i, 0, Exclusive);
10:      ScheduleLinks(v, g, G_U, F_i, l_i/4, Shared);
11:
12:      // Schedule primary and retry links for control data
13:      ScheduleLinks(g, v, G_v, F_i, l_i/2, Exclusive);
14:      ScheduleLinks(g, v, G_v, F_i, 3l_i/4, Shared);
15:
16:      if all link assignments are successfully then
17:         continue;
18:      else
19:         // Defer bandwidth request from node v
20:         return FAIL;
21:      end if
22:   end for
23: end for
24: return SUCCESS;
```

We present the framework of constructing the data communication schedule in Alg. 7. The construction of the management schedule follows the same approach and is omitted here. In the algorithm, we apply the (*FSRF*) policy in scheduling data transmissions. The construction is based on the reliable graphs we introduced in Section V. For each device v, in its sensing phase, it allocates the primary and retry links along the uplink graph G_U to the Gateway (Line 9 - 10); In the control phase, the Network Manager sends the control messages back and allocates the primary and retry links along the downlink graph G_v (Line 13 - 14). The ScheduleLinks(u, v, G, $\mathcal{F}$, t, o) function is described in Alg. 8. It allocates every link on the paths from u to v on graph G one by one in a depth-first manner. It allocates the earliest available timeslot t_i from t for each link and updates $\mathcal{M}$, $\mathcal{F}$ and each effected node's schedule accordingly. If we cannot find a slot in $[t, l_{\mathcal{F}}]$ to accommodate all the allocations, the Network Manager will defer the bandwidth request from the corresponding device until enough bandwidth resources are available (Line 19 - 20 in Alg. 7).

Notice that a device v is typically multi-hop away from the Gateway, and it has multiple paths to the Gateway due to the property of reliable graph routing. However, if we allocate the required communication bandwidth for device v on each hop along all its paths to the Gateway, most of the allocated links will be wasted because in each end-to-end transmission, only one path will be picked. This will severely degrade the schedulability of the network schedule. To address this problem, as shown in Alg. 8 (Line 17 - 33), when the device has two successors to forward the messages, we reduce the transmission rate between v and each of its successors to half of the original sample rate, and schedule the links on the corresponding superframe $\mathcal{F}'(l_{\mathcal{F}'} = 2 \cdot l_{\mathcal{F}})$. We determine the timeslot offset of these links in $\mathcal{F}'$ to make sure that their combinations will form a communication pattern the same as the original sample rate.

Alg 8 ScheduleLinks(u, v, G, $\mathcal{F}$, t, o)

```
1: // u and v are the source and destination of the communication
2: // G is the routing graph and F is the superframe
3: // t is the earliest slot to be allocated and o is the link option
4:
5: Identify data superframe F' with l_F' = 2l_F
6: for all node i ∈ Successor(u) do
7:    Identify the schedule S_u and S_i for node u and i
8:    if i is the only successor of u then
9:       Identify the earliest slot from t with a channel c to:
10:      Allocate entries M_{k·l_F+t_i,c}(k = 0, 1, ...) on M
11:      Allocate the slots k · l_F + t_i on S_u and S_i
12:      Allocate slot t_i on F
13:
14:      if All allocations are successful then
15:         ScheduleLink(i, v, G, F, t_i, o);
16:      end if
17:   else
18:      if i is the first successor then
19:         Identify the earliest slot from t with a channel c to:
20:         Allocate entries M_{k·l_F'+t_i,c} on M
21:         Allocate slots k · l_F' + t_i on S_u and S_i
22:         Allocate slot t_i on F'
23:      else
24:         Identify the earliest slot from t in M with a channel c to:
25:         Allocate entries M_{k·l_F'+l_F+t_i,c} on M
26:         Allocate slots k · l_F' + l_F + t_i on S_u and S_i
27:         Allocate slot l_F + t_i on F'
28:      end if
29:
30:      if All allocations are successful then
31:         ScheduleLink(i, v, G, F', t_i, o);
32:      end if
33:   end if
34:   if No feasible allocations available then
35:      return FAIL;
36:   end if
37: end for
38: return SUCCESS;
```

VII. System Implementation

We have built a complete WirelessHART communication system to verify the correctness and efficiency of our network management techniques. We are deploying the system in a large-scale manufacturing factory to collect sensor data from testing devices, and achieve factory automation. Figure 12 depicts the abstract architecture of our system which has five major components: the WirelessHART mesh network, Gateway, Access Point, Network Manager and Host applications. These components in our system are shown in Figure 13, and their design details will be presented in the following sections.

A. WirelessHART Mesh Network

Our WirelessHART mesh network is formed by two types of devices. Rosemount [24] sensors and the Freescale devices

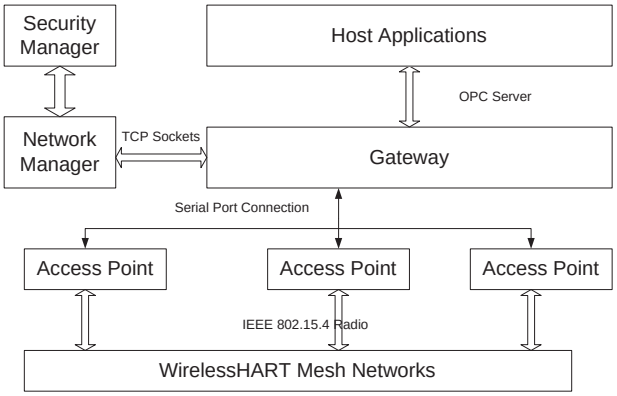

Fig. 12. Architecture of the complete WirelessHART communication system

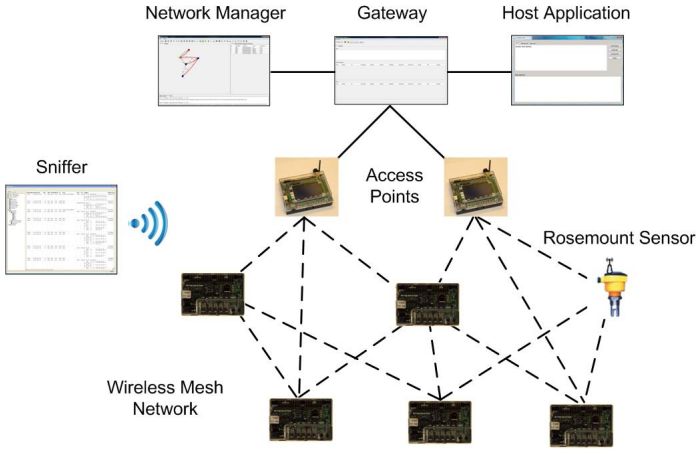

Fig. 13. The major components in the system

with the stack that developed by ourselves [3]. All these devices comply to the WirelessHART standard, thus have no problem to interoperate with each other. They join into the network through the standardized procedure [25]. The Network Manager organizes these devices into a multi-hop reliable mesh and configures them with corresponding routes and communication schedules. Once the devices are correctly configured, they begin to exchange management and data messages with the Network Manager and Gateway.

B. Gateway Design

The Gateway works as a server responsible for communicating with the Network Manager, processing the requests from the Host applications, collecting and caching data from all devices in the network. Its architecture is illustrated in Figure 14 and has the following major components:

Physical Connections: The Gateway provides a serial port connection to each attached Access Point. The Gateway talks with the Network Manager through a socket connection for message exchange. It also provides one or more Host Interfaces to backbone networks (e.g., the plant automation network) to receive the queries and send the responses back.

Real-time Database and Query Processor: The core parts of the Gateway are a real-time database and a query processor. The database provides data caching for burst mode, event notification, and common HART command responses. The query processor processes the queries from Host applications. If the requested data are already cached and still valid, they are returned immediately to the Host applications. This reduces network traffic

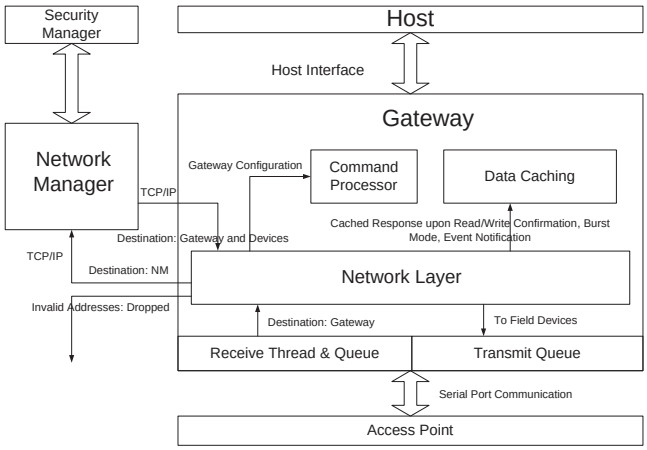

Fig. 14. The architecture of the Gateway

and improves the Host application's responsiveness. Otherwise, the query processor will generate the request messages and send them to the corresponding devices. The return response data are cached in the Gateway and sent back to the Host applications.

Time Source: The Gateway maintains a time source module for maintaining network-wide time synchronization. It will notify all the devices in the network and let them synchronize with the Gateway through the approaches discussed in Section IV. In our system, the actual time source is a designated Access Point instead of the Gateway. This Access Point will periodically update the accurate time to the Gateway and Network Manager.

C. Access Point Design

The Access Point is a bridge between the mesh network and the Gateway. There could be multiple Access Points attached to the Gateway providing load balancing and reliable graph routing. Each Access Point goes through the same join procedure as a normal device to authenticate itself and establish a secured connection with the Network Manager. As shown in Fig. 15, The communication stack on the Access Point is extended from that of the device by adding an extra UART module. The messages received from the mesh will be forwarded to the UART module and sent to the Gateway. In the other direction, the messages from Gateway/Network Manager will be sent through the serial port and put into the network layer queue in the Access Point.

If a network includes multiple Access Points, then they must be synchronized. In our design, except the designated time source, all other Access Points will be instructed by the Network Manager to scan the physical channels the same way as when a normal node joins the network. A normal node will send out join request message to a neighbor after synchronization. These Access Point directly send the join request to the Network Manager through the Gateway. Afterwards the Network Manager configures them just like it configures the original time source.

D. Network Manager Design

The core of a WirelessHART mesh network is the Network Manager. It is responsible for authenticating the devices, forming the network, allocating network resources and scheduling process data transmissions. We have described the detailed algorithmic issues in Section V and Section VI for generating routing graphs and constructing communication schedules. Here we describe our implementation of the Network Manager and how we integrate our network management solutions into it. Figure 16 shows

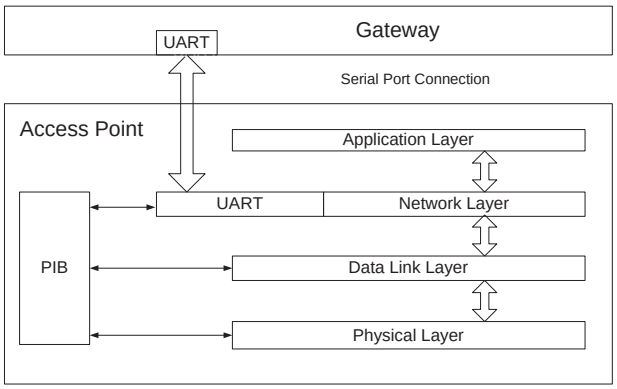

Fig. 15. The architecture of the Access Point

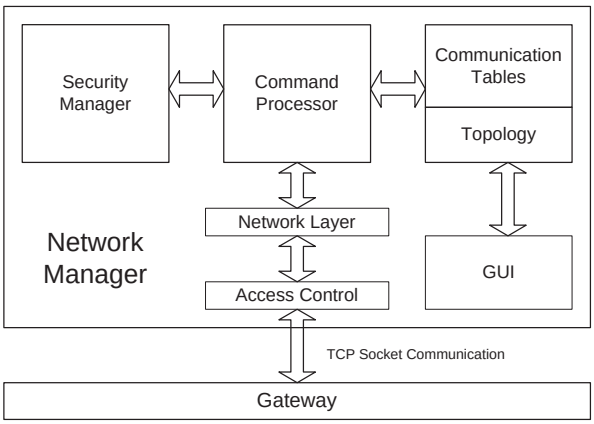

Fig. 16. The architecture of the Network Manager

the architecture of the Network Manager which has four major components:

Command Processor: The application layer of the WirelessHART standard is command-oriented. The WirelessHART devices and the Network Manager interact by exchanging command requests and command responses. The command processor in the Network Manager processes the commands from the devices, updates the network topology and triggers the algorithms if necessary to reconstruct the routing graphs and communication schedules.

Network Topology and Communication Tables: In the Network Manager, the network topology is maintained in a directed graph structure. All the algorithms for constructing routing graphs and allocating network resources are conducted in the graph and the results are maintained in a set of communication tables. Interested readers are referred to [3] for their details.

Security Manager and Access Control: WirelessHART is a secure wireless communication protocol and it provides encryption and authentication in both the data link layer and network layer. The main task of the security manager is to manage various key information for the devices. It takes charge of the device join authentication and updates the key information in the network periodically for protection purpose. The access control module maintains a list of pre-approved devices together with their valid join keys. Only the devices on the list can be admitted into the network by providing the correct join keys.

Visualizer: Our visualizer is implemented based on the JUNG library [26]. It provides the user a straight-forward way to observe the network topology, the routing graphs, the device communication schedules, and the exchanged messages. Any update on them will also be reflected in the visualizer in real-time. With the visualizer, users can identify problematic network topology and bottlenecks limiting network throughput and perform appropriate adjustments. [27] gives an example of a WirelessHART network with two Access Points and 50 field devices. It also shows the communication schedules which are generated based the proposed algorithms and each device's bandwidth usage.

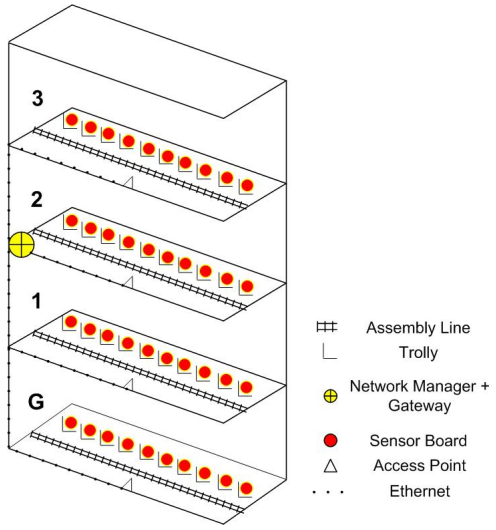

Fig. 17. The topology of the WirelessHART network under deployment

We note that our Network Manager design is not only for the WirelessHART communication systems. It is also a generic simulator for wireless mesh networks and allows the users to specify any network topology either through reading in a topology file or configuring it manually. It provides the user a platform to design their algorithms, exercise them on the specified topology and evaluate their performance.

E. System under Deployment

We are deploying our system in a manufacturing factory to help achieve factory automation. The factory has 4 floors, and each floor has around 20 trollies. Each trolly can carry up to 16 motherboards under test and each board is attached with a watchdog. All the watchdogs in a trolly are connected to a controller through I^2C bus, and they publish their sampling data (60 bytes) to the controller every one minute. Previously, the testers have to manually check each trolly and identify the malfunctioning boards. To achieve factory automation, we are integrating their testing equipments with our WirelessHART communication system. We attach our sensor board to each trolly and connect it to the I^2C controller. The samples from the watchdogs will be forwarded from the controller to our board and transmitted to the Access Point. To improve the network connectivity, we deploy one Access Point in each floor. The Gateway and Network Manager are installed on the third floor and all the Access Points are connected to them through ethernet. Fig. 17 shows the topology of the system under deployment. Once the system is set up, the tester can monitor the status of all motherboards under test simply through the Gateway. This will save a large amount of manpower, and speed up the testing period.

VIII. Performance Evaluation

This section summarizes the major results from our simulations to evaluate the performance of our algorithms. Our simulation

model and parameter settings are described in Section VIII-A. Section VIII-B compares our algorithms in constructing reliable routing graphs to traditional approaches. Section VIII-C evaluates the performance of our approach for constructing communication schedules. The results show that our approaches can achieve higher routing success rates, better end-to-end communication latency while incurring only modest configuration overheads on devices.

A. Simulation Model and Parameters

In the simulations, we assume open field, line-of-sight experimental scenarios. The simulation area is fixed at 450 m × 450 m and the default device communication distance is 100 meters with a 0 dBm transmitter. We assume that there is no edge between a pair of nodes if they are not in each other's communication range. Otherwise, an edge exists with an edge success probability p that is varied from 0.0 to 1.0. The size of the network is varied from 50 to 150 to evaluate the effect of network density on the algorithm's performance. We disable a given portion of links in the network to evaluate the reliability of the constructed routing graphs and this percentage is varied from 0% to 95%.

B. Performance of Reliable Routing Graphs

We conducted a series of experiments to evaluate the performance of the reliable broadcast graph G_B, reliable uplink graph G_U and reliable downlink graph G_v for each individual node v. Since essentially G_U is the reversed version of G_B, its performance is similar to that of G_B. For this reason, the experiment results of G_U are omitted here.

We compare our approach for constructing G_B with two baseline methods. The first method constructs a single broadcast tree using breadth-first search and the second method generates the max-reliable broadcast graph. In the latter method, when a node is chosen to be added to the broadcast graph, all its incoming edges from the current broadcast graph are also added. Different from this method, our approach only chooses the first two incoming edges of the chosen node with minimum latency, and thus achieve a good balance between the routing reliability and the configuration overhead. In this paper, the configuration overhead is defined as the average number of links to be configured per node. It is an important performance metric because wireless sensors' memory is limited and configuring large number of links in the network will severely hurt the schedulability of the communication schedule.

The first experiment compares the configuration overhead introduced by these three approaches. In the experiments, we vary the size of the network from 50 to 150 nodes and evaluate its impact. Figure 18 summarizes our results. As expected, we observe that the configuration overhead of the max-reliable approach is much higher than the other two and it increases linearly along with the increase of the network density. On the other hand, the overhead in our approach and the broadcast tree solution is much low and stable. The overhead in our approach is always below 2 links per node, and it is closer to the performance of the broadcast tree when the network density is low. This observation is mainly because when the network density is low, it is difficult for many nodes to find two parents in the network thus has only one link in the broadcast graph.

In the second experiment, we first construct the broadcast graphs based on these three approaches with 100 nodes in the network. We then gradually increase the percentage of failed links in the network from 0% to 95%. We measure the reliability of these three approaches and apply the recovery mechanisms we discussed in Section V-I on them. We compare their recovery overhead in terms of number of changed links. Figure 19 shows that along with the increased percentage of failed links in the network, the reliability of the broadcast tree drops quickly and when half of the links die, only around 25% nodes are reachable from the Gateway. Our approach performs much better. With the same percentage of failed links, around 55% of nodes are still connected. Among all three approaches, the max-reliable broadcast graph has the best performance as a tradeoff of its poor scalability and much higher configuration overhead. In figure 19, we also show a curve of the reachability for the broadcast graphs after the recovery. As the recovery mechanisms are all based on the same underlying network topology, all three approaches have the same reachability after reconstruction. This in turn verifies the correctness of our recovery mechanisms.

Figure 20 and Figure 21 compare the recovery overhead among these approaches. Figure 20 shows the overhead to resume the connectivity of the broadcast graphs while Figure 21 further shows the overhead to recover their reliability properties. We observe from Figure 20 that the broadcast tree always has the heaviest recovery overhead while the max-reliable broadcast tree has the minimum because of its best reliability. The performance of our approach sits between them. However, Figure 21 shows that to recover the reliability property, our approach needs to add more links than the other two alternatives. The reason is the broadcast tree has no reliability requirement while the max-reliable approach has already added most of the links in the construction stage thus its recovery overhead is relatively smaller.

In the third experiment, we evaluate the performance of the two proposed approaches for constructing reliable downlink graphs, the standard approach as defined in WirelessHART standard RDG(standard) and the sequential reliable downlink routing approach (SRDR). we compare them with two baseline methods. The first method finds a single shortest path from the Gateway to the destination, while the second one constructs a two node-disjoint path and can tolerate one link or node failure. Figure 22 summarizes the comparison of the routing reliability among these four approaches. It clearly shows that the single path approach always has the worst performance. On the other hand, RDG(standard) maintains the best reliability and always outperforms the two node-disjoint path method more than 30%. SRDR is around 8% worse than RDG(standard) in routing reliability. This is because the downlink graphs constructed under RDG(standard) have more redundant links. As a tradeoff, as shown in Figure 23 and Figure 24, RDG(standard) introduces a much higher configuration overhead. The average number of nodes in the constructed graphs is 2 times and 1.2 times larger than that of the single shortest path approach and two node-disjoint path approach respectively. Furthermore, as each node under RDG(standard) has two outgoing edges, the average number of links in the constructed graphs is even higher. As shown in Fig. 24, it is around 5.5 times and 2.8 times larger than that of the single shortest path approach and two node-disjoint path approach respectively. However, SRDR only introduces very limited configuration overhead because it only constructs local graphs and these local graphs can be further reused for assembling the downlink routes to different destinations. Its average number of nodes is the lowest among all the four approaches and its average number of links is only slightly higher that of the single shortest path approach and around 33% lower than the two node-disjoint path approach. In sum, SRDR achieves a good

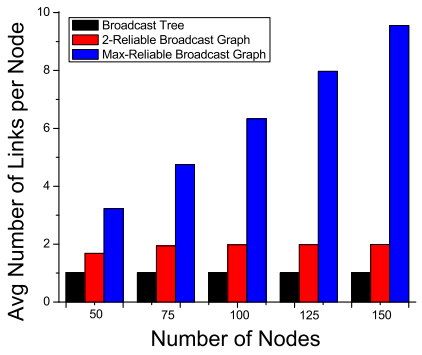

Fig. 18. Configuration overhead in broadcast graphs

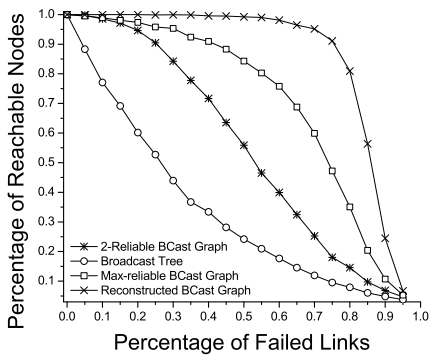

Fig. 19. Reachability in broadcast graphs

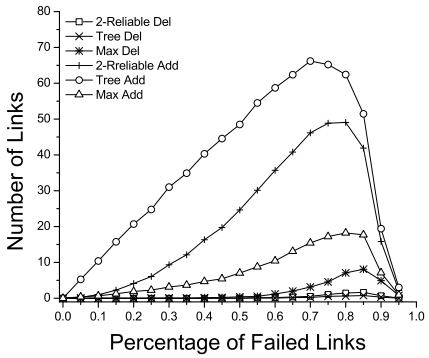

Fig. 20. Recovery overhead to regain connectivity

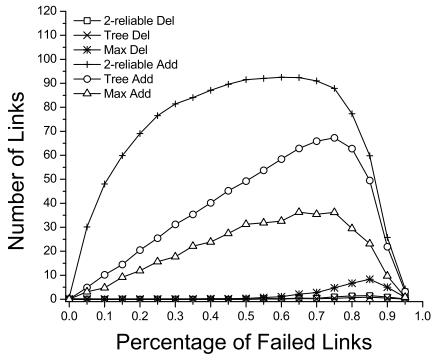

Fig. 21. Recovery overhead to regain reliability

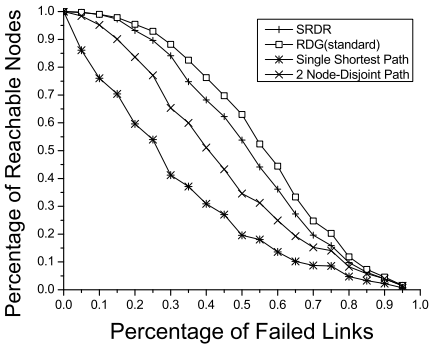

Fig. 22. Reachability in downlink graph

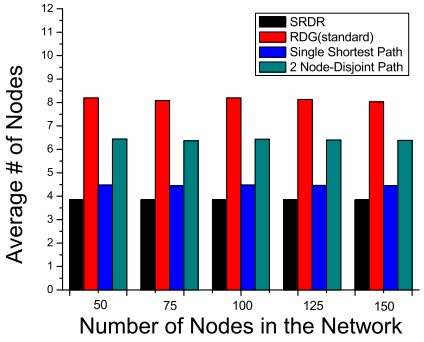

Fig. 23. Average # of nodes per downlink graph

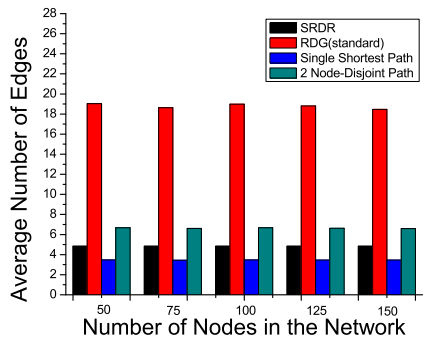

Fig. 24. Average # of edges per downlink graph

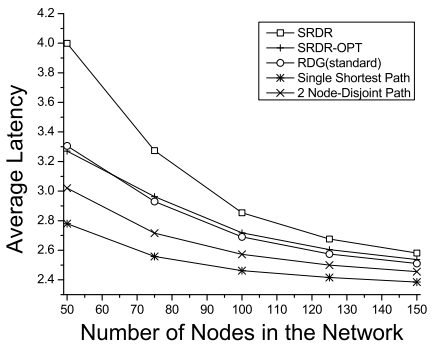

Fig. 25. Average latency vs. Network size

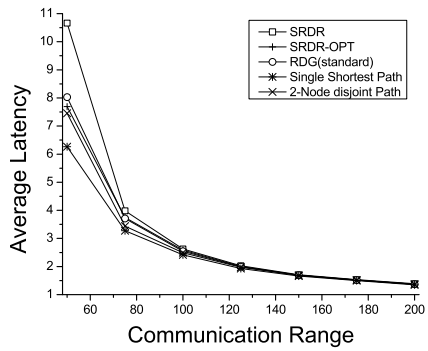

Fig. 26. Average latency vs. Communication range

balance between high routing reliability and low configuration overhead.

We also evaluate the performance of the optimization mechanism SRDR-OPT which is proposed in Section V-H, and measure its improvement on average latency in two different scenarios. In the first scenario, we fix the devices' communication range at 100m and increase the number of nodes in the network from 50 to 150. The results is shown in Fig. 25. We observe that SRDR has a much higher average latency compared with RDG(standard). This is because when constraint C2 is satisfied, SRDR chooses the node with larger latency as its parent in constructing downlink graph while RDG(standard) take both and its latency is calculated as their average plus one. The performance of SRDR-OPT is similar to RDG(standard) because the shortcuts are taken in the optimization. Obviously, the single shortest path approach always has the lowest latency. In the second scenario, we fix the number of nodes in the network at 150 and vary the communication range of the devices from 50m to 200m. As shown in Fig. 26, the average latencies of all the four approaches decrease with the increase of the communication range, and consistent with the observations in the first scenario, SRDR has a great improvement on the average latency when the optimization mechanism is applied.

C. Construction of Communication Schedules

Our approach for constructing the communication schedule has two unique features. First, we split the traffic from a device among all its successors by reducing the bandwidth requirement on each successor. The communication schedules on the successors are carefully designed so that their combination has the same patten as the original device. Second, we use the concept of shared timeslot to allow multiple devices to compete for communicating with the same device simultaneously. This is especially useful for the links that are allocated for retry purpose and it can significantly improve the network throughput.

In this section, we evaluate the performance of these two features by comparing our approach with three baseline methods. The basic methods either lack one of the features or both of them. For simplicity, we only show our experimental results on scheduling process data from devices to the Gateway on the

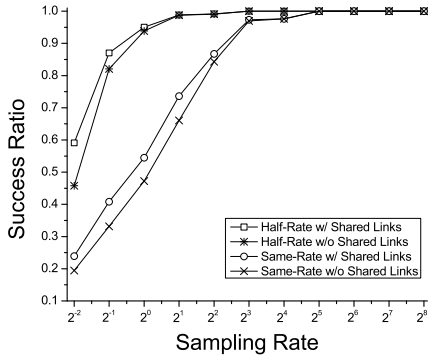

Fig. 27. Success ratio vs. Sample rate

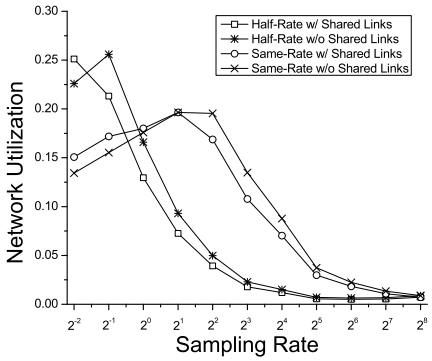

Fig. 28. Network utilization vs. Sample rate

uplink graph. Scheduling control data on the other direction is similar, and thus is omitted here. Two performance metrics are defined for this experiment. The first metric is the scheduling success ratio which measures the percentage of nodes that can successfully allocate the required bandwidth along its paths to the Gateway; The second metric is the network utilization which measures the percentage of entries in matrix $\mathcal{M}$ that are already allocated for communication. Our results are summarized in Figure 27 and Figure 28 respectively.

In Figure 27, we compare the scheduling success ratio by deploying 50 nodes in the network and varying the device sample rate from 250 ms to 4 min and 16 sec (each device has the same sample rate). We observe that by halving the bandwidth requirement on a device's successors (if it has two successors), the success ratio can be greatly improved. The improvement is more than 25% when the sample rate is 2 sec and is even higher when the sampling is faster. Figure 27 also shows that by applying the shared timeslot, the success ratio can be increased by 5% and this improvement is consistently shown in our experiment results until the sample rate is low enough that the scheduling success ratio approaches 100%. Figure 28 shows that when the approaches have a similar scheduling success ratio, our approach has a much lower network utilization, and this will further help include more devices into the network. When the sample rate is fast, our approach has a higher network utilization because in these scenarios, the success ratio for other approaches is so poor that a very limited number of devices can successfully allocate their required bandwidth along its path to the Gateway.

IX. Conclusions and Future Work

In this paper, we study the problem of how to achieve reliable and real-time communication in industrial wireless mesh networks. Taking WirelessHART network as an example, we abstract the reliability requirements in typical wireless industrial process control applications and present the algorithms for constructing three types of reliable routing graphs for different communication purposes. Based on these routing graphs, we describe how we construct the communication schedule in the network and highlight our approach's unique features. We present the architecture of a complete WirelessHART communication system that we have built and we have performed extensive simulations to evaluate the performance of our algorithms.

In ongoing and future work, we are deploying our system in a large-scale manufacturing factory, so that we can evaluate the performance of our network management techniques in real industrial environments. We shall continue to look for more efficient approaches for constructing routing graphs and communication schedules to maximize the power saving in WirelessHART networks, and study their corresponding recovery mechanisms.

References

[1] Andreas Willig, "Recent and emerging topics in wireless industrial communications: A selection," *IEEE Trans. on Industrial Informatics*, 2007.

[2] Dick Caro, *Wireless Networks for Industrial Automation*, ISA Press, 2004.

[3] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "WirelessHART: Applying wireless technology in real-time industrial process control," in *RTAS*, 2008.

[4] Rajeev Alur, Alessandro D'Innocenzo, Karl H. Johansson, George J. Pappas, and Gera Weiss, "Modeling and analysis of multi-hop control networks," in *RTAS*, 2009.

[5] Gera Weiss, Rajeev Alur, Alf J. Isaksson, and Karl H. Johansson, "Scalable scheduling algorithms for wireless networked control systems," in *CASE*, 2009.

[6] Joonas Pesonen, Haibo Zhang, Pablo Soldati, and Mikael Johansson, "Methodology and tools for controller-networking codesign in WirelessHART," in *ETFA*, 2009.

[7] Shahid Raza, Adriaan Slabbert, Thiemo Voigt, and Krister Landernäs, "Security considerations for the wireless hart protocol," in *ETFA*, 2009.

[8] Gabriella Fiore, Valeria Ercoli, Alf J. Isaksson, Krister Landernäs, and Maria Domenica Di Benedetto, "Multihop multi-channel scheduling for wireless control in WirelessHART networks," in *ETFA*, 2009.

[9] "ISA," `http://www.isa.org/`.

[10] "HART communication," `http://www.hartcomm.org`.

[11] "ZigBee Alliance," `http://www.zigbee.org`.

[12] "WirelessHART," `http://www.hartcomm.org/protocol/wihart/wireless_technology.html`.

[13] Abusayeed Saifulah, Chenyang Lu, You Xu, and Yixin Chen, "Real-time scheduling for WirelessHART networks," in *RTSS*, 2010.

[14] Pablo Soldati, Haibo Zhang, and Mikael Johansson, "Deadline-constrained transmission scheduling and data evacuation in wirelesshart networks," in *Technical Report TRITA-EE 2008:060*, 2008.

[15] Haibo Zhang, Pablo Soldati, and Mikael Johansson, "Optimal link scheduling and channel assignment for convergecast in linear wirelesshart networks," in *Technical Report TRITA-EE 2009:018*, 2009.

[16] "Bluetooth," `www.bluetooth.com/bluetooth`.

[17] "IEEE 802.15.4 WPAN Task Group," `www.ieee802.org/15/pub/TG4.html`.

[18] Stephen Mueller, Rosep. Tsang, and Dipak Ghosal, "Multipath routing in mobile ad hoc networks: Issues and challenges," *Performance Tools and Applications to Networked Systems*, 2004.

[19] Sasan Adibic Mohammed Tariquea, Kemal E. Tepeb and Shervin Erfanib, "Survey of multipath routing protocols for mobile ad hoc networks," *Journal of Network and Computer Applications*, vol. 32, no. 6, 2009.

[20] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 4, 2001.

[21] S.J.Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *ICC*, 2001.

[22] Mahesh K. Marina and Samir R. Das, "On-demand multipath distance vector routing in ad hoc networks," in *ICNP*, 2001.

[23] Z. Ye, S.V. Krishnamurthy, and S.K. Tripathi, "A framework for reliable routing in mobile ad hoc networks," in *INFOCOM*, 2003.

[24] "Rosemount," `http://www.emersonprocess.com/Rosemount/`.

[25] S. Han, J. Song, X. Zhu, A. K. Mok, D. Chen, M. Nixon, W. Pratt, and V. Gondhalekar, "Wi-HTest: testing suite for diagnosing WirelessHART devices and networks," in *RTAS*, 2009.

[26] "Java Universal Network/Graph Framework," `jung.sourceforge.net/`.

[27] "An example of network manager visualizer," `www.cs.utexas.edu/~shan/WH-example.pdf`.