Asymmetry-Aware Link Quality Services in Wireless Sensor Networks

Technical Report: MIST-TR-2005-006

Junzhao Du, Weisong Shi, and Kewei Sha
Department of Computer Science
Wayne State University
{*junzhao, weisong, kewei*}*@wayne.edu*

Abstract

Recent study in wireless sensor networks (WSN) has found that the irregular link quality is a common phenomenon. The irregular link quality, especially link asymmetry, has significant impacts on the design of WSN protocols, such as MAC protocols, neighborhood and topology discovery protocols, and routing protocols. In this paper, we propose asymmetry-aware link quality services, including *the neighborhood link quality service (NLQS)* and *the link relay service (LRS)*, to provide the timeliness link quality information of neighbors, and build a relay framework to alleviate effects of the link asymmetry. To demonstrate proposed link quality services, we design and implement two example applications, *the shortest hops routing tree (SHRT)* and *the best path reliability routing tree (BRRT)*, in the TinyOS platform. To evaluate proposed link quality services, we have conducted both static analysis and simulation through the TOSSIM simulator, in terms of four performance metrics. We found that the performance of two example applications was improved substantially. More than 40% of nodes identify more outbound neighbors and the percentage of increased outbound neighbors is between 14% and 100%. In SHRT, more than 15% of nodes reduce hops of the routing tree and the percentage of reduced hops is between 14% and 100%. In BRRT, more than 16% of nodes improve the path reliability of the routing tree and the percentage of the improved path reliability is between 2% to 50%.

1 Introduction

The integration of sensing, computing and communication into one embedded device makes wireless sensor networks (WSN) widely used in many applications [1, 9, 10, 11, 16, 17]. Recent empirical study [3, 7, 15, 18, 19, 20] on Berkeley motes platform shows that there are highly irregular links in real deployments. Firstly, the packet delivery performance, in term of packet reception rate (PRR), varies significantly with spatial and temporal factors. So it is difficult to get the timeliness link quality information between neighbors. The link quality is valuable for some routing protocols, which requires the timeliness link quality information to make decision. Secondly, approximately 5% to 15% of all links are asymmetric links and asymmetric links vary significantly in different directions and distances. Asymmetric links, especially unidirectional links, bring the problem to the design of WSN protocols, such as MAC protocols, neighborhood and topology discovery protocols, and routing protocols. To our knowledge, few previous protocols leverage the fact that there are outbound or inbound neighbors, which is the main problem that this paper intent to address.

In this paper, we intend to alleviate the problem caused by the link asymmetry. Our work is motivated by our measurements and observations. From our measurements, we found that the antenna orientation of sending and receiving motes contributes a lot to the radio irregularity and it heavily affects PRR between nodes. Furthermore, we also observed some interesting phenomena. Let's assume that Mote B can hear Mote A, but the latter can not hear Mote B. In this case, usually neither Mote A nor Mote B treats the counterpart as its neighbor. However, if there is a Mote C, sitting between Mote A and Mote B, and has good connections with both Mote A and B. So Mote C can help Mote B to forward packets to Mote A. By doing so, Mote A can treat Mote B as one of its outbound neighbors, while Mote B can treat Mote A as one of its inbound neighbors. The above observation inspires us to design and implement asymmetry-aware link quality services, including *the neighborhood link quality service (NLQS)* and *the link relay service (LRS)*, to provide the link quality information of neighbors and build a relay framework to alleviate effects of the link asymmetry in the TinyOS platform [5].

To demonstrate our proposed link quality services, we design and implement two example applications, *the shortest hops routing tree (SHRT)* and *the best path reliability routing tree (BRRT)*, in the TinyOS platform. To evaluate our proposed link quality services, we have conducted both static analysis and simulation through the TOSSIM simulator [6]. We found that the performance of two example applications was improved substantially. More than 40% of nodes identify more outbound neighbors and the percentage of increased outbound neighbors is between 14% and 100%. In SHRT, more than 15% of nodes reduce the hops of the routing tree and the percentage of the reduced hops is between 14% and 100%. In

BRRT, more than 16% of nodes improve the path reliability of the routing tree and the percentage of the improved path reliability is between 2% to 50%.

The contributions of the paper include three-fold:

- *Design and implement the neighborhood link quality service to provide the timeliness link quality information of neighbors.* We design and implement NLQS in the TinyOS platform to measure and estimate the link quality with link quality estimator. In this service, we identify inbound and outbound neighbors, build the inbound and outbound neighbors tables separately, and provide the *LinkQuality* interface to query the timeliness link quality information of neighbors.

- *Design and implement the link relay service to alleviate effects of the link asymmetry.* In this service, we propose a link relay protocol, implement a link relay framework, and provide the *LinkRelay* interface to relay packets across asymmetric links in the TinyOS platform.

- *Design and implement two example applications to demonstrate and evaluate link quality services.* We design and implement two example applications, SHRT and BRRT, using link quality services, in the TinyOS platform. In SHRT and BRRT, nodes will use link quality services to get the link quality information to the parent node and flood the routing tree information across asymmetric links.

The rest of the paper is organized as follows. The design and implementation of link quality services, including NLQS and LRS, are presented in Section 2 and Section 3 separately. SHRT and BRRT are described in Section 4. In Section 5, some performance metrics to evaluate link quality services are defined and evaluation results are discussed. Related work is compared in Section 6. In Section 7, we conclude our research work and list our future work.

2 Asymmetry-Aware Link Quality Services

In this section, an overview of our asymmetry-aware link quality services is presented first, followed by the design of NLQS and LRS.

2.1 System Overview

Currently, our link quality services provide two interfaces for applications. Figure 1 is the architecture of link quality services. Applications use the *LinkQuality* interface to query the timeliness link quality information of neighbors and use the *LinkRelay* interface to relay packets across asymmetric links. The *LinkQuality* is provided by NLQS. The *LinkRelay* is provided by LRS. And NLQS also uses the *LinkMeasure* interface provided by LRS to get packet information in order to measure

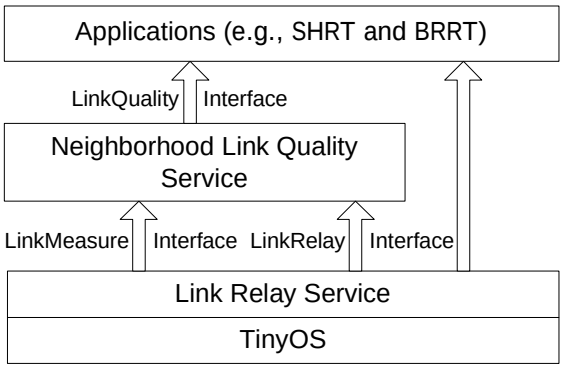

Figure 1: The architecture of link quality services.

and estimate the link quality between neighbors. NLQS uses the *LinkRelay* interface to relay the estimated PRR of neighbors across asymmetric links. In the following sections, we will focus on NLQS and LRS.

2.2 Neighborhood Link Quality Service

To make full use of those outbound neighbors isolated by asymmetric links, we should identify inbound and outbound neighbors and build inbound and outbound neighbors tables separately. Furthermore, NLQS provides the *LinkQuality* interface to facilitate applications to query the timeliness link quality information of neighbors.

2.2.1 Building Inbound and Outbound Neighbors Tables

In NLQS, if a node, A, can receive packets from a neighbor, B, node B will be identified as an inbound neighbor of node A. If node A can send packets to a neighbor, C, node C should be identified as an outbound neighbor of node A. However, to identify node C as its outbound neighbor, node A should receive acknowledgement from node C. Due to asymmetric links, if node A can not receive acknowledgement from node C, then node A can not identify node C as its outbound neighbor. Furthermore, the link quality, in term of PRR, from node A to node B is different from that from node B to node A. So in NLQS, every node distinguishes inbound neighbors and outbound neighbors and build tables for them independently. The node ID and PRR of the link from the inbound neighbor is stored in the inbound neighbors table while the node ID and PRR of the link to the outbound neighbor is stored in the outbound neighbors table.

We measure the link quality using the combination of active probing and passive overhearing techniques, and leverage the Window Mean Exponentially Weighted Moving Average Estimator (WMEWMA) [4, 14, 15] to estimate the link quality based on current and history measured results. Figure 2 is the pseudocode of the algorithm about how to build inbound and outbound neighbor tables using link quality measurement and estimation techniques. In this algorithm, every node will

Algorithm 1 Building inbound and outbound neighbors tables

Procedure Send-Packet ()
Begin Procedure

1: **loop**
2: periodically build an *active probing packet* with *node ID*, *packet ID*, and *estimated link quality* between neighbors
3: randomly select a time and broadcast the *active probing packet*
4: **end loop**

End Procedure

Procedure Receive-Packet ()
Begin Procedure

1: **loop**
2: passively overhear the *active probing packet*
3: update the *inbound neighbors table* according to *node ID*, *packet ID* of the received *active probing packet*
4: **if** it is time to do estimation for this inbound neighbor **then**
5: do estimation using *WMEWMA* for this inbound neighbor
6: **end if**
7: update the *outbound neighbors table* with *estimated link quality* between neighbors from the received *active probing packet*
8: **end loop**

End Procedure

Figure 2: The pseudocode of the algorithm for building inbound and outbound neighbors tables.

periodically broadcast some packets, which contain node ID and packet ID. And every node only measures and estimates PRR of the link from its inbound neighbors. After that, it will store the inbound neighbor in the inbound neighbors table and broadcast the estimated PRR in following probing packets. And when node receives the estimated PRR of the link to its outbound neighbors from broadcast packets, it will identify this outbound neighbor and store it in the outbound neighbors table. As we have discussed before, due to the link asymmetry, node will not receive the acknowledgement from its outbound neighbor. In this case, LRS (Section 2.3) will be used to relay the acknowledgement across asymmetric links.

The *WMEWMA* Link quality estimator is used to estimate PRR of the link from inbound neighbors into NLQS. For the completeness of this paper, we describe the basic idea of WMEWMA here as well. WMEWMA uses a time window to observe the received packet and it adjusts the estimation result using latest average value of PRR. Below is equation used by WMEWMA to estimate PRR.

$$\bar{M}_i = \frac{M_i}{P_i} \tag{1}$$

$$E_i = \alpha E_{i-1} + (1-\alpha)\bar{M}_i \tag{2}$$

where P_i is the number of packets sent in time window i; M_i is the number of packets received in time window i; $\bar{M}_i$ is the average PRR measured in time windows i; E_{i-1} is PRR estimated by previous time window; E_i is PRR estimated by the end of this time window. The parameter α is called the estimator gain, and is used to determine the reactivity of the estimator. If the gain is large, the old estimated value will dominate the estimate result and the estimator will be slow to change, making it stable. In contract, if the gain is small , the estimator will tend to be agile. In this paper, we will send 20 packets in every time window and set the value of α to 0.5.

2.2.2 Link Quality Interface

NLQS provides the *LinkQuality* interface for applications to query the timeliness link quality information of inbound and outbound neighbors. Figure 3 lists the details of the interface, which consists of one event and eight commands, complying with the TinyOS and nesC standard [5]. When the service is ready, it will signal the $Ready$ event to notify the availability of the service. Even the service is ready for use, NLQS will continue measure and estimate the timeliness link quality between neighbors and update inbound and outbound neighbors tables. Applications can call those commands to query the link quality information. Commands are divided into two groups, one is for inbound neighbors and the other is for outbound neighbors. Applications can get the information about how many inbound and outbound neighbors are identified by the service; what is the node ID of inbound and outbound neighbors; what is the link quality from inbound neighbors and to outbound neighbors; whether a given node ID is one of inbound and outbound neighbors.

Link Quality Interface

```
interface LinkQuality {
  event       result_t   Ready();
  command uint8_t    get_IN_Neighbor_num();
  command result_t   is_IN_Neighbor(uint16_t  id);
  command float      get_IN_Neighbor_prr(uint16_t  id);
  command uint16_t   get_IN_Neighbor_ID(uint8_t index);
  command uint8_t    get_OUT_Neighbor_num();
  command result_t   is_OUT_Neighbor(uint16_t  id);
  command float      get_OUT_Neighbor_prr(uint16_t  id);
  command uint16_t   get_OUT_Neighbor_ID(uint8_t index);
}
```

Figure 3: The LinkQuality interface.

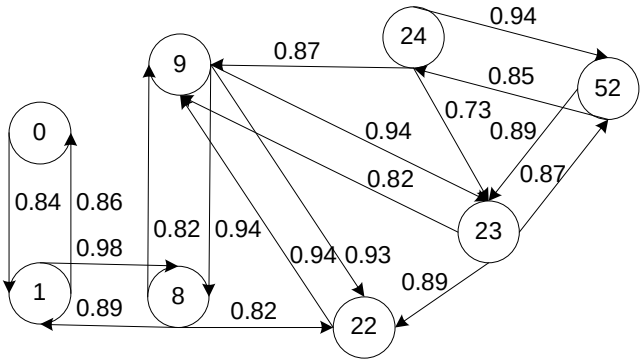

Figure 4: An example topology to illustrate the terminology.

2.3 Link Relay Service

In LRS, we design a simple link relay protocol and implement a relay framework to alleviate effects of the link asymmetry for general applications in the TinyOS platform. We will use this service, through the *LinkRelay* interface, to identify more outbound neighbors isolated by asymmetric links and forward packets across asymmetric links. In following sections, we will give some definitions used in LRS. After that, the framework, the protocol, and the *LinkRelay* interface of LRS will be described.

2.3.1 Terminology

To facilitate the description of the service, we first give out the definitions used in LRS. The topology of a wireless sensor network is a weighted directed graph (G). All nodes in the topology belong to the vertex set of the directed graph, called $V(G)$. All links in the topology belong to the edge set of the directed graph, called $E(G)$. We use $e(u, v)$ to represent the link from node u to node v, and assign PRR of a link between nodes as $weight$ of the edge. Next, we introduce several derived concepts using the terminology, including *theoretical outbound neighbors*, *No Relay outbound neighbors*, *One-Step Relay outbound neighbors*, and *Two-Step Relay outbound neighbors*.

Definition 1 Theoretical outbound neighbors of node v (N(v)):
$\{u|\forall u \in V(G) \wedge e(v, u) \in E(G)\}$

Definition 2 No Relay outbound neighbors of node v (N0(v)):
$\{u|\forall u \in V(G) \wedge e(v, u) \in E(G) \wedge e(u, v) \in E(G)\}$

Definition 3 One-Step Relay outbound neighbors of node v (N1(v)):
$\{u|(u \in N0(v)) \vee (\forall u \in V(G) \wedge e(v, u) \in E(G) \wedge \exists m \in V(G) \wedge e(u, m), e(m, v) \in E(G))\}$

Definition 4 Two-Step Relay outbound neighbors of node v (N2(v)):
$\{u|(u \in N1(v)) \vee (\forall u \in V(G) \wedge e(v, u) \in E(G) \wedge \exists m, n \in V(G) \wedge e(u, m), e(m, n), e(n, v) \in E(G))\}$

Figure 4 is an example weighted directed graph, which illustrates the terminology. In this topology, according to definitions, $N(24)$ includes node 9, node 23 and node 52; $N0(24)$ only includes node 52; $N1(24)$ includes node 52 and node 23; $N2(24)$ includes node 52, node 23, and node 9.

From these definitions and the example, we know that we should find a path from outbound neighbors to the node, which will identify its outbound neighbors. If there is a path, there will be opportunity to send packets from outbound neighbors to the node with the help of other nodes, which are in the middle of the path. If there is only one node in the middle of the path, we call this path *One-Relay path*. If there are two nodes in the middle of the path, we call this path *Two-Relay path*.

Next we informally define *No Relay algorithm*, *One-Step Relay algorithm* and *Two-Step Relay algorithm* used by applications to apply service from LRS. *No Relay algorithm is the algorithm, in which packet will not be relayed by other nodes.* This algorithm is the algorithm without using LRS. Every node only sends packets by himself. We will compare this algorithm with other algorithms and evaluate the improved performance for applications by using LRS. *One-Step Relay algorithm is the algorithm, in which a packet can be relayed one step.* In this algorithm, node can help other nodes relay packets across some asymmetric links. We will compare this algorithm with other algorithms and evaluate whether *One-Step relay* is good enough to help applications to alleviate asymmetric links in normal topology. *Two-Step Relay algorithm is the algorithm, in which a packet can be relayed two steps.* We will compare this algorithm with other algorithms and evaluate whether *Two-Step relay* is really better than two previous algorithms.

2.3.2 Framework and Protocol

To relay packets across asymmetric links, every node in the relay path will collaborate with each other to forward relay packets. We propose a link relay protocol for inter-node communication, and design a relay framework to implement the

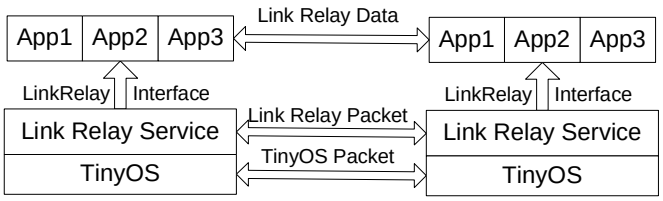

Figure 5: High level framework of the link relay service.

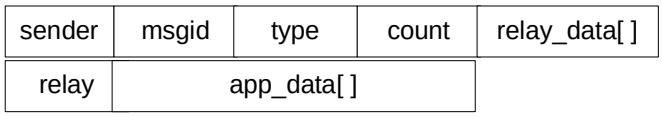

Figure 6: The generic packet format of the link relay service.

relay mechanism and provide an interface for applications to forward packets using relay policies.

Figure 5 is the high level overview of LRS framework, in which LRS is posed in its working context. There are three layers in this figure, TinyOS, LRS, and applications. LRS is an additional layer between TinyOS and applications. Nodes communicate with each other using TinyOS packets. When a relay packet is received by TinyOS, LRS will get the relay packet and forward the relay data to the associated application. When applications have relay data to forward, LRS will collect the relay data from applications, packet the relay data, and send the relay packet using TinyOS packets. In this framework, LRS provides mechanisms and applications make policy decisions. Next we will propose a link relay protocol for internode communication. We fist describe the packet format used in LRS.

Figure 6 is the generic packet format used by LRS. The *sender* field contains the node ID of the sender of this relay packet. The *mgsid* field contains the unique ID of every packet from the *sender* node. The value of the *msgid* will be increased for every new packet. The *type* field indicates the type of the relay data. In LRS, every *type* of the relay data is associated with one application and LRS uses the value of the *type* field to dispatch the relay data to the associated application. Usually one packet can contain more than one relay data. So there is the *count* field to store how many related data are contained in this packet. The *relay_data* field contains the relay data, which will be forwarded to the application. Usually the first field of the relay data is the *relay* field and the value of this field indicates how many relay steps this relay data can be relayed. The value of the *relay* field will be decreased by one for every relay step until it reaches zero. Applications will store its data in the *app_data* field. The whole packet is stored in the *data* field of a TinyOS packet, which only has 29 bytes. Therefore the length of the *app_data* field is no more than 24 bytes.

Link Relay Interface

```
interface LinkRelay {
  command  result_t subscribe(uint8_t size);
  command  result_t unsubscribe();
  event   result_t Relay(uint8_t  * data1);
  event   result_t RelayDone(TOS_MsgPtr ptr, result_t status);
  event   result_t Receive(uint8_t  * data1);
}
```

Figure 7: The LinkRelay interface.

2.3.3 Link Relay Interface

LRS provides the *LinkRelay* interface for applications to relay packet across asymmetric links. In this section, we will describe how to use this interface and the working mechanism of the link relay service.

Figure 7 lists the *LinkRelay* interface. This interface is a parameterized interface, which can support 256 applications concurrently. Every application will be associated with an unique ID. This ID will be store in the $type$ field of the relay packet. Two commands and three events are provided in this interface. When an application wants to use LRS to relay packet, it first call the *subscribe* command. The parameter of this command is the size of the relay data, which is no more than 24 bytes. When a relay packet arrives this node, the associated application will get the *Receive* event from LRS. The parameter of this event is the relay data. At this time, the application can make decision about how to process this incoming relay data. When the radio channel is available to send data, LRS will signal a *Relay* event to get the relay data from applications. At this time, the application can make decision about which relay data will be relayed and how many steps the relay data will be relayed. When the relay packet is sent, the *RelayDone* event will be signaled to the application to notify the status of the relay packet.

If the application is not interested in the LRS events any more, it can call the *unsubscribe* command to unsubscribe these events. Therefore, when new relay packet is received by LRS, it will be discarded without notifying the application. Figure 8 is the pseudocode of the algorithm for the working mechanism of the link relay service.

3 Implementation

As we have discussed, NLQL provides the timeliness link quality information of inbound and outbound neighbors to other applications and LRS implements a relay framework to store and forward the relay data across asymmetric links. In the implementation, NLQS uses the *LinkMeasure* interface provided by LRS to get the packet reception information from inbound neighbors to measure PRR and leverages WMEWMA to estimate PRR. Furthermore, NLQS uses the *LinkRelay* interface provided by LRS to relay the estimated link quality of

Algorithm 2 Working mechanism of the link relay service

Procedure Receive-Relay-Packet ()
Begin Procedure

1: get the relay packet from TinyOS
2: parse the $type$ field of the relay packet to find the associated application
3: **if** the application has $subscribe$d the link relay service **then**
4: Signal the $Receive(data)$ event to the application
5: **else**
6: Drop the relay packet
7: **end if**

End Procedure

Procedure Forward-Relay-Packet ()
Begin Procedure

1: **repeat**
2: choose an application which has $subscribe$d relay service
3: signal the $Relay(data)$ event to the application to collect relay data
4: **until** (get relay data $\vee$ no application has relay data to forward)
5: **if** have relay data to forward **then**
6: forward the data out
7: **if** forward the relay data successfully **then**
8: signal the $RelayDone(SUCCESS)$ event to the application
9: **else**
10: signal the $RelayDone(FAIL)$ event to the application
11: **if** the application need relay again **then**
12: forward the relay data again and repeat this process
13: **else**
14: drop the relay data
15: **end if**
16: **end if**
17: **end if**

End Procedure

Figure 8: The pseudocode of the link relay service.

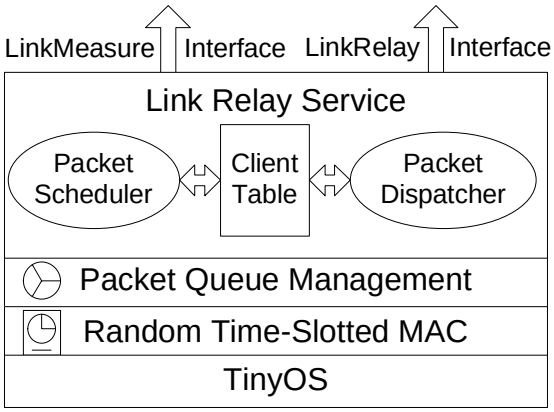

Figure 9: The architecture of the link relay service.

neighbors across asymmetric links to facilitate the identification of outbound neighbors.

3.1 Link Relay Service

LRS overhears the radio channel for relay packets. When LRS receives the relay packet, it will dispatch it to the associated application. LRS also schedules applications to forward relay packets. In the process, LRS needs to store those received packets and relays packets. We implement a packet queue to manage packets in LRS. To avoid collisions of simultaneous packets transmissions between neighbors, we implement a random time-slotted MAC protocol in LRS. Next we will present the architecture of LRS and describe its implementation details.

Architecture of LRS Figure 9 shows the architecture of LRS, which consists of three layers. The upper layer is the packet scheduler and dispatcher. The packet dispatcher dispatches the relay data to applications. The packet scheduler is responsible for scheduling applications to forward relay data. The middle layer is the packet queue management layer, which manages packets received from the TinyOS platform (in) and created by applications (out) of LRS. The random time-slotted MAC is in the lowest layer. This MAC protocol is designed to avoid collisions of simultaneous packets transmissions between neighbors.

Packet Dispatcher and Packet Scheduler As we have discussed, a relay framework is implemented in LRS and the *LinkRelay* interface is provided by LRS. The *LinkRelay* interface is a parameterized interface, which can support 256 applications concurrently. Every application is associated with an unique ID in this interface. The association is implemented in the compiler stage following the same mechanism as the active message in TinyOS. Applications can subscribe LRS in order to apply service from LRS. The subscription and schedule information of applications is stored in the client table. This associated ID of applications is also stored in the *type* field of the relay packet. When LRS receives a relay packet, it will post a packet dispatcher task to dispatch the received packet. The

```
Packet Queue Interface
interface PacketQueue {
  command TOS_MsgPtr exchange(TOS_MsgPtr ptr, uint8_t newstate);
  command TOS_MsgPtr getUpdateState(uint8_t oldstate, uint8_t newstate);
  command TOS_MsgPtr getByState(uint8_t oldstate);
  command result_t   setState(TOS_MsgPtr ptr, uint8_t newstate);
  command void       printState();
}
```

Figure 10: The PacketQueue interface.

packet dispatcher parses the *type* field of the relay packet and dispatches the relay data to the associated application. When LRS forwards the relay data, it will post a packet scheduler task.

Packet Queue Management in LRS When TinyOS receives a packet, it will forward the packet buffer to applications. The applications should return this packet buffer or another packet buffer to TinyOS immediately. If the application holds the packet buffer for a long time, TinyOS will fail to receive new packet from the radio channel. In this case, more packets are lost. So it is necessary to design and implement a packet queue management in LRS, and support different relay policies.

Figure 10 lists the interface provided by the packet queue management. We describe the command in the following. When TinyOS receives a relay packet, LRS will use the *exchange* command to exchange the packet buffer in TinyOS with a free packet buffer in the packet queue and set the received packet in a special state for later use. TinyOS then uses the switched packet buffer to receive new packet immediately. This approach will reduce the packet loss dramatically. Later, LRS can use the *GetUpdateState* or the *GetByState* command to get the packet from the packet queue to process it. If LRS has processed the packet, it can use the *setState* command to set the packet to new state. If the new state is *S_RELAY_WAIT_TO_SEND*, the packet will be forwarded later. If the new state is *S_FREE*, this packet will be freed. All these commands are processed as an atomic action. This is a general interface and applications can integrate their new states for the packet into the interface easily.

Random Time-Slotted MAC Protocol in LRS To avoid collisions of simultaneous packets transmissions between neighbors, we implement a random time-slotted MAC protocol. The MAC protocols works as follows. Every periodic time interval is divided into a certain time slots. Every time slot is long enough to send one packet. The number of time slots depends on the density of neighbors. In our current implementation, the periodic time interval is 2000 milliseconds, and the number of time slots in every periodic time is 20. When it is time to send packet, the MAC will post a packet scheduler task to schedule applications to relay data. The collected relay packet is stored in the packet queue. Then the MAC randomly selects a time slot and sends the packet in that time slot. To implement the MAC protocol, we define a periodic timer and one-short timer. The periodic timer is fired for every time interval. The on-short timer is fired for the randomly selected time slot. If there is a collision, the packet will corrupt and lost. The up-level application will handle this. Currently we implement the random time-slotted MAC protocol based on mica radio stack [13]. After many simulations, we find this simple MAC protocol can reduce most collisions in the link quality measurement process.

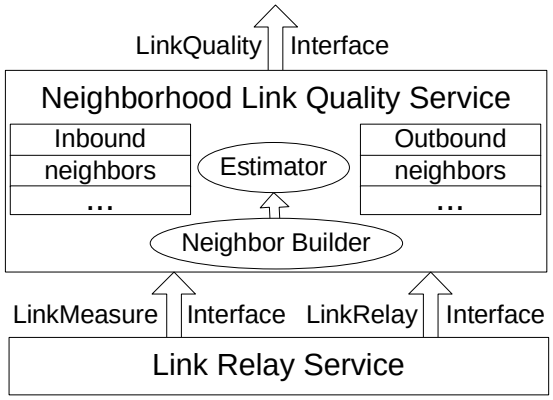

Figure 11: The architecture of the neighborhood link quality service.

sender	total_msg	window	window_msg	prr	policy

Figure 12: The data structure of the inbound neighbor table.

3.2 Neighborhood Link Quality Service

In this section, we will describe the implementation of NLQS and the interaction between NLQS and LRS.

Architecture of NLQS Figure 11 shows the architecture of NLQS, which consists of four components: the inbound neighbor table, the outbound neighbor table, the neighbor builder, and link quality estimator. The inbound neighbor table contains the link quality information of inbound neighbors and other data used by WMEWMA. The outbound neighbor table contains the link quality information of outbound neighbors and the relay data for other nodes. The neighbor builder builds the inbound neighbor table using PRR estimated by WMEWMA and builds the outbound neighbor table using PRR received from LRS.

Data Structure and Packet Format The data structure of the inbound neighbor table is illustrated in Figure 12. The *sender* field contains the node ID of the inbound neighbor. The *total_msg* field contains the number of total packets received from the *sender*. When the node gets a packet from the *sender*, it will get the ID of the packet. Using the packet ID and the value of the *total_msg* field, the node calculates PRR of the link from the *sender* for long term. The *window* field contains the number of the time window, which is used in WMEWMA. The *window_msg* field contains the number of packets received in this time window. Using the ID of the received packet and the value of the *window_msg*, the node can calculate PRR in

relay	sender	receiver	window	prr	policy

Figure 13: The data structure of the outbound neighbors table.

relay	sender	receiver	window	prr

Figure 14: The packet format of the active probing packet.

this time window. When Link quality estimator does the estimation, it will read the previous value of PRR from the *prr* field and store the estimated PRR in this field again. The *policy* field is used for policy decision. We detain the discuss of different policies in Section 4.3.

Figure 13 illustrates the data structure of the outbound neighbors table. The *sender* field contains the node ID of the sender and the *receiver* field contains the node ID of the receiver. The *window* field contains the time window number and the *prr* field contains PRR of the link from the *sender* to the *receiver* for this time window. If the *sender* is the node itself, the *receiver* is one of the identified outbound neighbor. Otherwise, this data is a relay data. The *relay* field contains remaining relay steps of this packet. The *policy* field is used for policy decision. The neighbor builder can use the *relay* and the *policy* field to make a decision about whether to relay this data.

As we have discussed, we will use the combination of active probing and passive overhearing techniques to do the link quality measurement and estimation and the estimated PRR will be sent in the active probing packet.

Figure 14 is the packet format of the active probing packet. This data follows the link relay protocol. The *relay* field contains the number of relay steps of this packet. Other fields indicate that estimated PRR from the *sender* node to the *receiver* node is the value of the *prr* field in the *window* time window. The node can use this packet to send the estimated link quality of its inbound neighbors and also relay the link quality for others. When the node receives this packet, it will update outbound neighbors according to PRR in the packet.

Using LinkMeasure and LinkRelay Interface To measure and estimate the link quality between neighbors, NLQS needs to know the packet reception information from inbound neighbors. This information is provided by LRS using the *LinkMeasure* nterface, as shown in Figure 15. When LRS receives a packet, it will get the sender ID and the packet ID and signal this event to applications. Currently, NLQS uses *LinkMeasure* interface to measure the link quality. The neighbor builder will update the data in inbound neighbors according to this information. When the neighbor builder finds that it is time to do the estimation for this inbound neighbor, it will create a WMEWMA estimator task to do this work.

To relay the estimated PRR across asymmetric links, NLQS uses the *LinkRelay* interface provided by LRS. In this case, the value of the *relay* filed is 0, 1, or 2, which stands for *No Relay*, *One-Step Relay*, and *Two-Step Relay* algorithms, respectively. The *sender* field contains the node ID of the inbound neighbor. The *receive* field contains its node ID. The neighbor builder will decide which relay data in the outbound neighbor table will be sent again. It first checks whether the value of the *relay* filed is zero, which means no need to relay this data again. If the value of the *relay* field is larger or equal to one, the packet should be relayed one more time. To assure the estimated PRR can be received by neighbors, the neighbor builder relays it many times. In current implementation, the neighbor builder will continue send the estimation PRR using the probing packet until it sends enough packets during the current time window. The neighbor builder sends the estimated PRR in a round-robin way.

Link Measure Interface

```
interface LinkMeasure {
  event   result_t Receive(uint16_t moteid, uint16_t msgid);
}
```

Figure 15: The LinkMeasure interface.

4 Example Applicatons

To demonstrate and evaluate link quality services, we design and implement two example applications, *the shortest hops routing tree* (SHRT) and *the best path reliability routing tree* (BRRT), in the TinyOS platform. In SHRT, every node selects a parent node, which has a routing path with shortest hops to the sink node. In BRRT, every node selects a parent node, based on which the node can build a best reliability path to the sink node. We define the path reliability for every node as follows:

$$Reliability_A = \prod_{A}^{S} PRR \tag{3}$$

where the $Reliability_A$ is the path reliability for node A. The PRR is the PRR of one of links along the routing path from node A to the sink node S. For example, in Figure 16, node 0 is the sink node and the routing path of node 5 is $0.84 * 0.93 = 0.78$.

In both SHRT and BRRT, the sink node floods a packet to start the process of building a tree. In SHRT, the routing tree information contained in the packet includes the node ID and the count of hops to the sink node. In BRRT, the routing tree information contained in the packet includes the node ID and its path reliability to the sink node. Every node will choose its parent to build SHRT or BRRT when it receives the flooding packet. However, due to the link asymmetry, nodes have more chances to build a broken routing tree when it chooses the parent by considering the information embedded in the received packet only. For example, in Figure 16, if node 6 gets a flooding packet from node 1, node 6 will choose node 1 as its

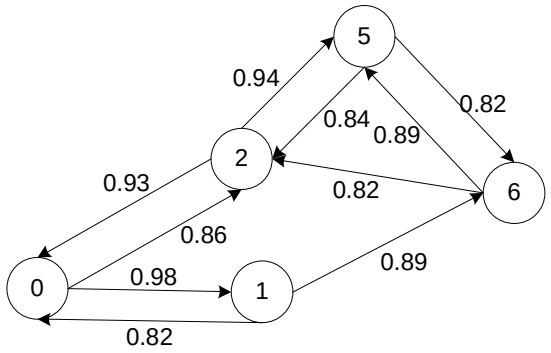

Figure 16: A topology to illustrate SHRT and BRRT.

parent. Unfortunately, as shown in the figure, node 1 can not receive packet from node 6. So when node 6 chooses node 1 as its parent, it will build a broken routing tree.

The link asymmetry also has negative effects on building a good routing tree. For example, in Figure 16, node 2 is the best option for node 6's parent. But due to asymmetric links, node 6 can not get flooding packet from node 2. In this topology, node 6 can get flooding packet from node 5 and choose node 5 as its parent. In this case, node 6 needs one more hop to route packets to the sink node. However, if node 2 is able to relay its flooding packet across asymmetric links using link quality services, node 6 will get the flooding packet and will choose node 2 as its parent. From this example, we know that if the node is able to relay the flooding packet across asymmetric links, other nodes will have more chances to choose better parent. Thus, we will evaluate *No Relay*, *One-Step Relay*, and *Two-Step Relay* algorithms in the context of SHRT and BRRT. In the next two sections, we describe the details of how to build the routing tree by using link quality services.

4.1 Building Shortest Hops Routing Tree

In SHRT, every node initializes its hops to the sink node as the maximum value and set its parent empty. The sink node floods a packet, which contains its ID and its hop count. In our simulated topology, the node ID of the sink node is 0 and its hop count is 0 in the flooding packet. Other node receives the flooding packet, it will update its parent according to the routing tree information in the flooding packet. Figure 17 lists the pseudocode of the algorithm. In this algorithm, node receives flooding packet from the $Receive$ event signaled by LRS. It will use NLQS to check whether the node, P, in the flooding packet is its outbound neighbor. If the hops of node P is shorter than that of its current parent, the node will choose node P as its parent and update its hops.

4.2 Building Best Path Reliability Routing Tree

In BRRT, every node initializes its path reliability as zero and set its parent empty. The sink floods a packet, which contains its ID and its path reliability. In our simulated topology, the node ID of the sink node is 0 and its path reliability is 1.0 in the flooding packet. Other node receives the flooding packet, it will update its parent according to the routing tree information in the flooding packet. Figure 18 shows the pseudocode of the algorithm. The procedure is very similar to the SHRT algorithm. The following equation is used to calculate the path reliability based on node P.

$$Reliability_{A,P} = Reliability_P * PRR_{A,P} \quad (4)$$

where the $Reliability_{A,P}$ is the path reliability of node A based on node P. The $Reliability_P$ is the path reliability of node P, which is obtained from the flooding packet. The $PRR_{A,P}$ is the PRR of the link from node A to node P, which can be easily obtained through the *LinkQuality* interface of NLQS.

Algorithm 3 Building the shortest hops routing tree

Event Receive ($parent$)
Begin Event
1: **if** is-outbound-neighbor($parent$) **then**
2: **if** is-shorter-hops-based-on-it($parent$) **then**
3: set the $parent$ node as the current parent and update its hops to the sink node as $parent.hops + 1$
4: **end if**
5: **end if**
6: **if** is-shorter-hops-than-before(parent) **then**
7: update the relay data for the $parent$ node
8: **end if**
End Event

Figure 17: The pseudocode of building SHRT.

Algorithm 4 Building the best path reliability routing tree

Event Receive ($parent$)
Begin Event
1: **if** is-outbound-neighbor($parent$) **then**
2: **if** is-better-reliability-based-on-it($parent$) **then**
3: set the $parent$ node as the current parent and update path reliability with calculated value based on the $parent$ node
4: **end if**
5: **end if**
6: **if** is-better-reliability-than-before(parent) **then**
7: update the relay data for the $parent$ node
8: **end if**
End Event

Figure 18: The pseudocode of building BRRT.

4.3 Packet Retransmission Policies

In both SHRT and BRRT, the flooding packets has a high probability to be lost during the forwarding. To make sure most nodes can receive the flooding packet, we define three policies to retransmit the flooding packet.

1. **Transmit the routing tree information twice (P2)** When a node updates its parent, it will flood its routing tree information twice. In *No Relay* algorithm, the flooding packet contains only the routing tree information of the flooding node. In *One-Step Relay* and *Two-Step Relay* algorithms, the relay data is attached in the flooding packet. This policy will not flood the relay data individually.

2. **Transmit the routing tree information twice and transmit the relay data once (P2+R1)** The difference of this policy with the *P2* Policy is that this policy will flood the relay data at least once when it is updated in *One-Step Relay* and *Two-Step Relay* algorithms. Every flooding packet contains two relay data if there is enough relay data to be relayed.

3. **Transmit the routing tree information twice and transmit the relay data twice (P2+R2)** The difference of this policy with the *P2+R1* Policy is that this policy will flood the relay data twice when it is updated in *One-Step Relay* and *Two-Step Relay* algorithms.

From these policies, we can find that *P2* will send out the least packets and *P2+R2 Policy* will send out the most packets. The more the packet is sent, the higher possibility the packet is received. We will evaluate three polices in the simulation using TOSSIM.

5 Performance Evaluation

After describing the link quality services and example applications, we are now in the position to evaluate the performance of our link quality services. First, four performance metrics are proposed. Then, based on proposed performance metrics, we conduct a static analysis to calculate the optimal results of four performance metrics in an ideal network without packet loss, and compare it with the simulation using TOSSIM. In the simulation, we simulate *No Relay*, *One-Step Relay*, and *Two-Step Relay* algorithms with *P2 Policy*, *P2+R1 Policy*, and *P2+R2 Policy* using TOSSIM.

5.1 Performance Metrics

To evaluate our asymmetry-aware link quality services, we propose and define the following performance metrics. One of them is used to evaluate link quality services directly, the others are used to evaluate the improved performance of example applications.

Number of Increased Outbound Neighbors Due to asymmetric links, some nodes will not receive acknowledgements from their outbound neighbors and these outbound neighbors can not be identified. Using our link quality services,a neighborhood discovery process will identify more outbound neighbors. In Section 2.3.1, we have defined *No Relay*, *One-Step Relay* and *Two-Step Relay* outbound neighbors. Those outbound neighbors are the optimal results for *No Relay*, One-Step Relay, and Two-Step Relay algorithms. We will build those outbound neighbors for every node in the generated topology using both the static analysis and simulation.

Number of Reduced Hops in SHRT In SHRT, every node will choose a parent, base on which the node can build a shortest hops routing path to the sink node. Due to asymmetric links, some nodes in SHRT can not get flooding packet from the parent. By building SHRT with *No Relay*, One-Step Relay, and Two-Step Relay algorithms, we can get the hops of every node in the routing tree for these algorithms in both static analysis and simulation.

Path Reliability Improvement in PRRT In PRRT, every node will choose a parent, based on which the node can build a best reliability routing path to the sink node. With the support of link quality services, we can build a more reliable routing tree. By building PRRT with *No Relay*, One-Step Relay, and Two-Step Relay algorithms, we can compute the path reliability of every node in the routing trees in both static analysis and simulation.

Energy Consumption in SHRT and PRRT In link quality services, every node will broadcast a certain number of packets to measure the link quality between neighbors. So we need not compare the energy consumption in this process. We only compare the energy consumption resulting from applications, which build the routing tree with *No Relay*, One-Step Relay and Two-Step Relay, algorithms. Link quality services can alleviate effects of link asymmetry in the tree building process, but it also need to send more packets to relay packets across asymmetric links. The more packets is relayed, the more information will be shared, but the more energy will be consumed. That is a trade off. We will evaluate the energy consumption in term of total number of receiving and sending packets in each algorithm.

5.2 Simulation Setup

In this part, we describe how to setup the simulation environment. TOSSIM [6] is an emulation tools for TinyOS platform, which is widely used in WSN research area. We simulate link quality services using TOSSIM in the lossy model to get performance results. To do the simulation, we specify a lossy file of the simulated topology for TOSSIM. The bit error rate of links in the topology will be listed in the lossy file. We totally simulate 156 Motes. The PRR of links are randomly chosen from the real measurements using Berkeley MICA2 motes in a controlled environment.

5.3 Evaluation Results

In this part, we will present evaluation results of link quality services according to performance metrics.

5.3.1 Increased Outbound Neighbors

# of Increased Outbound Neighbors	0	1	2	3
One-Step Relay Analysis	93	52	8	3
One-Step Relay Simulation	94	51	8	3
Two-Step Relay Analysis	83	61	7	5
Two-Step Relay Simulation	85	61	6	4

Table 1: The distribution of increased outbound neighbors.

Table 1 lists the distribution of increased outbound neighbors in both analysis and simulation, including how many nodes identify how many more outbound neighbors using *One-Step Relay* and *Two-Step Relay* algorithms than *No Relay*. For example, in this table the *One-Step Relay* algorithm in the simulation help 51 nodes identify one more outbound neighbor and 8 nodes identify two more outbound neighbors than *No Relay*. From this table, we can see that performance results in the simulation and those in the analysis match very well. For *One-Step Relay* algorithm, only one node fails to identify one of its outbound neighbors in the simulation. For example, 51 nodes have identify one more outbound neighbor in the simulation while the number of this kind of node is 52 in the analysis. For *Two-Step Relay* algorithm, only two nodes fail to identify more outbound neighbors in the simulation. For example, 6 nodes identify two more outbound neighbors and 4 nodes identify three more outbound neighbors while the number of these kinds of nodes are 7 and 5 separately in the analysis. From this table, we also can see that the performance of *Two-Step Relay* is better than *One-Step Relay*. More than 10 nodes identify one more outbound neighbors in *Two-Step Relay* than those in the *One-Step Relay* algorithm. For example, 61 nodes identify one more outbound neighbors in *Two-Step Relay* algorithm while the number of this kind of node is only 51 in the *One-Step Relay* algorithm.

Figure 19 shows the cumulative distribution function for the percentage of increased outbound neighbors. In this figure, the x-axis stands for the percentage of increased outbound neighbors for every node and the y-axis stands for the percentage of those nodes. Results of the *One-Step Relay* and *Two-Step Relay* algorithms in the analysis and simulation are shown together to facilitate comparison. From this figure, we can see that more than 40% of nodes identify more outbound neighbors in the *One-Step Relay* algorithm and the percentage of increased outbound neighbors is from 17% to 100%. Alao we can find that more than 45% of nodes identify more outbound neighbors in the *Two-Step Relay* algorithm and the percentage of increased outbound neighbors is from 14% to 100%. Note that the analysis results is a little better than those in the simulation. The reason for this is some acknowledgement packets are lost due to loss links in the simulation. Therefore some

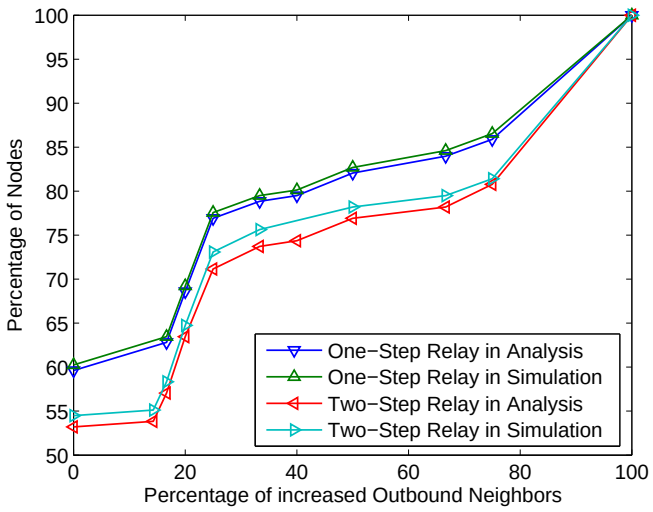

Figure 19: The CDF of the percentage of increased outbound neighbors.

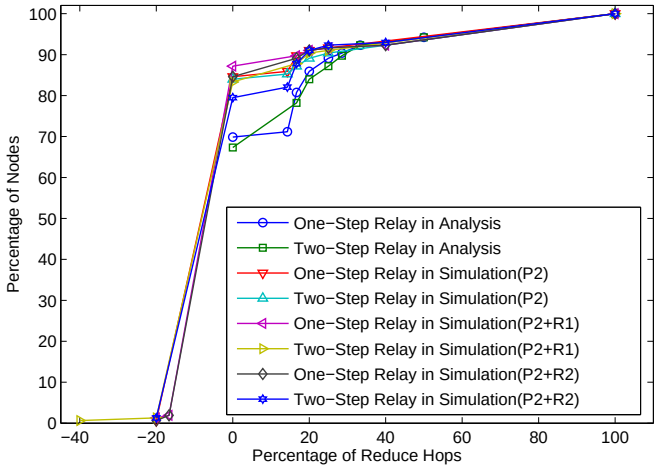

Figure 20: The CDF of the percentage of reduced hops.

nodes fail to identify those outbound neighbors. The *Two-Step Relay* algorithm is better than *One-Step Relay* as we expected.

5.3.2 Reduced Hops in SHRT

The performance metrics of reduced hops for every node in SHRT is used to evaluate link quality services by comparing the routing tree built using *No Relay*, *One-Step Relay*, and *Two-Step Relay* algorithms. With the help of link quality services, we expect to build a better SHRT, in which every node has shorter hops to the sink node.

Figure 20 reports the CDF of the percentage of reduced hops for every node in the simulation. To see the details of the results in the middle of Figure 20, we zoom in the results between 0% to 20%, as shown in Figure 21. In these figures, the x-axis is the percentage of reduced hops for every node and

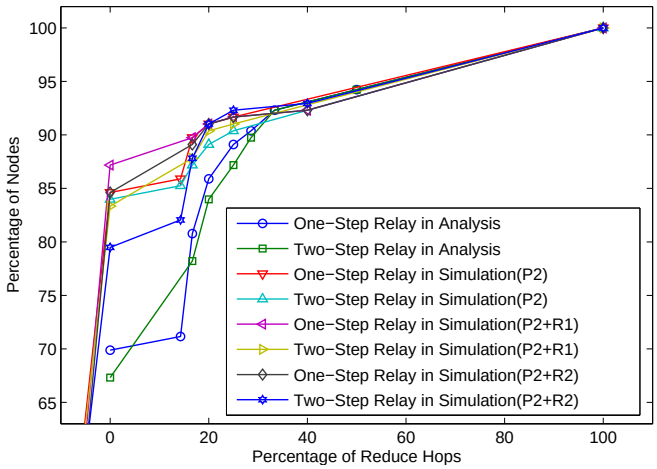

Figure 21: Zooming of the CDF of the percentage of reduced hops.

the y-axis shows the percentage of those nodes.

From these figures, we can see that the results in the simulation do not match well with those in the analysis. In the analysis, there are more than 30% of nodes reduce their hops to the sink node and the percentage of reduced hops is mainly between 20% to 40%. In the simulation results, there are only more than 15% of nodes that reduce their hops to the sink node and the percentage of reduced hops is mainly between 15% and 25%. The *P2+R2* policy of *Two-Step Relay* is the best algorithm and more than 20% of nodes reduce their hops to the sink node. While the *P2+R1* Policy of *One-Step Relay* is the worst one of all algorithms and only more than 13% of nodes reduce their hops to the sink node.

Again, we also observe that the results in the simulation are worse than those in the analysis, because of the packet loss in the simulation. If one node near the sink node fails to choose the best parent, all nodes, which include this node in their routing path, fail to reduce their hops. It is a common phenomenon in WSN that the performance of the node near the sink node has more impacts on the performance of the whole system. Some nodes increase hops to the sink node in simulations. The reason is some packets are lost due to loss links and collisions of packet transmissions. But this probability is very low. Therefore, only less than 2% of nodes choose a worse parent. From the comparison of results, we argue that a better policy to flood packets is needed to improve the performance.

5.3.3 Improved Path Reliability in BRRT

Figure 22 reports the CDF of the percentage of the improved path reliability for every node in the simulated topology. Figure 23 is the zooming of the middle part of Figure 22. In these figures, the x-axis stands for the percentage of the improved path reliability for every node and and the y-axis stands for the percentage of those nodes. In the simulation, more than 15% of nodes improve the path reliability and the percentage of the

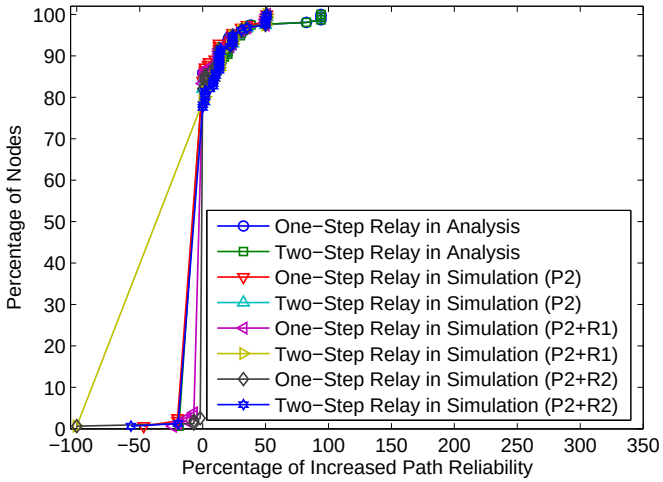

Figure 22: The CDF of the improved path reliability.

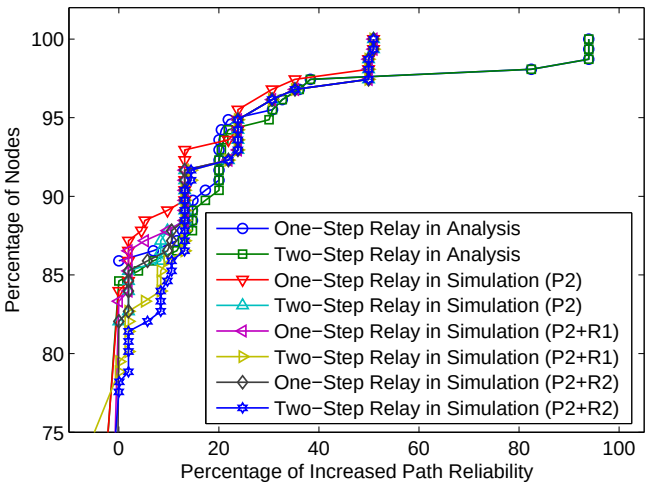

Figure 23: Zoom in of the CDF of the improved path reliability.

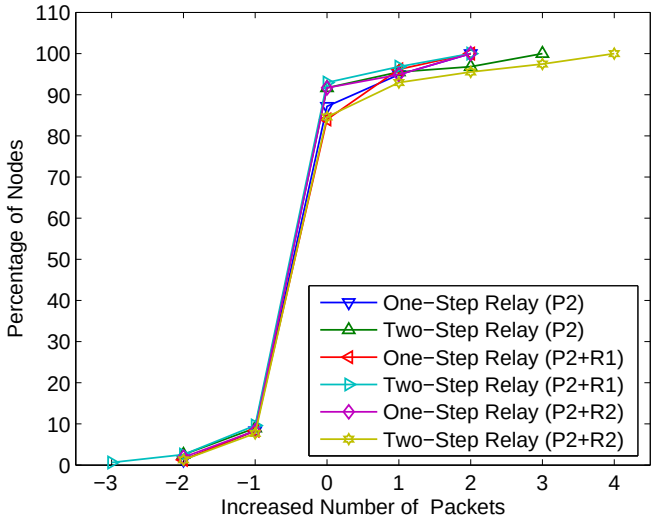

Figure 24: The CDF of more packets sent in SHRT.

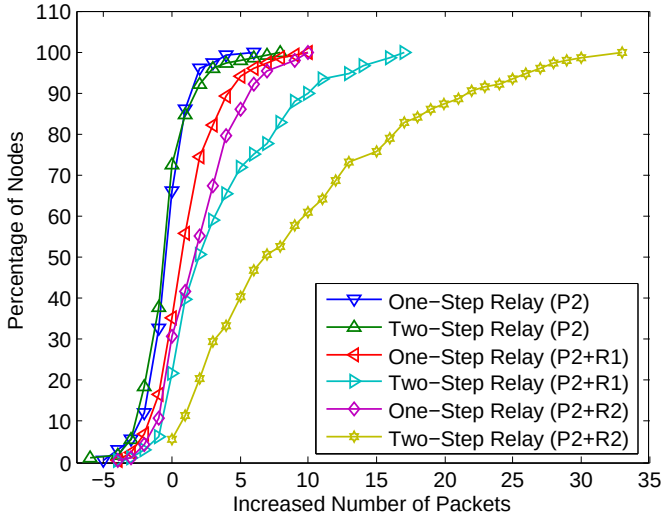

Figure 25: The CDF of more packets received in SHRT.

improved path reliability is mainly between 2% and 40% and the percentage of some nodes even reach on 50%. The combinationo of the *P2+R2* policy with the *Two-Step Relay* algorithm is the best one, where more than 27% of nodes improve the path reliability to the sink node and the percentage of the improved path reliability focus on 15% to 22% and 50%. The *P2* policy with the *One-Step Relay* algorithm is the worst one of all algorithms. Also we note that there are a lot of nodes improve the path reliability than the number in SHRT. The reason is that the difference between BRRT and SHRT. Nearly every neighbor of a node in BRRT has different path reliability while most neighbors of a node in SHRT have same hops to the sink nodes. Therefore, nodes in BRRT have more opportunities to change their parents than they do in SHRT.

5.3.4 Energy Consumption in SHRT and BRRT

In this part, we will evaluate the energy consumption in SHRT and BRRT.

Energy Consumption in SHRT Figure 24 shows the CDF of more packets sent by the *One-Step Relay* and *Two-Step Relay* algorithms than *No Relay*. In this figure, the x-axis stands for more packets send by every node and the y-axis reports the CDF. From this figure, we can see that all algorithms have little difference in terms of the number of extra packets sent. For the *One-Step Relay* and *Two-Step Relay* algorithms, about 90% of nodes do not send more packets and about 10% of nodes send less packets.

Figure 25 shows the CDF of more packets received by the *One-Step Relay* and *Two-Step Relay* algorithms than *No Relay*. In this figure, the x-axis stands for more packets received by every node. From this figure, we can see the following results. The *P2+R2* policy of *Two-Step Relay* receives more packets than others. The *P2+R1* of *Two-Step Relay* is the second worst. The *P2* policy of *One-Step Relay* and the *P2* policy of *Two-Step Relay* follow the similar pattern and they receive

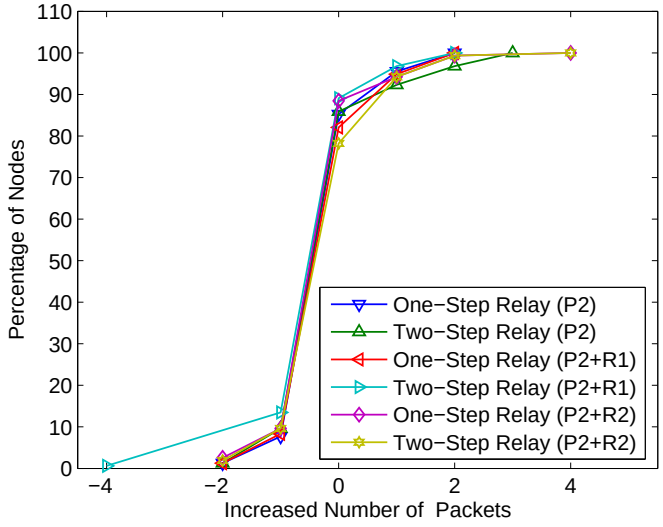

Figure 26: The CDF of more packets sent in BRRT.

less packets than other algorithms. Except the *P2+R2* policy of *Two-Step Relay*, most algorithms have nearly 20% of nodes which receive less packets than the *No Relay* algorithm.

Energy Consumption in BRRT Figure 26 and Figure 27 present the CDF of more packets sent and received by the*One-Step Relay* and *Two-Step Relay* algorithms than *No Relay*, respecitively. In these figures, the x-axis stands for more packets sent/received by every node. From Figure 26, we can see that all algorithms have no big difference with the number of sending packets. From Figure 27, we can see the similar results in SHRT. The combination of the *P2+R1* policy and the *Two-Step Relay* algorithm receives the most packets.

The results in this section show that there is a tradeoff between the energy consumption and the benefits brought by the link relay service. We argue that it is the application's decision to turn on/off the link relay service. For example, if the path re-

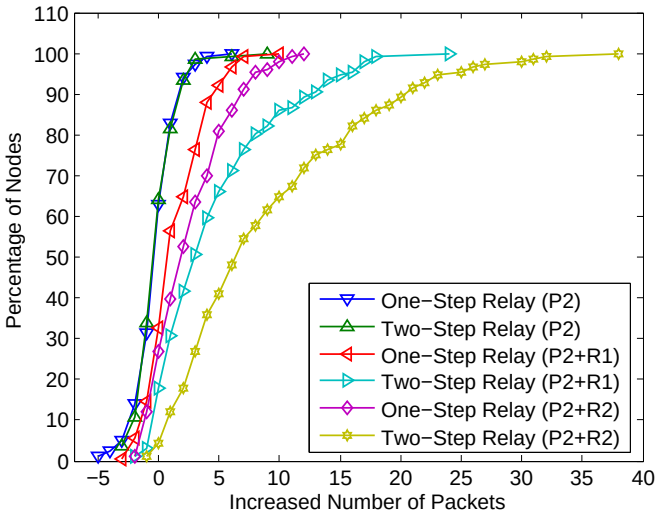

Figure 27: The CDF of more packets received in BRRT.

liability or path length is more important than the energy, e.g., event notification, then it should choose the link relay service. However, for monitoring-based applications it's better to turn off this service.

6 Related Work and Discussions

Our research work has been inspired by a variety of previous research work. Instead of describing them individually, we classify them into three groups: *link layer characterization*, *link quality estimation*, and *existing solutions to link irregularity*.

Link layer characterization Ganesan *et al.* do an experimental study of low-power WSN in a large scale. They find the problem in large scale, such as long links, backward links, stragglers and clustering [3]. And they also find the bidirectional link and asymmetric links is common in large scale. Zhao *et al.* measure the spatial and temporal characteristics of packet delivery [18]. They find the gray area in which there are significant variability in packet delivery performance and the no-determining relationship between signal strength and packet delivery. They also find the asymmetry in packet delivery and explain some cause of it. Zhou *et al.* show that radio irregularity is a common phenomenon which arises from multiple factors, such as variance in RF sending power, and different path losses depending on the direction of propagation [19]. All of these research work shows us the reality of the link quality in WSN, which make us have a deeper understanding of the wireless communication in WSN. Our research is inspired by those previous efforts, and is a follow-up which provides solutions to link asymmetry. Also, we are the first to provide a set of APIs for link quality services. We plan to release our implementation to the TinyOS community soon.

Link quality estimation. Kim *et al.* present a couple of estimators for wireless communication and evaluate them using different performance metrics [4]. Woo *et al.* [15] declare that link connectivity statistics should be captured dynamically through an efficient yet adaptive link estimator and routing decisions should exploit such connectivity statistics to achieve reliability. In another research paper [14], Woo *et al.* show that WMEWMA is a very effective estimator, in term of stability and agility. Cerpa *et al.* present a statistical model of lossy links in WSN [2]. They intend to model lossy links with statistical method of the real measurement data. Zhou *et al.* establish a radio model for simulation, called the Radio Irregularity Model (RIM) [19]. Zuniga *et al.* intend to identify the causes of the transitional region, and a quantification of their influence [20]. They also incorporate important channel and radio parameters such as the path loss exponent, shadowing variance of the channel, and the modulation and encoding of the radio. Different from these previous work, we focus on the asymmetry-aware link quality services that alleviate the effect of asymmetric links in the link quality measurement and estimation process, which is neglected in previous research.

Existing solutions to link irregularity Woo *et al.* study and evaluate the neighborhood table management and reliable routing protocol techniques [15]. They also evaluate the frequency based table management, and cost-based routing. Seada *et al.* present energy-efficient forwarding strategies for geographic routing in lossy WSN [7]. Their work is to find optimal geographic forwarding in the lossy link. They results also show that reception based forwarding strategies are more efficient than purely distance-based strategies; relatively blacklisting schemes reduce disconnections and achieve higher delivery rates than absolute blacklisting schemes. Ramasubramanian *et al.* provide a bidirectional abstraction of the unidirectional network to routing protocols in *ad hoc networks* [12]. These efforts face the reality and try to solve the real problem to provide better services in WSN. We step further to distinguish inbound and outbound neighbors to make good use of outbound neighbors for high level routing service, and to provide a set of link quality interfaces. We also provide LRS, which consists of a link relay protocol, a link relay framework, and an interface for relaying packets across asymmetric links.

7 Conclusions and Future Work

In this paper, we design, implement and evaluate the proposed asymmetry-aware link quality services, which include the neighborhood link quality service and the link relay service. NLQS distinguishes inbound and outbound neighbors and provide the *LinkQuality* interface to query the timeliness link quality information. In LRS, we propose a link relay protocol, implement a link relay framework, and provide *LinkRelay* interface to relay packets across asymmetric links. The performance of link quality services is evaluated in the context of two example applications. From both the static analysis and simulation using TOSSIM, we find the performance of two example applications improve substantially using the

proposed link quality services.

Our future work includes three directions. First, evaluate link quality services in a large scale real deployment of WSN. Second, improve algorithms to relay packet in a more adaptive and energy efficient way. Third, apply the proposed link quality services to high level routing protocols, such as WEAR [8].

References

[1] M. Batalin et al. Call and responses: Experiments in sampling the environment. In *Proc. of ACM SenSys 2004*, Nov. 2004.

[2] A. Cerpa, J. Wong, L. Kuang, M. Potkonjak, and D. Estrin. Statistical model of lossy links in wireless sensor networks. Technical Report CENS Technical Report, University of California, Los Angeles, Apr. 2004.

[3] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report UCLA/CSD-TR-02-0013, University of California, Los Angeles, Feb. 2002.

[4] M. Kim and B. Noble. Mobile network estimation. In *Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'01)*, June 2001.

[5] P. Levis et al. The emergence of networking abstractions and techniques in tinyos. In *Proceedings of the First USENIX/ACM Networked System Design and Implementation*, Mar. 2004.

[6] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First ACM SenSys'03*, Nov. 2003.

[7] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proc. of ACM SenSys 2004*, Nov. 2004.

[8] K. Sha, J. Du, and W. Shi. Wear: A balanced, fault-tolerant, energy-aware routing protocol forwireless sensor networks. Technical Report MIST-TR-2005-001, Wayne State University, Jan. 2005.

[9] G. Simon, A. Ledeczi, and M. Maroti. Sensor network-based countersniper system. In *Proc. of ACM SenSys 2004*, Nov. 2004.

[10] R. Szewczyk et al. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, June 2004.

[11] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. of ACM SenSys 2004*, Nov. 2004.

[12] R. C. V. Ramasubramanian and D. Mosse. Providing a bidirectional abstraction for unidirectional ad hoc networks. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'02)*, June 2002.

[13] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'01)*, Rome, Italy, July 2001.

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

[15] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the First ACM SenSys'03*, Nov. 2003.

[16] N. Xu et al. A wireless sensor network for structural monitoring. In *Proc. of ACM SenSys 2004*, Nov. 2004.

[17] H. Zhang and J. Hou. On deriving the upper bound of α-lifetime for large sensor networks. In *Proceedings of the Fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, May 2004.

[18] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the First ACM SenSys'03*, Nov. 2003.

[19] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proc. of ACM MobiSys'04*, June 2004.

[20] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *The First IEEE International Conference on Sensor and Ad Hoc Communications and Networks(SECON'04)*, Oct. 2004.