No way back? An SDN protocol for directed IoT networks

Renan Cerqueira Afonso Alves
University of São Paulo
São Paulo, Brazil
renanalves@usp.br

Cintia Borges Margi
University of São Paulo
São Paulo, Brazil
cintia@usp.br

Fernando A. Kuipers
Delft University of Technology
Delft, The Netherlands
f.a.kuipers@tudelft.nl

***Abstract*—The Internet of Things (IoT) has and will continue to permeate many aspects of everyday life. However, IoT devices often use wireless communication to transfer their data, which may experience ambient noise and multipath fading. This, together with the wide heterogeneity in types of devices, leads to the emergence of unidirectional links in IoT networks. Surprisingly, many routing protocols for wireless networks either do not account for such links or employ radical mechanisms, like blacklisting. In this paper, we leverage the features of Software-Defined Networking (SDN) to develop a network discovery algorithm that is able to cope with unidirectional links, while containing the control overhead. We provide a proof-of-concept implementation of an SDN protocol for constrained devices that uses our algorithm and perform experiments. The experiment results show that our solution is scalable and performs well both for unidirectional links as well as for fully bidirectional networks.**

***Index Terms*—Software-Defined Networking, Unidirectional Links, Wireless Sensor Networks**

I. Introduction

The Internet of Things (IoT) is quickly expanding into a wide spectrum of human activities, from agriculture to manufacturing and smart cities, with industry likely to invest billions of dollars in IoT in the next few years [8]. The high expectations for IoT have spurred a lot of research from the academic community, for example on efficient medium access schemes, middleware, security, and routing protocols [14].

When it comes to the design of communication protocols for IoT networks, researchers usually assume the communication link between devices is bidirectional. However, unidirectional links are widely present in wireless networks and cannot be ignored. Even in homogeneous setups, links may become unidirectional due to non-isotropic antennas, varying ambient noise, and multipath fading [22]. Moreover, an IoT network tends to be composed of diverse devices with different energy resources and radio power, further increasing the occurrence of unidirectional links. For example, a gateway might have stronger transmission power than the nodes it reaches.

Regardless of the cause of unidirectionality, dealing with unidirectional links requires dedicated procedures. If a link is bidirectional, receiving a message is sufficient to know the receiver is also able to reach the sender. On the other hand, knowledge about the presence of a unidirectional link must be advertised back to the sender through an alternative path.

Consider the example network in Figure 1. All links surrounding node 2 are bidirectional, therefore any message received from nodes 1, 3, or 5 is enough to establish communication between them and node 2. Differently, unidirectional links have to be advertised to be used. The directed link from node 6 to node 3, can only be used if node 3 advertises the link's existence through node 7. In this case, there is a short detour to relay the information, however the shortest path to advertise the unidirectional link from node 4 to 7 is four hops long ($7 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4$).

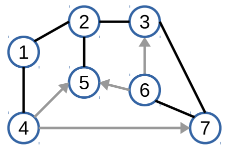


Fig. 1. Network with unidirectional links. How can node 4 know it reaches 7?

The main challenge is finding a reverse path corresponding to unidirectional links in an efficient manner. As illustrated by the example, reverse paths could potentially be long.

An alternative approach is let a central entity decide whether the unidirectional links should be used or not. For example, all nodes in Figure 1 could acquire their inbound neighborhood (therefore finding a reverse path is not necessary) and send it to node 1, which acts as the central entity. Subsequently, node 1 could calculate and distribute the routes to the other nodes.

Centralization of route decision-making matches the Software-Defined Networking (SDN) paradigm, a technology with growing usage in wired networks. A key component of SDN is the southbound protocol, which specifies the transactions between a logically centralized controller and the SDN-enabled nodes. OpenFlow [18] is an example of a popular southbound protocol. Its specification contains message formats and state machines for configuring the flow tables and obtaining network state information. The southbound protocol may rely on other protocols to perform tasks related to network control, such as neighbor discovery and controller discovery.

Renan C. A. Alves is supported by grants #2016/21088-1 and #2018/11295-5, São Paulo Research Foundation (FAPESP). Cintia B. Margi is supported by CNPq research fellowship #307304/2015-9. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001.

A neighbor discovery protocol is responsible for gathering and maintaining neighborhood information, i.e., learning reachability information regarding other network nodes. This includes estimating the link quality between adjacent nodes and deciding when to transmit information to the controller.

The goal of a controller discovery protocol is to find the next hop towards the controller at network bootstrap. This is necessary as the flow tables are empty at the beginning and the nodes need to send neighborhood information and flow requests to the controller. The controller overwrites the decisions taken by the controller discovery protocol once it acquires enough topological information.

SDN has not yet been widely used in the context of wireless networks, but the few known efforts to softwarize constrained wireless networks are not equipped with appropriate discovery algorithms to support unidirectional links. Instead, nodes discover their neighbors by receiving messages and assuming the link is bidirectional.

The problem addressed in this paper is to find a centralized solution for routing in networks that contain unidirectional links. In particular, we develop neighbor and controller discovery algorithms that are embedded in an SDN framework.

Our main contributions in this paper are:

- We present, in Sections II and III, the underlying discovery algorithms for the centralized SDN-based solution to deal with unidirectional links, and the corresponding proof-of-concept implementation.
- In Section IV, we describe the experiment scenarios, including protocols, topology and gathered metrics. We perform various experiments in Section V to assess the overhead, data delivery, data packet delays, and link discovery rates of our solution.
- Our solution is shown to perform better than a naive approach in the presence of directional links and it has similar performance to traditional bidirectional-focused solutions in the absence of unidirectional links.

II. Improved neighbor discovery

The classic technique to perform neighbor discovery is to broadcast beacon packets at constant intervals and to assume the beacon sender is reachable by the receiver. However, due to the centralized nature of SDN, this assumption is not needed.

Advantages of the beacon broadcasting approach are the simplicity of the protocol and the asynchronous operation. Unfortunately, this simplicity leads to a waste of resources as beacon packets do not serve any other purpose and increase medium congestion.

By leveraging overhearing (Section II-A) and non-constant beacon intervals (Section II-B), we decrease the use of discovery packets at the expense of a small increase in complexity.

In Section II-C, we describe how to integrate the task of neighbor discovery with link quality estimation. Furthermore, since discovery algorithms often focus on adding nodes in the neighbor tables and neglect node departure detection, we describe a scheme for detecting node unreachability considering unidirectional links in Section II-D.

A. Neighbor discovery by overhearing

Neighbor discovery by overhearing, also known as passive neighbor discovery, is an inexpensive way of detecting surrounding nodes [3], [25]. However it may yield inconsistent discovery delays and hinder node departure detection. Therefore, we propose to jointly use passive and active discovery. The purpose of beacon packets is to advertise sender existence. However, this can be achieved by any broadcast packet, as long as the neighbor discovery protocol is informed of the reception and the addressing information is correct. Additionally, due to the broadcast nature of a wireless medium, unicast packets may also be used for discovery, if the radio operates in promiscuous mode.

Relying solely on overhearing increases the uncertainty of the discovery delay, as there are no guarantees of packet transmission by other protocols or applications. To overcome this drawback, a node should send periodic beacons in the absence of other packet transmissions. To achieve the desired behavior, each node sets a timer to transmit a beacon packet according to the default interval. Every time any packet is successfully transmitted, the timer is reset, postponing the beacon transmission. This ensures a minimum packet transmission rate to guarantee continuous discovery, while avoiding unnecessary beacons.

B. Adaptive beacon interval

Maintaining a constant beacon transmission rate is hardly the optimal strategy for saving network resources. A node should transmit more often at boot, to enforce a quick detection by peer nodes, while less packets may need to be transmitted when the network connections are stable. Therefore varying the timer interval may decrease discovery delay and further decrease the number of discovery packets. By default, we set the initial interval to 10 seconds plus a random value based on the node id, to avoid repeated collisions. Every time a beacon is transmitted, the interval is doubled up to a maximum value (set to 2 minutes).

Using adaptive beacon intervals integrates almost seamlessly with overhearing. The only consequence is that, as the transmission timer is increased when a beacon is transmitted and the overhearing mechanism avoids beacon transmission, it may take longer to reach the maximum beacon interval.

C. Link quality estimation (LQE)

To the best of our knowledge, there is hardly any work on link quality estimation over unidirectional links. Most packet reception ratio (PRR)-based estimators are based on acknowledged messages and calculate the metric at the transmitter (such as ETX [6], F-ETX [4], and EAR [13]). An alternative to PRR-based estimators are the hardware-based estimators, such as LQI and RSSI. However, such estimators are hardware-dependent and inaccurate [2].

ETF (Expected number of Transmissions over Forward links) estimates the delivery at the receiver by the ratio of received probe packets over the transmitted probe packets [22].

However, implementation details are not provided, for example, how a node knows the number of transmitted probe packets, what triggers a metric calculation, and how to estimate the time window.

In this paper we adapt the ETF mechanism by filling in the missing gaps. In practical terms, the mechanism we use is close to the *Moving Average* algorithm described by Woo and Culler [24], but we estimate the link quality at the receiver and do not rely on link-layer ACKs. We also make use of the overhearing feature to reduce the number of probe packets.

The receiver node maintains the status history (success or failure) of the last n messages from each inbound neighbor. However, as lost messages are not detected, the history is updated only upon successfully receiving a message.

The number of lost messages between successful receptions are calculated according to the link-layer sequence numbers. Since it is assumed that all packets are overheard, the difference between the current and the last received sequence number represents the number of packet losses plus one.

The link quality is estimated as the number of losses over the number of entries in the history. The loss rate is preferred over the success rate to provide an additive routing metric.

The history size is a key parameter, as it directly influences the estimator reactivity, stability and granularity. Also, in the context of Software-Defined Wireless Sensor Networking (SDWSN), LQE is also responsible for triggering the ND algorithm to send a neighbor report packet to the controller due to differences between the last reported link quality estimate and the current estimate.

To set the history size and the controller advertisement threshold, we simulated packet transmissions by sampling binomial distributions with varying success rates. Based on our results, a history size of 16 entries combined with a threshold of 12.5% presents a reasonable tradeoff between accuracy and reactivity. This LQE integrates well with the overhearing feature, as more information is gathered with less signaling.

D. Node Unreachability Detection

Detecting node departure is a tricky task as both false positives and false negatives lead to dire consequences to the established flows, causing route recalculations and decreasing the network packet delivery rate.

If a node knows it is moving or its battery is low, it could send a message advertising this information to the neighborhood (active departure detection). However, the devices are usually not provided with appropriate hardware to obtain such information. Also, the cause of the link failure is often oblivious to the node, e.g., due to environmental changes. Therefore, we focus on passive node departure detection. The periodic beacon transmission is the baseline for the detection, as it sets a minimum packet transmission rate.

A receiving node knows at least one packet was lost if it has not received messages from a given neighbor for a time interval greater than the current beacon interval. As the beacon interval is not constant, the interval must be included within the packets, increasing the beacon packet size.

A neighbor is removed from the neighbor table if it fails to deliver messages for a period longer than a multiple of the beacon interval, the unreachability threshold t.

The threshold t is precalculated based on the current estimated loss rate r, considering a maximum false negative rate of 1%, that is, the minimum t such that $r^t < 1\%$. The value of t is limited to a minimum of 2, to avoid false positives, and to a maximum of 8, to avoid extending the departure detection.

III. Improved Controller Discovery

The controller discovery problem is inherent to SDWSN and requires a global algorithm to solve the general unidirectional link case. Particularly in unidirectional circle topologies, such as illustrated in Figure 2a, a global algorithm is required. For example, the only way for node 7 to know it reaches the controller directly is for that information to propagate via node 1 throughout node 6.

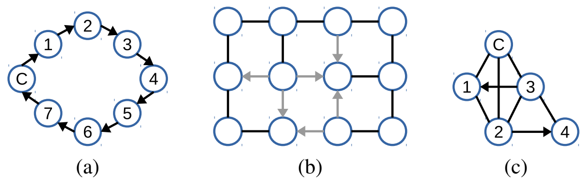


Fig. 2. a) Unidirectional circle. b) Network with unidirectional links and a bidirectionally connected component. c) Example network.

If the network graph contains a bidirectionally connected component, that is, the topology graph is still connected if all unidirectional links are removed, as the example in Figure 2b, then controller discovery can be solved by a local algorithm. We believe it takes very specific radio and environmental conditions to result in a pure unidirectional network, and therefore the existence of a bidirectionally connected component is plausible in practice.

With this assumption in mind, building a tree rooted at the controller provides an efficient solution to the problem, although nodes out of the bidirectionally connected component are not able to join the network. Since the route set by the controller discovery is temporary, the algorithm does not seek the optimal route in terms of link quality and relies on hop count to build the tree.

The controller is initialized with the minimum hop count value, while the other nodes are initialized with the maximum value. The controller discovery packets contain the current hop count towards the controller and the set of known inbound neighbors (obtained from the neighbor discovery protocol), so the receivers can check if the link is symmetric.

Each node checks its neighbor table size at exponentially increasing intervals (up to a maximum) and transmits controller discovery packets if the number of neighbors increased. Upon receiving a controller discovery packet, the node checks if there is a bidirectional link to the sender and if the hop count is better than the current value. If both conditions are true, the next hop towards the controller and the hop count are updated and the SDN layer is informed of the discovery. Also, a controller discovery packet is scheduled for transmission, regardless of neighborhood changes.

A node N_1 also transmits a controller discovery packet whenever it already knows how to reach the controller and receives a controller discovery packet with the maximum metric from node N_2. This condition indicates N_2 still does not know a next hop towards the controller and N_1 is a next hop candidate. This procedure is intended to speed up the discovery by late nodes and allow new nodes to join the network after the initial bootstrap.

Although an individual node does not know whether the other nodes already obtained a valid controller route, the controller discovery algorithm eventually stops sending messages if the network topology is stable and the nodes' neighborhoods remain constant.

Take the network of Figure 2c as an example. First, the controller detects nodes 1, 2, and 3 as inbound neighbors and transmits a controller discovery beacon with this information. As the links are bidirectional, these nodes can reach the controller directly. In the next round, nodes 1, 2, and 3 transmit their own beacon with neighborhood information. Node 2 does not switch the next hop to 1 or 3, because it is a longer route. Node 4 sets the next hop as node 3, since the beacon received from node 2 does not contain its address, and transmits a beacon to advertise its discovery. At this point controller discovery beacons are no longer transmitted as 1) the topology is stable and the set of neighbors does not change, and 2) none of the nodes will change their next hop towards the controller.

IV. Evaluation method

We have evaluated the performance of the proposed discovery algorithms and compared them to other approaches found in the literature, namely the Collect-based [17] and the Simple naive protocol [1]. All algorithms were implemented on IT-SDN [17], an SDWSN framework and southbound protocol based on Contiki OS that allows changing the discovery algorithms. IT-SDN was configured to use end-to-end acknowledgements, source-routed control packets, and neighbor table size of 10 entries.

The algorithms were benchmarked with the COOJA WSN simulator/emulator tool [20], using sky mote binaries and DGRM (Directed Graph Radio Medium) to model the radio links. DGRM allows defining unidirectional links, opposed to the other available radio medium models.

The nodes were positioned to form square grid and random topologies. The random topologies were generated with NPART [19], using the default parameters for Berlin networks.

The baseline scenarios were fully bidirectional networks, without any unidirectional links. We used three approaches to introduce unidirectional links in the baseline topologies: 1) random links turned unidirectional (15% of all links) 2) random nodes with increased range (the range of 20% of all nodes is doubled), and 3) only controller with increased range (reaching all network nodes).

We tested with increasing topology sizes to check the algorithms sensitivity to the number of nodes in the network. This parameter ranged from 16 to 100 nodes (square numbers only). All nodes in the network transmitted CBR data, except the data sink and the controller node. The data payload size was 10 bytes, transmitted at 1 packet per minute. The controller was positioned at a grid corner, while the data sink was placed at the grid midpoint. The positioning of these nodes was random in the random topologies.

The controller software ran on the host machine and connected with the network through the COOJA serial server extension. It calculated the best routes according to link quality values provided by the neighbor discovery protocol.

We have considered the following performance metrics:

- *Data delivery*: the global percentage of data packets that successfully reached their destination.
- *Data delay*: the average time between the data packets transmission and reception (at the application layer, therefore queuing and flow setup delays are included).
- *Control overhead*: the total number of non-data packets transmitted in the network, which is related to the discovery algorithm's efficiency.
- *Link discovery rate*: the percentage of existing links that the neighbor discovery algorithm was able to detect throughout the simulation. The discovery rate is measured at the controller, considering its global representation.
- *Bootstrap transient duration*: the time elapsed since the beginning of the simulation until the controller internal representation of the network contains all possible nodes.

For each parameter combination, ten 60-minute-long simulation runs were executed to achieve statistical significance. The graphs presented in the following section show 95% confidence intervals.

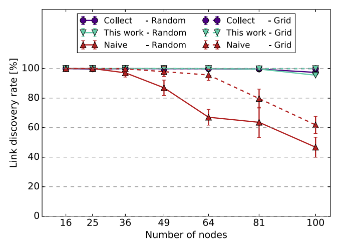
V. Results and discussion

The results presented in this section are organized with one graph for every pair of metric, and link type (fully bidirectional, controller to all, nodes with increased range and random unidirectional links), each graph containing the evolution of metric values as the network size grows for the combinations of neighbor discovery algorithm (collect, naive, and this work) and topology type (random or grid).

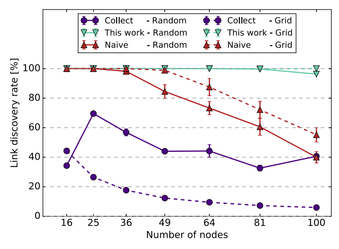
Let us start the analysis with the link discovery rate, displayed in Figure 3. Our approach discovered nearly all links in all topologies (100% on 75% of the test cases, minimum 92%), while the collect-based discovery failed to find most of the unidirectional links, and the naive algorithm presented scalability issues, consistently presenting discovery above 80% only on networks up to 36 nodes.

A neighborhood larger than the neighbor table capacity causes our algorithm to discover less than 100% of the links, as the exceeding links are ignored. The other reason for undetected links is the lack of a bidirectionally connected component, as the nodes without a bidirectional path towards the controller are unable to send neighbor report packets (49-nodes and 81-nodes random links scenarios, Figure 3d).

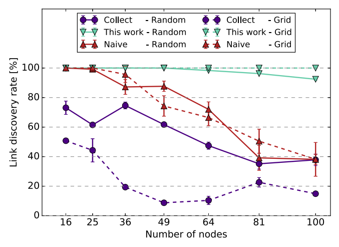
The naive algorithm performed worse on random than grid topologies, as the number of links per node can be large,



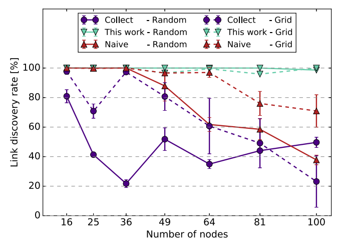
(a) Fully bidirectional



(b) Controller to all



(c) Some nodes with increased range

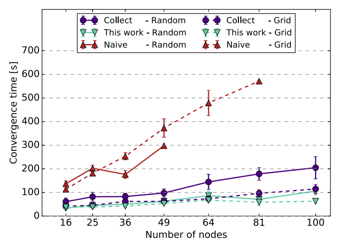


(d) Random unidirectional links

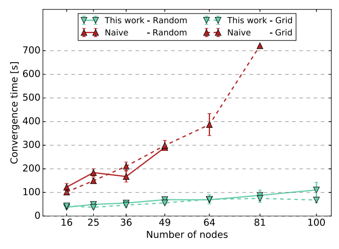
Fig. 3. Link discovery rate.

quickly depleting the limited amount of memory reserved to store reachability information. From 64-node scenarios onward, the number of links is just too large, hampering whole network discovery.

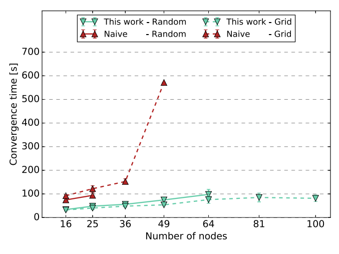
Collect-based discovery performed as good as our algorithm for fully bidirectional networks, the difference is that it stores the outbound neighbors while we store the inbound neighbors, causing the small difference for the 100-node random topology (Figure 3a). On the other hand, collect failed to detect unidirectional links, degrading the observed performance on the other test cases.



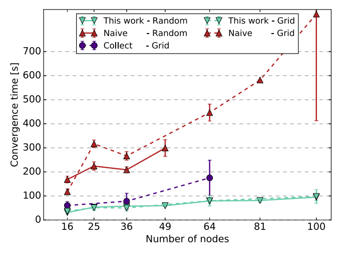
(a) Fully bidirectional



(b) Controller to all



(c) Some nodes with increased range



(d) Random unidirectional links

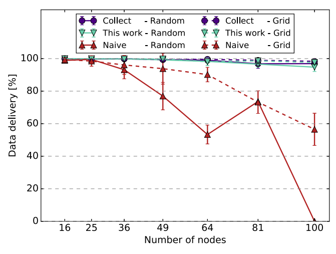
Fig. 4. Discovery algorithms convergence time.

The next metric is the convergence time of the discovery algorithms, shown in Figure 4. With fully bidirectional, our algorithm converged 33% faster than collect on average. Once again, the lack of scalability of the naive algorithm is clear, as the convergence time grew fast and did not converge in larger networks. Note that missing points in the graphs indicate the algorithm did not converge at all.

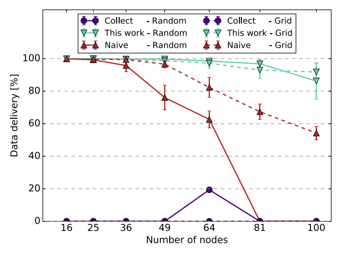
Convergence time in random topologies is usually larger than in grids, due to the larger network diameter. Regardless of the topology, our algorithm convergence time displays a linear trend regarding the number of nodes, but with a mild slope (for example, the convergence time increases 1.5 times on average, comparing 100-node and 16-node scenarios). The scenarios without convergence are due to the neighbor table limitation, causing the only link connecting certain nodes to be left out (Figure 4c) or due to the lack of a bidirectional path to the controller (Figure 4d).

Considering the naive algorithm, the discovery time increases about 3.1 times considering the 16-node and largest scenario with convergence. It converges on grids with more nodes in comparison to random topologies, as the algorithm is aided by the grid regularity.

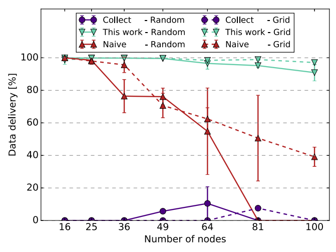
As collect is not adapted to unidirectional links, it is unable to converge on most scenarios. It may detect a unidirectional link as bidirectional, causing packet losses and preventing the control packets from reaching the most distant nodes.



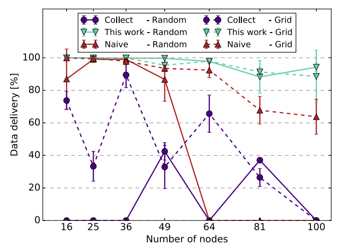
(a) Fully bidirectional



(b) Controller to all



(c) Some nodes with increased range



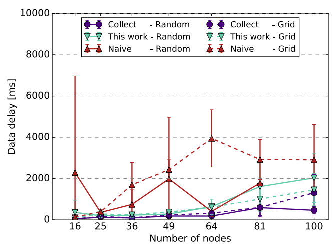
(d) Random unidirectional links

Fig. 5. Data delivery results.

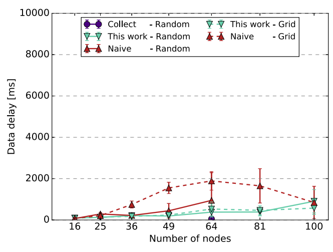
The data delivery results (Figure 5) are partially explained by the last two examined metrics. Links unbeknownst to the controller limit the paths possibilities, increasing the chances of congestion and collisions. Moreover, all data transmission attempts from undiscovered nodes fail. Undiscovered nodes are the main cause of decreasing delivery as the network size increases for the naive algorithm. In large random topologies, no data is delivered as the sink does not make it into the controller network representation.

It is noticeable that all algorithms tend to perform worse on random topologies in comparison to grids, especially for the naive algorithm. This highlights the importance of testing algorithms in different setups. Our algorithm performed up to 6.4% worse on random topologies in comparison to grids.

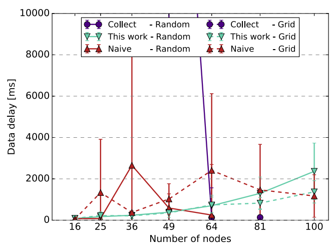
In the fully bidirectional links scenario, our solution performs up to 2.2% less than the collect algorithm. The reason behind this result is that the flows take longer to be installed on the nodes, causing losses at the network startup. The flows take longer to be installed because, whenever a node asks a flow in the collect algorithm, the controller already has a route to answer the request, while in our approach this may be not true, as the forward and backward links are informed separately. The other factor is we disabled link-layer acknowledgements to support unidirectional links properly. This increases the link-layer losses, causing more retransmissions of flow setup packets, thus contributing to the initial flow setup delay.



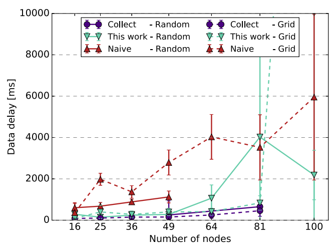
(a) Fully bidirectional



(b) Controller to all



(c) Some nodes with increased range



(d) Random unidirectional links

Fig. 6. Data delay results.

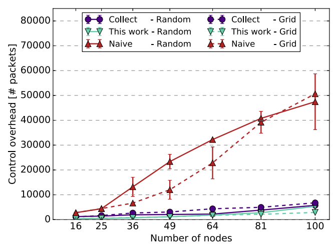
Figure 6 shows the delay of data packets. Our algorithm is outperformed by collect in most of the scenarios in which collect delivered data packets, but often it did not. Scrutinizing the data, we observed the delay from the delivery of the first data packets of some nodes cause the difference in the delay average. After the flows are properly set, the remaining delays are very close, regardless of the discovery protocols.

The explanation for the increased initial delay for our algorithm is similar to the reason why some packets are lost at the network start-up. That is, it takes longer to successfully deliver flow setup packets with our algorithm, because the link-layer acknowledgements are disabled, and the forward and backward links are informed separately (for bidirectional links). The failure to deliver flow setup packets may also cause transient loops in the network, further increasing the delay to set the initial flows. Initial flow setup delay is more evident as the network gets larger, since the paths from the controller to the nodes get longer. The random topology aggravates network congestion as some nodes are highly connected, which explains the performance difference between random and grids at large networks.

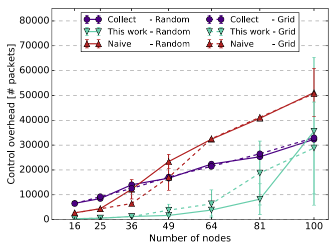
It is noticeable that some scenarios presented a high dispersion, represented by the confidence interval. As the aforementioned delays of initial flow setups depend on essentially random events, some simulations yielded a high delay, typifying an outlier among the data. This randomness affects the stability of the collect protocol in networks with unidirectional links, for example causing the spike observed in Figure 6c.

The "controller to all" link type showed little variation for our algorithm, as the initial setup delay is largely mitigated by the unidirectional link from the controller to the other nodes. The faster flow configuration also causes the overall delay to be lower than the other scenarios, indicating that increasing the controller radio power is beneficial for starting the network.

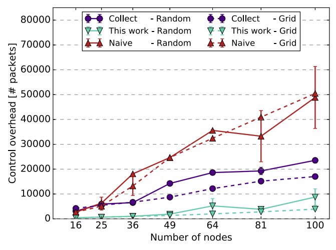
The naive algorithm once again shows disadvantageous results in most of the scenarios, also presenting larger dispersion. The low packet delivery rate causes the delay to be small in large networks, as only the nodes near the sink are able to deliver packets. At least it get results while collect often not.



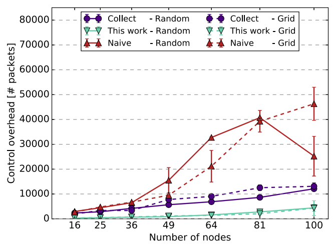
(a) Fully bidirectional



(b) Controller to all



(c) Some nodes with increased range



(d) Random unidirectional links

Fig. 7. Control overhead results.

Lastly, let us investigate the control overhead generated by each discovery algorithm, exhibited in Figure 7. Our algorithm transmits less than the other algorithms, except in the following scenario: 100-node networks with controller-to-all links. In fully bidirectional grids, our algorithm transmits 52% less control messages on average. However, the number of control packets shown in the graphs accounts for every kind of control packet, including discovery beacons, neighbor reports to the controller, flow requests and flow setups. Analyzing the distribution of the control packet types gives further insight into the behavior of each algorithm.

With respect to discovery beacons, our algorithms transmits at least 68% less packets than collect, 92% on average. This is the effect of the beacon reducing mechanisms implemented in our approach (overhearing and dynamic beacon intervals).

On the other hand, the number of neighbor report packets the nodes send to the controller is larger in our approach, notably at controller-to-all link type. This indicates that our link quality estimator is more sensitive than the collect's LQE, thus, there is room for fine-tuning our estimator parameters. For example, the controller transmits a lot of packets, causing the calculated LQE to vary in controller-to-all link scenarios.

More neighbor report packets reaching the controller may lead to route recalculation, thus triggering transmission of flow setup packets. We notice a correlation between the transmission frequency of these packet types. On the one hand, the naive algorithm does not estimate the link quality, therefore it transmits very few neighbor report and flow setup packets. On the other hand, it is not equipped with any mechanism to curb the overhead of the discovery packets, which is its major source of control overhead.

VI. Related work

In this section, we present works focused on enabling routing over unidirectional links. A strategy is to reduce the scope of the problem and provide a solution that works only in specific conditions [5], [12]. Another strategy is to find an alternative path to the unidirectional link by flooding the network [11], [21].

Chen *et al.* proposed a probabilistic routing algorithm focused on many-to-one traffic [5]. The network is sliced into concentric stripes centered at the destination. Nodes transmit every packet multiple times, and the receivers forward with a certain probability. A node closer to the sink may overhear the packet and opportunistically transmit it to the sink, even if the link is unidirectional. The main drawbacks of this work are restriction to many-to-one traffic and waste of resources due to multiple transmissions of the same packet [5].

Kim *et al.* assume a network with a single sink able to reach any node in the network in one hop (but not the other way around). The objective is to provide efficient and reliable downward data transmission. However, the main assumption is not easy to generalize to the case where any link could be unidirectional [12].

Bidirectional Routing Abstraction (BRA) provides full support for unidirectional links by searching through the network for multihop reverse paths to the unidirectional links. It uses a Distributed Bellman–Ford algorithm, which floods the network with distance vector information. To curb the control overhead, the maximum length of the reverse path is limited. The simulation results, based on the IEEE 802.11 standard, show that the AODV protocol performs better with BRA than using the traditional blacklisting mechanism [21].

Unidirectional Link Counter (ULC) is actually a cross-layer protocol, as it mixes functionality from medium access and routing layers. The protocol is similar to AODV in the sense that Route Request and Route Response messages are used to discover routes, i.e., it is a flooding-based protocol. In addition, these messages are used to perform link discovery. If a link is unidirectional, the forwarding node relegates the forwarding task to its neighbors, expecting that at least one of them is able to contour the unidirectionality and find a reverse path. Both simulations and testbed results show performance enhancements in comparison to AODV [11].

It is noteworthy that BRA and ULC are focused on MANETs and use flooding-based messages. In the context of WSN it is desired to avoid flooding in order to diminish the number of control packets throughout the network.

There are several Software-Defined Wireless Sensor Networking frameworks in the literature, but the neighbor discovery process is not detailed in most of them. The earlier proposals were adaptations of the Openflow protocol [18] to WSN, namely FlowSensor [16] and Sensor Openflow [15]. Neither of them mentions the discovery process, although one could infer they use a variation of the Link Layer Discovery Protocol (LLDP), since it is the OpenFlow default.

TinySDN is a southbound protocol specification based on flow labeling, built on top of TinyOS ActiveMessage component [7]. Neighbor and controller discovery are relegated to the Collection Tree Protocol (CTP) [10], which in turn builds a tree rooted at the controller. Link quality is obtained from the TinyOS 4-bit link estimator, requiring bidirectional links. The authors do not detail the criteria for transmitting neighborhood information to the controller due to link quality variations.

IT-SDN is based on TinySDN, but with the goal of being OS-independent and to allow for replaceable discovery algorithms [17]. The default configuration is to use a CTP-like protocol for neighbor discovery, which is similar to TinySDN and presents the same drawbacks. A node sends topological information to the controller if the CTP link quality estimation differs more than 20% from the last reported value.

In a previous work, we introduced the naive neighbor and controller discovery algorithms for networks with unidirectional links [1]. This work is an improvement, we fixed the scalability issues (as shown in the results section), added a link quality estimation and neighbor unrechability detection.

SDN-Wise executes controller and neighbor discovery as a single operation by implementing its own topology discovery protocol [9]. The (possibly) multiple controllers start the construction of a tree by transmitting Topology Discovery packets. The tree is rebuilt periodically to obtain fresh topology information. Nodes transmit topology information to the controller based on a fixed periodic interval. The authors analyzed the overhead of shortening this interval.

The main shortcomings of the SDN-Wise discovery protocol are assuming links are bidirectional, unnecessary transmission of control packets due to the lack of adequate criteria to send topological information to the controller, fixed Topology Discovery transmission interval, and use of RSSI as link metric (hardware dependent).

Theodorou and Mamatas proposed two discovery protocols, which they called “neighbor advertisement” and “neighbor request” [23]. The controller is responsible for starting either algorithm by transmitting a unicast message to its neighbors, which in turn broadcasts a discovery packet. In the “neighbor advertisement” each receiving node advertises the discovered link to the controller, while in the “neighbor request” each receiving node answers back to sender, which is responsible for reporting the controller about the links. The controller transmits a unicast packet to the newly detected nodes, until all network nodes are discovered. However, they do not specify how to perform the link quality assessment, how the controller discovery process occurs and how to deal with unidirectional links. Also, the paper focuses only on initial discovery and does not discuss when to re-collect the neighborhood information.

Our work improves on the existing neighbor discovery algorithms for SDWSN as we employ techniques to reduce the overall overhead associated with the task, while we properly support discovery of unidirectional links. We also integrated the task of link quality assessment and node departure detection, providing details on how to calculate the ETF and a criterion to send new topology update messages to the controller. Table I summarizes the main features of the discovery algorithms found in SDWSN frameworks.

TABLE I
SDWSN NEIGHBOR DISCOVERY COMPARISON.

Work	Approach	Unidir link	LQE	Criteria to controller	Neighbor departure
[7]	Collect	No	ETX	Undisclosed	Long timer
[17]	Collect	No	ETX	ETX variation	Long timer
[9]	Periodic	No	RSSI	Periodic	Periodic recollection
[23]	Controller	No	RSSI	Undisclosed	Undisclosed
[1]	Periodic	Yes	Hop	New nodes	Undisclosed
This work	Periodic + overhearing	Yes	ETF	LQE variation	Based on LQE

VII. Conclusion

Unidirectional links are a reality in constrained networks due to heterogeneity of devices, different radios, and multipath fading. Nonetheless, most existing routing protocols either do not account for such links or employ blacklisting mechanisms. The few algorithms that make use of unidirectional links are tailored to specific network topologies or depend on flooding to find a reverse path to the unidirectional link.

We argue that a centralized approach is both general purpose and flooding free. Therefore, we have proposed an SDN-based solution, imbued with controller and neighbor discovery mechanisms, to make use of unidirectional links. The discovery algorithms are designed to generate low control overhead, while providing link quality estimation.

Simulation results show that our solution performs well in the presence of unidirectional links and is more scalable than an existing discovery algorithm for unidirectional networks. Also, considering fully bidirectional networks, we are competitive with traditional discovery algorithms, while the latter perform poorly under unidirectional links.

We have identified the “controller-to-all” topology as a useful technique to decrease the overall data delivery delay and initial network setup at the expense of increasing link layer congestion. Thus, coupling a radio power control mechanism to the controller, with the objective of balancing this trade-off, could be a good direction for future research.

References

[1] R. C. A. Alves and C. B. Margi. Discovery protocols for SDN-based wireless sensor networks with unidirectional links. In *XXXV SBrT*, São Pedro, Brazil, 2017. Sociedade Brasileira de Telecomunicações.

[2] N. Baccour, A. Koubâa, H. Youssef, and M. Alves. Reliable link quality estimation in low-power wireless networks and its impact on tree-routing. *Ad Hoc Netw.*, 27(C):1–25, Apr. 2015.

[3] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. SenSys '08, pages 43–56, New York, NY, USA, 2008. ACM.

[4] S. Bindel, S. Chaumette, and B. Hilt. F-ETX: An Enhancement of ETX Metric for Wireless Mobile Networks. *Communication Technologies for Vehicles*, 9066:117–128, 2015.

[5] X. Chen, Z. Dai, W. Li, and H. Shi. Performance guaranteed routing protocols for asymmetric sensor networks. *IEEE Transactions on Emerging Topics in Computing*, 1(1):111–120, June 2013.

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

[7] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel. TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. In *LATINCOM*, pages 1–6, Nov 2014.

[8] Forbes. 2017 roundup of internet of things forecasts. https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts, 2017. Accessed on August 27th 2018.

[9] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo. SDN-WISE : Design , prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. *INFOCOM*, pages 513–521, 2015.

[10] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. SenSys '09, New York, NY, USA, 2009. ACM.

[11] R. Karnapke and J. Nolte. Unidirectional link counter - a routing protocol for wireless sensor networks with many unidirectional links. In *MED-HOC-NET*, pages 1–7, June 2015.

[12] H.-S. Kim, M.-S. Lee, Y.-J. Choi, J. Ko, and S. Bahk. Reliable and energy-efficient downward packet delivery in asymmetric transmission power-based networks. *ACM TOSN*, 12(4):34:1–34:25, Sept. 2016.

[13] K.-H. Kim and K. G. Shin. On accurate measurement of link quality in multi-hop wireless mesh networks. MobiCom '06, pages 38–49, New York, NY, USA, 2006. ACM.

[14] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE IoT Journal*, 4(5), Oct 2017.

[15] T. Luo, H.-P. Tan, and T. Q. S. Quek. Sensor openflow: Enabling software-defined wireless sensor networks. *IEEE Communications Letters*, 16(11):1896–1899, 2012.

[16] A. Mahmud and R. Rahmani. Exploitation of openflow in wireless sensor networks. In *Computer Science and Network Technology (ICCSNT)*, volume 1, pages 594–600, 2011.

[17] C. B. Margi, R. C. A. Alves, G. A. N. Segura, and D. A. G. Oliveira. Software-defined wireless sensor networks approach: Southbound protocol and its performance evaluation. *OJIOT*, 4(1):99–108, 2018.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM*, 38(2):69–74, Mar. 2008.

[19] B. Milic and M. Malek. NPART - Node Placement Algorithm for Realistic Topologies in Wireless Multihop Network Simulation. Simutools '09, pages 9:1–9:10, 2009.

[20] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. Nov 2006.

[21] V. Ramasubramanian and D. Mosse. Bra: A bidirectional routing abstraction for asymmetric mobile ad hoc networks. *IEEE/ACM Transactions on Networking*, 16(1):116–129, Feb 2008.

[22] L. Sang, A. Arora, and H. Zhang. On link asymmetry and one-way estimation in wireless sensor networks. *ACM Trans. Sen. Netw.*, 6(2):12:1–12:25, Mar. 2010.

[23] T. Theodorou and L. Mamatas. Software defined topology control strategies for the internet of things. (NFV-SDN), Nov 2017.

[24] A. Woo and D. Culler. Evaluation of efficient link reliability estimators for low-power wireless networks. Technical Report UCB/CSD-03-1270, EECS Department, University of California, Berkeley, 2003.

[25] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. SenSys '03, pages 14–27, New York, NY, USA, 2003. ACM.