A Framework Supporting Centralized Routing in Multi-Hop TSCH Networks

Yun-Shuai Yu

Abstract

Multi-hop Time-Slotted Channel Hopping (TSCH) networks seem to be the most promising solution for collecting events or changes in an industrial environment. However, its distributed routing model prevents administrators or operators from having full control of the wireless sensor network. In this paper, a framework based on existing TSCH implementation is proposed for the provision of centralized routing functionality. Experiment results prove its feasibility.

Keywords: TSCH, industrial environment, wireless sensor network, centralized routing

1. Introduction

A wireless sensor network can significantly improve the productivity and safety of industrial plants by providing the events or changes in the factory to the administrators or plant workers. Since noises generated from machines often make the wireless links unreliable for an uncertain amount of time, some standards [1, 2, 3] are made available to the general public for this issue. The OpenWSN project [1] is an open-source implementation of a fully standards-based protocol stack for wireless sensor networks, rooted in the new IEEE 802.15.4e [4] Time Slotted Channel Hopping standard. IEEE802.15.4e, coupled with Internet of Things standards, such as 6LoWPAN [5], RPL [6] and CoAP [7], it enables ultra-low-power and highly reliable mesh networks, which are fully integrated into the Internet.

The Time-Slotted feature requires sensor nodes to synchronize with a gateway. If a node can't receive the signal from the gateway, it must synchronize with another node which has already synchronized. The gateway periodically broadcasts an enhanced beacon for synchronization. The time period between any two consecutive beacons is divided into several slots, named time slot. Each node sends/receives a frame within a time slot. Thus, collisions can be significantly avoided. Channel-Hopping feature means each node periodically switches the channel using a sequence known to both sending and receiving devices where the entire frame is sent on a single channel. This feature improves the transmission reliability when a small set of channels are blocked.

OpenWSN adopts a distributed routing algorithm, named RPL, in which each node determines its next-hop relay towards the gateway. However, a node may not be able to locate a suitable parent node within a reasonable time due to its limited knowledge of the entire network. In addition, an administrator may expect the full control of the wireless network. Accordingly, this paper proposes a framework to enable central control over OpenWSN. We leverage the RPL to explore the network status. Then two UDP applications are designed to collect the network status and to distribute the commands for centralized routing. Finally, the forwarding module of OpenWSN is modified to execute the centralized routing commands. Experimental results verify its feasibility.

The remainder of this paper is organized as follows. Section 2 reviews the background and motivation. Section 3 describes the architecture of OpenWSN, which is the basis of our framework. Section 4 describes the proposed framework and introduces the related modules. Section 5 describes the experimental methodology and results. Finally, Section 6 summarizes the major contributions of the study and indicates the intended direction of future research.

2. Background and Motivation

This section introduces the distributed routing algorithm, RPL, and discusses its limitation.

A basic RPL topology is a DODAG, Destination-Oriented Directed Acyclic Graph. A DAG is a directed graph having the property that all edges are oriented in such a way that no cycles exist. A Destination-Oriented DAG means the DAG is rooted at a single destination, i.e. at a gateway, which may act as a border router between its DODAG and a backbone network.

*Corresponding Author: Yun-Shuai Yu
(E-mail: yys@ncut.edu.tw)
[1] Department of Electronic Engineering, National Chin-Yi University of Technology, No.57, Sec. 2, Zhongshan Rd., Taiping Dist., Taichung 411, Taiwan (R.O.C.)

RPL defines three types of control messages for DODAG construction and maintenance:

1). DIO: The DODAG Information Object carries information that allows a node to discover a RPL Instance, learn its configuration parameters, select a DODAG parent set, and maintain the DODAG.

2). DAO: The Destination Advertisement Object (DAO) is used to propagate destination information upward along the DODAG.

3). DIS: The DODAG Information Solicitation (DIS) message may be used to solicit a DODAG Information Object from a RPL node.

Nodes advertise their presence, affiliation with a DODAG, routing cost, and related metrics by sending link-local multicast DIO messages to all-RPL-nodes. One important field in the DIO is Rank. The Rank of a node is a scalar representation of the location of the node within a DODAG. Roughly, and a node with a smaller rank value means that it is closer to the gateway.

The details of Rank calculation is illustrated in Figure 1. The function of this example is called Objective Function Zero [8]. The formula for Rank is:

$$Rank(N) = Rank(P) + rank\_increment \quad (1)$$

N denotes a node receiving a DIO message. *Rank*(*N*) is the Rank derived from the received DIO message and a wireless link quality. *P* denotes a node sending a DIO message to node *N*. *Rank*(*P*) is the current Rank value of *P*. The *rank_increment* is defined as following:

$$rank\_increment = (2*ETX) * MinHopRankIncrease \quad (2)$$

ETX means the expected transmission number from *N* to *P*. A smaller ETX translates into a better-quality wireless link. *MinHopRankIncrease* is set as 256 by default.

In this example, we assume that the ETX of each wireless link is 4/3. At first, the Rank of the gateway is defined as 0. Therefore, the Rank field of the DIO sent by the gateway is always 0. When node *A* receives DIO sent from the gateway, its Rank is computed as:

$$Rank(A) = 0 + (2*4/3)*256 = 683.$$

If node *A* selects the gateway as its parent node, its Rank will be 683. Hereafter, the Rank field of the DIO message sent from node *A* will be 683. If node *B* received the DIO message sent from node *A*, its Rank is computed as:

$$Rank(B) = 683 + (2*4/3)*256 = 1366.$$

A DAGRank is derived from Rank form for determination of parent relationships. The DAGRank is defined as:

$$DAGRank = \text{floor}(Rank/MinHopRankIncrease) \quad (3)$$

When selecting a parent, a node with the smallest DAGRank will be selected.

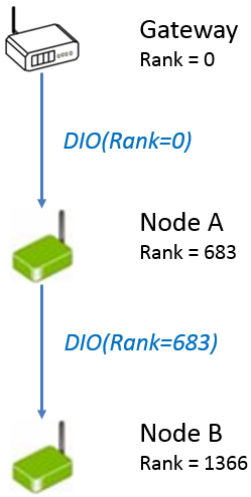

Figure 1: An example of Rank calculation

Figure 2 shows an example DODAG constructed by RPL when ETX of each wireless link is 1. If there is another obstacle lying between node *C* and node *A*, the quality of the wireless link (*C*, *A*) may become worse. Assume that the ETX of link (*C*, *A*) becomes 1.49, and node *C*'s DAGRank derived from node *A* is still 4. In this case, node *B* is apparently a better choice than node *A*, but node *C* still selects node *A* as its parent node.

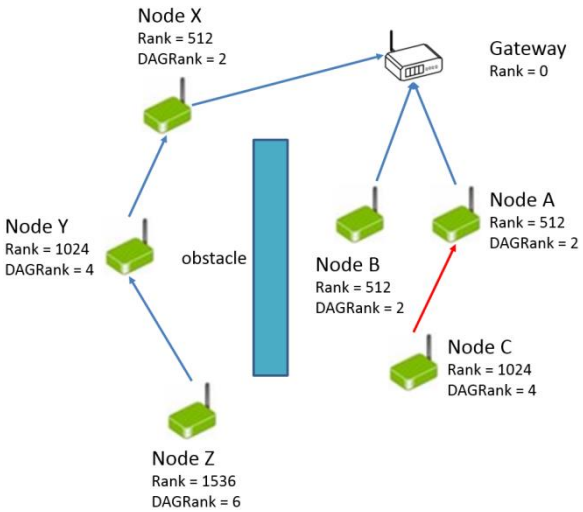

Figure 2: An example DODAG in which ETX of each wireless link is 1

If there is a noise source lying among nodes *A*, *B*, and *C*, both the ETX of link (*C*, *A*) and link (*C*, *B*) may become worse. Assume that the ETX of link (*C*, *A*) and link (*C*, *B*) is 1.99, and *C*'s DAGRank derived from node *A* or node *B* is 5. Thus, node *C* will not select node *Z*. However, selecting node *Z* can reduce almost 50% of sending time slots needed by node *C*. In other words, selecting node *Z* can notably reduce the power consumption of node *C*. This is a very important issue in an industrial environment because many sensors are/will be battery-powered.

The above examples explain some weaknesses of RPL, so it motivates us to propose a centralized routing framework for OpenWSN.

3. Architecture of OpenWSN

To the best of our knowledge, OpenWSN is the only open-source project which supports TSCH functionalities. Thus, we design and implement our centralized routing framework based on OpenWSN. In this section, we first describe the architecture of OpenWSN. Then our new modules and the modified modules will be described.

Figure 3 depicts the hardware architecture of OpenWSN. There are three components in the architecture: sensor, gateway, and PC. Sensors are the devices which detect events or conditions of their ambient. They forward the sensed data to a gateway via IEEE 802.15.4e. The gateway not only plays the role of an RPL root node, but also relays packets from/to its associated wireless sensor network to/from a PC with which it connects. The gateway exchanges packets with the PC via a USB-to-TTL serial cable. The PC may store the sensed data carried in the packets from the wireless sensor network, or just forwards the packets to the Internet. In addition, the PC can forward packets from the Internet into the wireless sensor network, or send packets to sensors by its own.

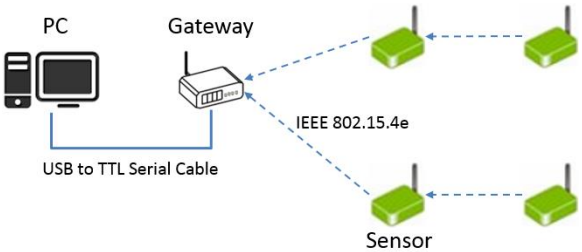

Figure 3: Hardware architecture of OpenWSN

The OpenWSN software consists of two parts: openwsn-fw and openwsn-sw. The openwsn-fw is executed in the sensors and the gateway, while the openwsn-sw is run in the PC.

Openwsn-fw is purely written in C programming language. The software architecture of openwsn-fw is illustrated in Figure 4. When the *RES* module gets a frame from the *IEEE802154E* module, it checks whether the sender is recorded in the *Neighbors* module or not. If it is not found, the MAC address of the sender will be recorded in the *Neighbors* module, currently, openwsn-fw records at most 10 neighbors. When reaching the limit, the newly discovered neighbor is discarded. When the *ICMPv6PRL* module receives a DIO message from a neighbor, it updates the sender's Rank value in the *Neighbors* module. The recorded neighbor, which can let the node have the smallest DAGRank, will be selected as the parent node of the node. The *OpenBridge* module is used by the gateway to exchange packets with the PC.

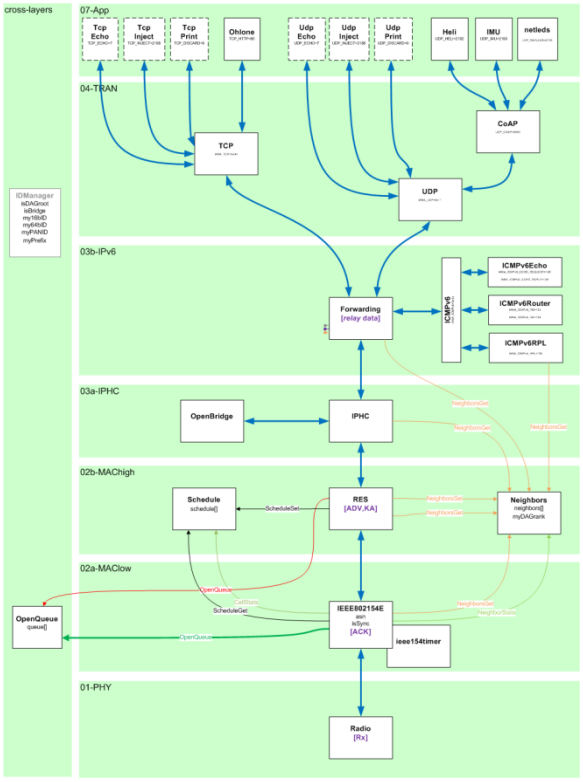

Figure 4: Software architecture of openwsn-fw

Openwsn-sw is purely written in Python programming language. The software architecture of openwsn-sw is illustrated in Figure 5. In this figure, the gray rectangles mean software modules, the white pipe is an event bus, and the arrows are events. The modules exchange events with each other through the event bus. If a module is interested in a specific event, it can register the event with the event bus. When a module triggers an event, it puts the event into the event bus. The event bus is responsible for forwarding events to the modules which have registered for the events.

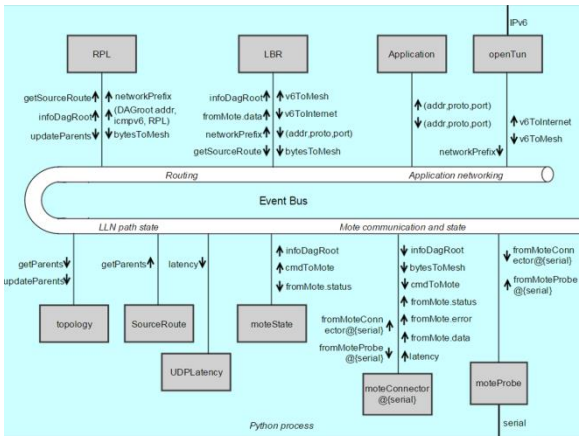

Figure 5: Software architecture of openwsn-sw

Figure 6 shows the event flow when receiving a data packet from the gateway. The *moteProbe* module receives a packet from the serial cable byte by byte. When receiving a complete packet, the *moteProbe* module forwards the packet to the *moteConnector* module. The *moteConnector* module checks the type of the packet. If the packet belongs to data packet, it will be forwarded to the *LBR* module. The *LBR* module performs 6LowPAN decompression for the data packet and then sends it to an application module. If there is no application module handling the data packet, the *LBR* module will divert the data packet to the *openTun* module. The *openTun* module just forwards the data packet to the operating system of the PC. Normally, the operating system forwards the data packet to the Internet.

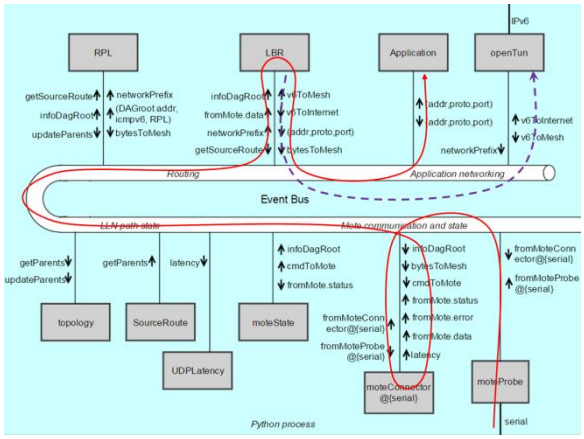

Figure 6: Event flow in openwsn-sw when receiving data packet form gateway

Figure 7 shows the event flow when receiving a data packet from the Internet. The *openTun* module forwards the data packet to the *LBR* module for 6LowPAN compression. The compressed packet traverses the *moteConnector* module, then the *moteProbe* module, to the gateway.

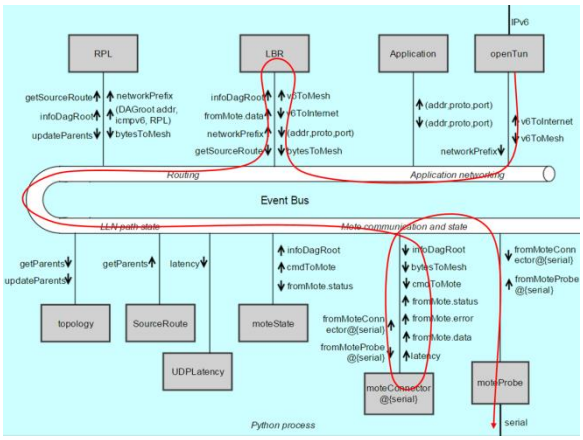

Figure 7: Event flow in openwsn-sw when receiving data packet form Internet

It is worth noting that the *LBR* module currently does not support 6LowPAN defragmentation, so the sensors must not generate any packet which is unable to fit a single IEEE 802.15.4e frame.

4. Centralized Routing Framework

This section describes the proposed framework for the provision of centralized routing in multi-hop TSCH networks. The proposed framework is realized by modifying both the openwsn-fw and openwsn-sw.

Figure 8 shows our modified openwsn-fw. We modifies the *Neighbors* module for the storage of the centralized routing rule. The *Forwarding* module is modified in order to execute the centralized routing rule. The RPL-related modules remains unchanged since we leverage them to discover the network topology. Besides, we add a new module, named *CR-fw*. *CR* stands for Centralized Routing. The *CR-fw* module is a task which communicates with the PC via UDP connections. It periodically sends a UDP packet containing the detected network status information from local port 61617 to remote port 61617. The *CR-fw* module also receives its own centralized routing rules from the PC on local port 61618. After receiving a centralized routing rule, it updates the rule in the *Neighbors* module. Note that the new and modified modules behave in the traditional way when openwsn-fw runs in the gateway.

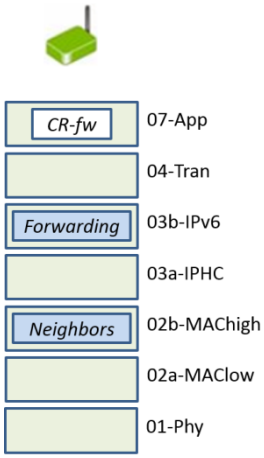

Figure 8: Modification of openwsn-fw

Figure 9 shows our modified openwsn-sw. Currently, we develop a standalone process, *CR-sw*, to interact with the original openwsn-sw because of flexibility. In the near future, we may incorporate the *CR-sw* module into openwsn-sw for performance improvement. The *CR-sw* module performs three tasks: (1) it collects the reported network status from the wireless sensor network; (2) it generates routing rules based on the collected network status information and the known data rate of each sensor; (3) it sends the new routing rule for an individual sensor. In current version, we generate the new routing rules manually for simplicity.

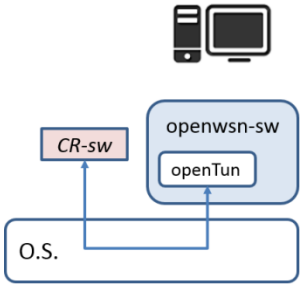

Figure 9: Modification of openwsn-sw

Figure 10 shows an example of a message flow of the proposed framework for the network status report. At first, the *CR-fw* module of sensor *A* reads the information of itself and its neighbors from the *Neighbors* module for the generation of a report message. Then the *CR-fw* module sends out a report message to its parent node, i.e. sensor *B* in this example. The report message reaches the *Forwarding* module of sensor *B*, and then is forwarded to the gateway. The gateway relays the report message to the openwsn-sw. Finally, openwsn-sw delivers the report message to our *CR-sw* module.

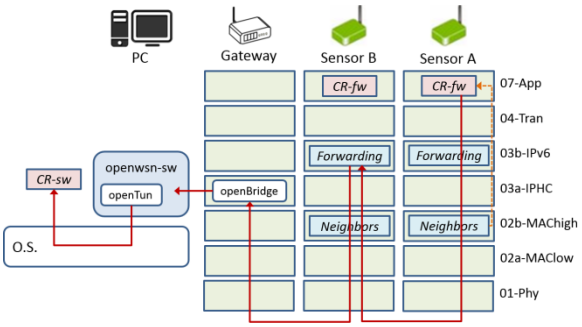

Figure 10: Message flow for network status report

After generating centralized routing rules, the *CR-sw* module sends each rule for its associated sensor one at a time. The path for responding the rule is the reverse of the path for reporting the network status. Finally, the *CR-fw* module writes the rule into the *Neighbors* module.

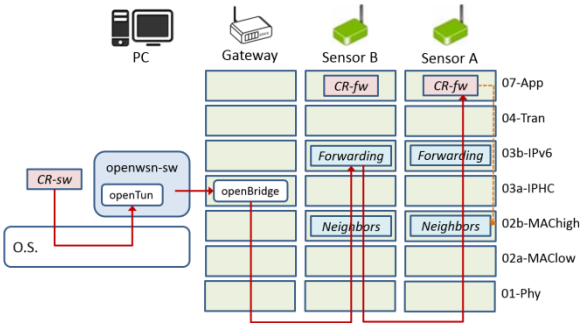

Figure 10: Message flow for centralized routing rule

Figure 11 depicts the message format of our network status report. It contains three kinds of fields: *Code*, *Itself*, and *Nbr*. The *Code* field determines the size of the *Itself* field and the number of the *Nbr* fields. The *Itself* field contains the information about the sensor which generates this message. The *Nbr* filed has the information of a unique neighbor node. Recall that the *LBR* module of openwsn-sw does not support 6LowPAN decompression. Therefore, we limit the maximum number of the *Nbr* fields to 3 so to prevent the 6LowPAN fragmentation. Although the limitation may forbid our *CR-sw* module from having a complete knowledge of the wireless network, we think the collected information is sufficient to make good decisions. In addition, the information inside the parentheses is the size of the field, and the size unit is Byte.

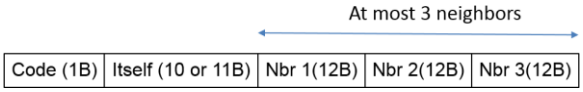

Figure 11: Message format of network status report

Figure 12 shows the format of the *Code* field. It contains three sub-fields: *res*, *p*, and *Num of Nbr*. In this figure, the first row is the name of each sub-field, the second row is the size, and the third row is the possible values. The first sub-field is reserved for future use. The *p* sub-field determines the size of the *Itself* field. If the value of the *p* sub-field is 0, the size of the *Itself* field is 10 Bytes. If the value of the *p* sub-field is 1, the size of the *Itself* field is 11 Bytes. The *Num of Nbr* sub-field indicates the number of neighbor information in this message. The value of this sub-field ranges from 0 to 15, although current possible value ranges from 1 to 3. This design reserves the flexibility for the provision of more neighbor information in the case that OpenWSN supports 6LowPAN fragmentation and defragmentation someday.

res	p	Num of Nbr
3b	1b	4b
xxx	0, 1	0~15

Figure 12: Format of *Code* field

If the above *p* sub-field is 0, the format of the *Itself* field is shown in Figure 13(a). The *MAC* sub-field has the 64-bits MAC address of the sensor which generates this message. The value of the *Rank* sub-field is the Rank value generated by its own sensor's *ICMPv6PRL* module. If the above *p* sub-field is 1, the format of the *Itself* field is shown in Figure 13(b). The definitions of the first two are the same with the ones shown in Figure 13(a). The *Power* sub-field is the residual energy of the sensor, whose size unit is %. Since the data rates of sensors tend to be relatively low, e.g. no more than one packet per second, the residual energy changes slowly. Thus, it is unnecessary for a sensor to respond its residual energy every time it reports.

MAC	Rank
8B	2B

(a)

MAC	Rank	Power
8B	2B	1B

(b)

Figure 13: Format of *Itself* field

Figure 14 shows the format of the *Nbr* field. It contains four sub-fields: *MAC*, *Rank*, *numTx*, and *numTxAck*. The *MAC* sub-field has the 64-bits MAC address of the neighbor node. The value of the *Rank* sub-field is the Rank value carried in the DIO message sent from the neighbor. The *numTx* sub-field is the number of data frames transmitted form its own sensor to the neighbor. The *numTxAck* sub-field is the number of ack frames received from the neighbor. The ratio of the value of the *numTx* sub-field to the one of the *numTxAck* sub-field is considered as the ETX. When the value of the *numTx* sub-field equals 255, both of the *numTx* and *numTxAck* subfields are divided by 2. If the value of the *numTxAck* subfield is 0, the ETX for this neighbor is set as 15 by default.

MAC	Rank	numTx	numTxAck
8B	2B	1B	1B

Figure 14: Format of *Nbr* field

Figure 15 shows the format of the command for the centralized routing rule. The first sub-field indicates the MAC address of a sensor should receive this command. The second sub-field denotes the MAC address of a new parent node for the sensor specified by the first sub-field.

MAC of a node to command	MAC of its new parent
8B	8B

Figure 15: Format of command for centralized routing rule

The modification of the *Neighbors* module is as follows: (1) a data structure is added for the storage of the MAC address of a new parent assigned by the command for a routing rule. (2) a global variable, *isCentralized*, is added. After receiving a command for routing rule, this variable is set as *True*. If the value of the ETX for the new parent specified in the routing rule is below a threshold, the variable continues to be *True*; otherwise, it will be *False*. The key idea of the design is that a sensor can know the quality of the wireless link between itself and its parent for the centralized routing much earlier than the *CR-sw* can. Therefore, if the reliability of the parent for the centralized routing is not accepted, sensors will switch back to the distributed routing.

Figure 16 shows the routing algorithm implemented in our modified *Forwarding* module. The *Hybrid* word means our algorithm can switch between the centralized routing and the distributed routing. A variable, *mode*, is used to indicate whether an input packet should be routed in a centralized fashion or not. By default, a packet is routed in a traditional distributed fashion unless all the conditions specified in lines 2, 3, 4, and 5 are satisfied. The condition of line 2 means that we currently support only UDP packet for simplicity. Note that there are three types of traffics in a wireless sensor network: up-stream, down-stream, and peer-to-peer. An up-stream traffic is from a sensor to a gateway, a down-stream traffic is the opposite, and a peer-to-peer traffic is between any pair of sensors. The condition of line 3 means only the up-stream traffic can be routed in a centralized fashion since we only assign a parent node for this purpose. We plan to enable the centralized routing for other traffic types in the near future. The condition of line 4 means that the sensor should have been assigned a parent for the centralized routing. The condition of line 5 requires that the reliability of the parent for centralized routing is acceptable. If it is not reliable enough, the variable, *isCentralized*, is reset. Currently, the value of the threshold *T* is set as 4 since there are at most 4 transmissions for a data frame in an IEEE 802.15.4e

network. Finally, we select the next-hop relay based on the value of the variable, *mode*, in line 9.

Algorithm Hybrid_Forwarding(packet):

1. mode = distributed
2. If packet ∈ UDP:
3. If packet.dst = gateway or packet.dst ∈ Internet:
4. If *isCentralized* = True:
5. If parent.ETX < Threshold *T*:
6. mode = centralized
7. else:
8. *isCentralized* ← False:
9. *Forwarding*(*mode*, *packet*)

End Algorithm

Figure 16: Algorithm of our hybrid routing algorithm

5. Experiments

In order to validate our proposed framework, we implemented it on OpenMote development kit [9], as shown in Figure 17. The OpenMote-CC2538 is equipped with a wireless microcontroller System-on-Chip (SoC), CC2538 [10], which incorporates a robust IEEE 802.15.4 radio. The OpenBattery is an extension board composed of a battery placeholder. The OpenBase is an interface board for the OpenMote-CC2538. It enables the OpenMote-CC2538 to communicate with a computer through a serial port. The OpenBase can also be powered via the serial port. The OpenMote-CC2538 can be mounted on an OpenBattery as a sensor or on an OpenBase as a gateway.

OpenMote-CC2538 OpenBattery OpenBase

Figure 17: Hardware for experiment

We further adopt CC2531 USB Evaluation Module Kit [11], as shown in Figure 19, for an IEEE 802.15.4 packet sniffer. It is a USB Dongle which can be mounted on a computer. SmartRF Protocol Packet Sniffer [12] is the corresponding sniffer software.

Figure 19: CC2531 USB Evaluation Module Kit

The overall experimental platform is shown in Figure 18. The italic words indicate the last two bytes of the 64-bit MAC addresses of their device.

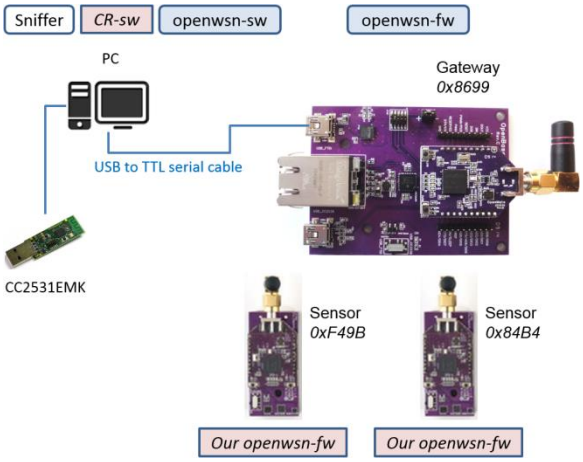

Figure 18: Experimental platform

In the first experiment, sensor 0x84B4 was turned on while sensor 0xF49B was turned off. When observing that a network status report message was sent from sensor 0x84B4, the *CR-sw* module was asked to select sensor 0xF49B as a new parent for the centralized routing. Via the sniffer's graphic user interface (GUI), we observed that sensor 0x84B4 sent data frames to sensor 0xF49B hereafter. Apparently, there was no ack frame from sensor 0xF49B since it was currently shutdown. Then after a while, sensor 0x84B4 was switched back to the distributed routing fashion, i.e. it sent data frame to the gateway directly.

In the second experiment, sensor 0x84B4 was powered on first. When making sure that sensor 0x84B4 selected the gateway, we turned on sensor 0xF49B. After sensor 0xF49B joined the TSCH network, we commanded sensor 0x84B4 to select sensor 0xF49B. As we expected, the sniffer's GUI showed that sensor 0x84B4 sent data frames to sensor 0xF49B and latter responded ack frames to the former. The above two experiments proved that the proposed framework is feasible.

6. Conclusions

In this paper, a framework is proposed for the provision of the centralized routing functionality over multi-hop Time-Slotted Channel-Hopping networks. We leverage the existing distributed routing mechanism for network status discovery. A network status collection mechanism is developed for providing the collected information to administrators to make centralized routing rules. A command distribution mechanism is also designed to notify sensors about new routing rules. We further propose a hybrid routing algorithm for sensors to switch between the proposed centralized routing and the existing distributed routing based on the network conditions. Experiments on OpenMote development kit prove that the proposed framework is feasible.

Future studies will address the problem of generating optimal centralized routing rules without the involvement of humans. In addition, large-scale simulations will be conducted for performance analysis.

References

[1]. T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister., "OpenWSN: A Standards-Based Low-Power Wireless Development Environment," Wiley's Transactions on Emerging Telecommunications Technologies, vol. 23, no. 5, pp. 480-493, Aug. 2012.

[2]. IEC 62591 Ed. 1.0 b:2010, "Industrial Communication Networks -- Wireless Communication Network and Communication Profiles -- WirelessHART™," 2010.

[3]. ISA, "Wireless systems for industrial automation: Process control and related applications", ISA 100.11a, May 2008.

[4]. IEEE, "IEEE Standard for Local and metropolitan area networks -- Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer", IEEE Std. 802.15.4e-2012, Apr. 2012.

[5]. J. Hui, Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, Sep. 2011.

[6]. T. Winter, Ed., P. Thubert, Ed., A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, Mar. 2012,

[7]. Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, Jun. 2014.

[8]. P. Thubert, Ed., "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6552, Mar. 2012

[9]. OpenMote. < http://www.openmote.com/>.

[10]. CC2538. < http://www.ti.com/product/CC2538>.

[11]. CC2531EMK. < http://www.ti.com/tool/cc2531emk>.

[12]. SmartRF Protocol Packet Sniffer. < http://www.ti.com/tool/packet-sniffer>

Yun-Shuai Yu obtained his B.S., M.S., and Ph.D. degrees in electrical engineering from National Cheng Kung University, Taiwan, in 2002, 2004, and 2011 respectively. He is an Assistant Professor in the Department of Electronic Engineering at National Chin-Yi University of Technology. His research interests include Internet of Thing, wireless sensor network, embedded system, and media streaming.