TEZP Protocol Reference

Versions 1.0

Document version 1.0.01d

Date:

Document History

revision	date	description
01	January 2567	ฉบับตั้งต้น

คำอธิบายทั่วไป

Document revision

หมายเลข revision ของเอกสารนี้จะมีรูปแบบเป็น V.v.r โดยแต่ละตัวจะมีความหมายดังนี้

V เป็นเลข major version ของโปรโตคอล

v เป็นเลข minor version ของโปรโตคอล

r เป็นเลขฉบับแก้ไข (revision) ของเอกสารอธิบายโปรโตคอล ถ้าหมายเลขนี้ลงท้ายด้วย d จะหมายถึงเป็นฉบับร่าง (draft)

ตัวอย่างเช่น 1.0.01 จะหมายถึงเอกสารอธิบายโปรโตคอล version 1.0 ฉบับแก้ไขที่ 01

รูปแบบที่ใช้ในเอกสารนี้

`Courier font`	อักขระ ASCII ในข้อมูล
␣	อักขระช่องว่าง
…	มีข้อมูลต่อท้ายจำนวน 0 ตัวอักษร หรือมากกว่า
0x	ข้อมูลในรูปแบบฐานสิบหก (hexadecimal)
ABC	แสดงถึงข้อมูลส่วนเพิ่ม (optional)

สารบัญ

1. บทนำ

TEZP เป็นโปรโตคอลสำหรับงาน IoT ที่ถูกออกแบบมาเพื่อรองรับข้อมูลจากอุปกรณ์ตรวจจับต่างๆ ซึ่งโดยปรกติจะมีปริมาณข้อมูลไม่มาก และสามารถยอมรับความล่าช้าในการส่งข้อมูลถึงปลายทางได้ในระดับหลายสิบถึงหลายร้อยมิลลิวินาที และไม่จำเป็นที่การส่งข้อมูลจะไปถึงผู้รับได้ถูกต้องสมบูรณ์ทั้งหมด แต่จะขึ้นกับความต้องการของการใช้งานในระดับบน ซึ่งถ้าลักษณะการใช้งานมีความต้องการข้อมูลที่ถูกต้องสมบูรณ์ จะเป็นหน้าที่ของโปรแกรมใช้งานระดับบนที่จะรองรับการจัดการตรงนี้

แนวคิดในการออกแบบ

การออกแบบ TEZP วางอยู่บนพื้นฐานเบื้องต้นดังนี้

a. การส่งและรับข้อมูลของ TEZP ทำงานอยู่บนเครือข่ายไร้สาย LoRa
b. ข้อมูลที่รับส่งในแต่ละครั้งมีจำนวนไม่เกินกว่า 32 ไบท์
c. การส่งข้อมูลเป็นไปในลักษณะ best practice คือไม่มีการรับรองว่าผู้รับจะได้รับข้อมูลทุกข้อมูลที่ส่งออกไป
d. ข้อมูลอาจจะมีความล่าช้า (latency) ได้ คือระยะเวลาที่ใช้นับตั้งแต่ผู้ส่งได้ทำการส่งข้อมูลออกไปจนกระทั่งผู้รับได้รับข้อมูล อาจจะใช้เวลาได้มากถึง 10 วินาที ระยะเวลาที่มากกว่านี้ให้ถือเป็น timeout ของการส่งและรับข้อมูลนั้นๆ
e. การรับส่งอาจจะเป็นการสื่อสารระหว่างผู้ส่งและผู้รับโดยตรง หรือเป็นการสื่อสารผ่านผู้ส่งต่อ (relay) และสามารถมีผู้ส่งต่อได้มากกว่า 1 ราย แต่ไม่เกิน 8 ราย

LoRa โดยสังเขป

LoRa ใช้เทคนิคการโมดูเลทสัญญาณชนิด Chirp Spread Spectrum (CSS) โดยข้อมูลที่จะส่งออกอากาศจะถูกโมดูเลทกับคลื่นความถี่วิทยุที่มีการเปลี่ยนความถี่จากค่าหนึ่งไปยังอีกค่าหนึ่งอย่างต่อเนื่องในช่วงเวลาสั้นๆ ลักษณะคลื่นความถี่ลักษณะนี้จะเรียกว่า chirp pulse และช่วงของความถี่ที่เปลี่ยนแปลงเรียกว่าช่วงกว้างของความถี่ หรือ bandwidth การโมดูเลทข้อมูลกับสัญญาณ chirp pulse ทำให้สัญญาณวิทยุผลลัพธ์ที่ได้ออกมามีความทนทานต่อสัญญาณรบกวนสูง จึงทำให้สามารถรับส่งได้ในระยะทางที่ไกลโดยใช้ระดับกำลังส่งที่ต่ำได้

ความเร็วของข้อมูลที่สามารถส่งด้วย LoRa จะขึ้นกับสองปัจจัยคือช่วงกว้างของช่องความถี่ที่สามารถใช้ได้ (bandwidth) และอัตราความเร็วของการเปลี่ยนความถี่ ในข้อกำหนดมาตรฐานของ LoRa สำหรับกลุ่มประเทศอเมริกา กำหนดให้สามารถใช้ช่วงกว้างของความถี่ได้ 3 ค่าคือ 125KHz, 250KHz, และ 500KHz ส่วนกลุ่มประเทศยุโรปและเอเชียจะสามารถใช้ได้เฉพาะ 125KHz และ 250KHz เท่านั้น

สำหรับอัตราความเร็วในการเปลี่ยนความถี่จะเรียกว่า spreading factor (SF) โดยระบุเป็นตัวเลขตั้งแต่ 6 ถึง 12 หมายเลขที่ต่ำกว่าหมายถึงอัตราความเร็วในการการเปลี่ยนแปลงความถี่ที่มากกว่า ค่า SF แต่ละลำดับที่น้อยลงจะหมายถึงการเปลี่ยนแปลงความถี่ที่เร็วขึ้นเป็นสองเท่า ซึ่งจะมีผลทำให้ความเร็วข้อมูลเพิ่มเป็นสองเท่าด้วย แต่ก็มีผลทำให้ความทนทานต่อสัญญาณรบกวนลดน้อยลงด้วยเช่นกัน

ตารางข้างล่างนี้เป็นอัตราความเร็วข้อมูลสำหรับค่า SF แต่ละค่า สำหรับการส่งโดยใช้ช่วงกว้างของความถี่ 125KHz

spreading factor	equivalent bit rate (kb/s)
12	0.293
11	0.537
10	0.976
9	1757
8	3125
7	5468
6	9375

ข้อดีอีกประการหนึ่งของโมดูเลชั่นนี้คือการส่งที่ใช้ค่า SF แตกต่างกันในช่องความถี่เดียวกันจะ แต่จะถูกมองเป็นเหมือนสัญญาณรบกวนทั่วไป จึงทำให้ในแต่ละช่วงความถี่เดียวกันสามารถมีจำนวนเครือข่ายหลายๆ เครือข่ายใช้งานร่วมกันได้ เพียงกำหนดให้แต่ละเครือข่ายใช้ค่า SF ที่แตกต่างกันออกไป

2. โครงสร้างการทำงาน

TEZP จะแบ่งโครงสร้างการทำงานออกเป็น 2 ส่วนหลัก คือ Protocol Driver และ Protocol Manager Driver

Protocol Driver

Protocol Driver ทำหน้าที่ติดต่อกับอุปกรณ์ฮาร์ดแวร์ที่ทำการส่งข้อมูลออกทางคลื่นความถี่วิทยุ และแปลงข้อมูลที่ได้รับทางความถี่วิทยุกลับมาเป็นข้อมูลเพื่อดำเนินการต่อไป วิธีการรับส่งข้อมูลในชั้นของ Driver จะใช้รูปแบบ best effort คือข้อมูลที่ถูกส่งออกไปจะไม่มีการยืนยันจากปลายทางว่าได้รับแล้ว หรือไม่สามารถรับได้ จะเป็นหน้าที่ของ Protocol Manager ที่จะจัดการต่อไป การรับส่งข้อมูลในส่วนนี้จะขึ้นอยู่กับลักษณะและชนิดของอุปกรณ์ที่ใช้ ซึ่งในเบื้องต้น TEZP ได้รับการออกแบบเพื่อใช้งานกับอุปกรณ์ LoRa ตามมาตรฐานที่ได้รับอนุญาตให้ใช้งานในประเทศไทย ที่ย่านความถี่ 923MHz และกำลังส่งไม่เกินข้อกำหนด

Protocol Manager

ดังที่ได้กล่าวไปแล้วว่า Protocol Driver ใช้รูปแบบการรับส่งชนิด best effort จึงเป็นหน้าที่ของ Protocol Manager ที่จะทำหน้าที่ร่วมกับ Protocol Driver เพื่อควบคุมการรับส่งระหว่างอุปกรณ์ตั้งแต่สองตัวขึ้นไปให้ได้คุณภาพการรับส่ง (QOS-quality of server) ในระดับที่ต้องการ โดยจะขึ้นอยู่ว่าผู้ใช้จะใช้โครงสร้างการติดต่อชนิดใด รูปแบบโครงสร้งที่เป็นไปได้อาจจะเป็นชนิด point to point หรือ single master multi slave หรือ mesh การทำงานของ Protocol Manager จะเรียกใช้บริการ API ของ Protocol Driver ที่มีไว้ให้เพื่อส่งข้อมูลออกอากาศ และรับข้อมูลที่เข้ามาทางสายอากาศ ในการออกแบบนี้จะแยกหน้าที่ของ Protocol Manager ออกเป็น 4 ส่วนย่อย ให้สามารถเรียกใช้เฉพาะกลุ่มการทำงานที่ต้องการตามรูปแบบเครือข่ายที่ต้องการใช้งาน ดังนี้

Datagram Service การรับส่งข้อมูลจะสามารถกำหนดแอดเดรสต้นทางปลายทาง โดยข้อมูลเป็นชนิดความยาวแปรเปลี่ยนได้ และรองรับการรับส่งข้อมูลชนิดไม่เจาะจงผู้รับ (broadcast) แต่ไม่มีการยืนยันความถูกต้องของการรับข้อมูลจากปลายทาง

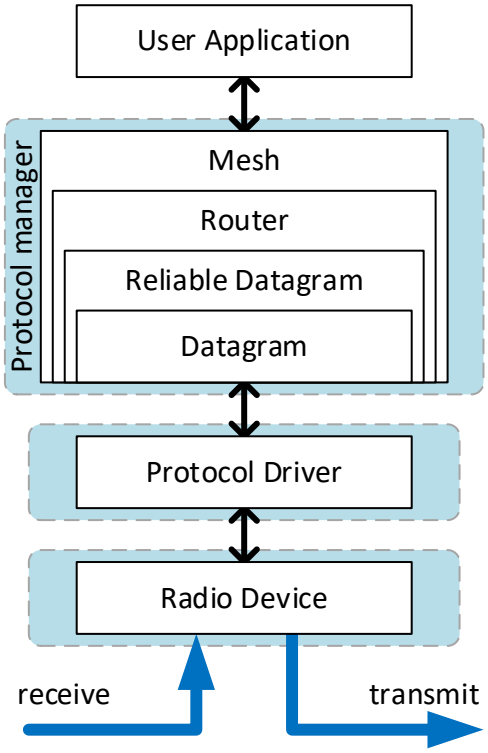
Reliable Datagram Service มีความสามารถทำงานทุกอย่างที่มีอยู่ใน Datagram Service และเพิ่มความสามารถในการยืนยันความถูกต้องของข้อมูลที่ปลายทางได้รับกลับมายังต้นทาง และความสามารถในการส่งซ้ำข้อมูลที่ผิดพลาดได้

Router Service มีความสามารถทำงานทุกอย่างที่มีอยู่ใน Reliable Datagram Service และเพิ่มเติมความสามารถรองรับการส่งต่อข้อมูลผ่านอุปกรณ์ตัวกลาง (intermediate node) ไปยังอุปกรณ์ปลายทางที่รัศมีการครอบคลุมของสัญญาณจากอุปกรณ์ต้นทางไม่สามารถส่งสัญญาณถึงได้โดยตรง อุปกรณ์ตัวกลางนี้สามารถมีได้หลายตัวตามความต้องการ โดยแต่ละตัวจะต้องได้รับการกำหนดเส้นทางของการรับส่งข้อมูลไว้ล่วงหน้า

โครงสร้างการทำงาน

Mesh Service มีความสามารถทำงานทุกอย่างที่มีอยู่ใน Router Service และเพิ่มเติมความสามารถในการเรียนรู้เส้นทางส่งต่อข้อมูลผ่านตัวกลางได้โดยอัตโนมัติ ไม่จำเป็นต้องกำหนดเส้นทางล่วงหน้าไว้ก่อน และสามารถปรับเปลี่ยนเส้นทางได้เอง ในกรณีที่เส้นทางที่เรียนรู้ไว้ก่อนหน้านี้ ไม่สามารถใช้งานได้

ลักษณะการทำงานของ TEZP จะทำงานเป็นลำดับชั้นตามรูปข้างล่างนี้



Protocol Driver API

Protocol Driver จะมีฟังก์ชั่นเพื่อรองรับการเรียกใช้งานโดย Manager ดังนี้

bool init ()

เตรียมอุปกรณ์โมดูลรับส่งวิทยุให้พร้อมทำงาน รวมทั้งตั้งค่าตัวแปรที่จำเป็น และทรัพยากรต่างๆ ที่ต้องใช้เพื่อการติดต่อรับส่งให้พร้อมสำหรับการทำงาน

bool available ()

ตอบกลับสถานะของ Driver ว่ามีข้อมูลที่ได้รับเข้ามารออยู่หรือไม่ ในการทำงานปรกติ Manager อ่านสถานะนี้และถ้ามีข้อมูลรออยู่ Manager จะเรียกฟังก์ชั่นเพื่ออ่านข้อมูลนำไปใช้งาน ถ้าอุปกรณ์ไม่ได้อยู่ในสถานะพร้อมรับข้อมูล จะจัดการให้อุปกรณ์อยู่ในสถานะพร้อมรับข้อมูล

bool recv (uint8_t *buf, uint8_t *len)

โครงสร้างการทำงาน

ส่งข้อมูลที่รออยู่ในตัวอุปกรณ์รับส่ง และระบุความยาวของข้อมูล ถ้าอุปกรณ์ไม่ได้อยู่ในสถานะพร้อมรับข้อมูล จะจัดการให้อุปกรณ์พร้อมรอรับข้อมูล

bool send (const uint8_t *data, uint8_t len)

รอให้สถานะของอุปกรณ์พร้อมส่งข้อมูล และตรวจสอบช่องความถี่ว่าไม่มีผู้ใช้งาน (ถ้าอุปกรณ์รองรับการตรวจสอบนี้) และเมื่ออุปกรณ์พร้อม จะทำการส่งข้อมูลออกอากาศ ข้อมูลที่จะส่งออกจะต้องมีความยาวไม่เกินที่กำหนด ข้อมูลที่ยาวเกินกำหนดจะไม่ถูกส่งออก และจะตอบกลับสถานะนี้ให้ผู้เรียกใช้งานได้รับทราบ

uint8_t maxMessageLength ()

ตอบกลับค่าจำนวนข้อมูลสูงสุดที่อุปกรณ์จะสามารถรับส่งได้

void waitAvailable (uint16_t polldelay)

รอรับข้อมูลจนกว่าจะมีข้อมูลเข้ามา

bool waitAvailableTimeout (uint16_t timeout, uint16_t polldelay)

รอรับข้อมูลจนกว่าจะมีข้อมูลเข้ามา หรือจนกว่ากำหนดระยะเวลาในการรับข้อมูลที่ระบุไว้ได้หมดลง

bool waitPacketSent ()

รอจนกว่าการส่งข้อมูลครั้งล่าสุดสำเร็จ

bool waitPacketSent (uint16_t timeout)

รอจนกว่าการส่งข้อมูลครั้งล่าสุดสามารถส่งได้สำเร็จ หรือจนกว่ากำหนดระยะเวลาในการส่งข้อมูลที่ระบุไว้ได้หมดลง

bool waitCAD ()

รอจนกว่าช่องความถี่จะว่าง หรือจนกว่ากำหนดระยะเวลาในการรอที่ได้ตั้งไว้ได้หมดลง ในกรณีที่อุปกรณ์ไม่รองรับการตรวจสอบช่องความถี่ ให้ตอบกลับ true ทันที

void setCADTimeout (unsigned long cad_timeout)

ตั้งค่ากำหนดระยะเวลาในการรอการตรวจสอบช่องความถี่ ในกรณีที่อุปกรณ์มีความสามารถตรวจสอบการใช้ช่องความถี่ว่าว่างหรือไม่ว่างได้

bool isChannelActive ()

แจ้งกลับสถานะช่องความถี่ว่าง ถ้าอุปกรณ์ไม่รองรับการตรวจสอบช่องความถี่ ให้แจ้งสถานะ true กลับในทันที

โครงสร้างการทำงาน

void setThisAddress (uint8_t thisAddress)

ตั้งค่าแอดเดรสเพื่อใช้รับข้อมูล ในเครือข่ายใดๆ อันหนึ่ง ค่าแอดเดรสสำหรับอุปกรณ์แต่ละตัวในเครือข่ายจะต้องไม่ซ้ำกัน

void setHeaderTo (uint8_t to)

ตั้งค่าแอดเดรส TO ใน Header เพื่อใช้ส่งข้อมูล

void setHeaderFrom (uint8_t from)

ตั้งค่าแอดเดรส FROM ใน Header เพื่อใช้ส่งข้อมูล

void setHeaderId (uint8_t id)

ตั้งค่า ID ใน HEADER เพื่อใช้ในการส่งข้อมูล

void setHeaderFlags (uint8_t set, uint8_t clear=RH_FLAGS_APPLICATION_SPECIFIC)

ตั้งค่า FLAG ใน HEADER เพื่อใช้ในการส่งข้อมูล

void setPromiscuous (bool promiscuous)

ตั้งโหมดการทำงานของอุปกรณ์ให้อยู่ในโหมดรอรับข้อมูล โดยไม่มีการเจาะจงแอดเดรสใดๆ

uint8_t headerTo ()

ตอบกลับค่าแอดเดรส TO ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t headerFrom ()

ตอบกลับค่าแอดเดรส FROM ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t headerId ()

ตอบกลับค่า ID ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t headerFlags ()

ตอบกลับค่า FLAGS ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

int16_t lastRssi ()

ตอบกลับค่าความแรงสัญญาณ (RSSI) ที่อ่านได้จากการรับสัญญาณครั้งล่าสุด

RHMode mode ()

ตอบกลับค่าแสดงโหมดการทำงานปัจจุบันของ Driver

void setMode (RHMode mode)

ตั้งโหมดการทำงานของอุปกรณ์

bool sleep ()

กำหนดให้อุปกรณ์อยู่ในสถานะใช้พลังงานต่ำ

uint16_t rxBad ()

ตอบกลับค่าจำนวนครั้งของการรับข้อมูลที่ไม่สำเร็จด้วยสาเหตุต่างๆ

uint16_t rxGood ()

ตอบกลับค่าจำนวนครั้งของการรับข้อมูลที่สำเร็จ

uint16_t txGood ()

ตอบกลับค่าจำนวนครั้งของการส่งข้อมูลที่สำเร็จ

Protocol Manager API

Manager แต่ละกลุ่มย่อย มีฟังก์ชั่นเพื่อรองรับการเรียกใช้งานจาก User Application ดังนี้

Datagram API

bool init ()

ตั้งค่าต่างๆ เตรียมพร้อมสำหรับการให้บริการเป็น Datagram Service พร้อมกับสั่งการให้ Drive พร้อมรอรับคำสั่ง

void setThisAddress (uint8_t thisAddress)

ตั้งค่าแอดเดรสของอุปกรณ์

bool sendto (uint8_t *buf, uint8_t len, uint8_t address)

ส่งข้อมูลที่กำหนดไว้ไปยังอุปกรณ์ปลายทางตามแอดเดรสที่ระบุไว้

bool recvfrom (uint8_t *buf, uint8_t *len, uint8_t *from, uint8_t *to, uint8_t *id, uint8_t *flags)

รับข้อมูล(ถ้ามี) พร้อมทั้งค่าแอดเดรสต้นทาง (from), แอดเดรสปลายทาง (to), id, และ flags ที่ได้จากข้อมูลที่ได้รับเข้ามาพร้อมกับข้อมูล

bool available ()

ตรวจสอบว่ามีข้อมูลที่ได้รับเข้ามารออยู่หรือไม่

โครงสร้างการทำงาน

void waitAvailable (uint16_t polldelay)

รอรับข้อมูลจนกว่าจะมีข้อมูลเข้ามา

bool waitPacketSent ()

รอจนกว่าการส่งข้อมูลครั้งล่าสุดสำเร็จเรียบร้อย

bool waitPacketSent (uint16_t timeout)

รอจนกว่าการส่งข้อมูลครั้งล่าสุดสำเร็จเรียบร้อย หรือจนกว่าเวลาที่ระบุไว้หมดลง

bool waitAvailableTimeout (uint16_t timeout, uint16_t polldelay=0)

รอรับข้อมูลจนกว่าจะมีข้อมูลเข้ามา หรือจนกว่าเวลาที่ระบุไว้หมดลง

void setHeaderTo (uint8_t to)

ตั้งค่าแอดเดรส TO ใน Header เพื่อใช้ส่งข้อมูล

void setHeaderFrom (uint8_t from)

ตั้งค่าแอดเดรส FROM ใน Header เพื่อใช้ส่งข้อมูล

void setHeaderId (uint8_t id)

ตั้งค่า ID ใน Header เพื่อใช้ส่งข้อมูล

void setHeaderFlags (uint8_t set, uint8_t clear=RH_FLAGS_NONE)

ตั้งค่า FLAGใน Header เพื่อใช้ส่งข้อมูล

uint8_t headerTo ()

ตอบกลับค่าแอดเดรส TO ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t headerFrom ()

ตอบกลับค่าแอดเดรส FROM ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t headerId ()

ตอบกลับค่า ID ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t headerFlags ()

ตอบกลับค่า FLAG ใช้งานล่าสุดที่ได้กำหนดไว้ใน HEADER

uint8_t thisAddress ()

ตอบกลับแอดเดรสของอุปกรณ์นี้

Reliable Datagram API

void setTimeout (uint16_t timeout)

ตั้งค่ากำหนดระยะเวลาที่จะรอการส่งข้อมูลให้สำเร็จ ซึ่งถ้าการส่งข้อมูลไม่สำเร็จภายในระยะเวลานี้ จะทำการส่งข้อมูลเดิมซ้ำ ค่าระยะเวลานี้จะต้องไม่น้อยกว่าเวลาที่ฝั่งผู้รับใช้ในการรับข้อมูลทั้งหมด

void setRetries (uint8_t retries)

ตั้งค่าจำนวนครั้งในการส่งข้อมูลซ้ำก่อนที่จะแสดงสถานะการส่งข้อมูลไม่สำเร็จ

uint8_t retries ()

ส่งกลับค่าจำนวนการส่งซ้ำที่ได้กำหนดไว้ก่อนหน้านี้

bool sendtoWait (uint8_t *buf, uint8_t len, uint8_t address)

ส่งข้อมูลที่กำหนดไว้ออกไป และรอจนกว่าจะมีการตอบรับกลับมาจากผู้รับปลายทาง และตอบกลับสถานะได้รับการตอบรับ (true) ในกรณีที่ได้รับการตอบรับกลับมา หรือตอบกลับสถานะไม่สำเร็จ ในกรณีที่ไม่ได้รับการตอบรับกลับมา กลับไปยังโปรแกรมที่เรียกใช้งาน ในกรณีที่กำหนดแอดเดรส TO เป็นแอดเดรสที่ไม่เจาะจงผู้รับ (broadcast) ซึ่งผู้รับปลายทางจะไม่ตอบรับกลับมา จะตอบกลับสถานะสำเร็จกลับไปยังโปรแกรมที่เรียกใช้งานในทันที

bool recvfromAck (uint8_t *buf, uint8_t *len, uint8_t *from, uint8_t *to, uint8_t *id, uint8_t *flags)

ถ้าอุปกรณ์ได้รับข้อมูลที่กำหนดแอดเดรสผู้รับ (TO) ตรงกับตัวอุปกรณ์ จะทำการส่งข้อมูลตอบรับกลับไปยังแอดเดรสของผู้ส่งต้นทาง (FROM) และส่งข้อมูลดังกล่าวพร้อมทั้งความยาวข้อมูลและข้อมูล ID และ FLAG กลับไปยังโปรแกรมที่เรียกใช้งาน ในกรณีที่แอดเดรสผู้รับ(TO) ระบุเป็นแอดเดรสไม่เจาะจงผู้รับ (broadcast) จะไม่มีการส่งข้อมูลยืนยันการรับกลับแต่อย่างใด

bool recvfromAckTimeout (uint8_t *buf, uint8_t *len, uint16_t timeout, uint8_t *from, uint8_t *to, uint8_t *id, uint8_t *flags)

ทำงานเช่นเดียวกับ recvfromAck แต่สามารถระบุระยะเวลาที่จะรอรับข้อมูล และจะตอบกลับไปยังโปรแกรมเรียกใช้งานถ้าระยะเวลาดังกล่าวหมดลง

uint32_t retransmissions ()

ตอบกับค่าจำนวนการส่งซ้ำที่เกิดขึ้นตั้งแต่มีการการเรียกดูค่านี้ครั้งล่าสุด หรือตั้งแต่เปิดการทำงานตัวอุปกรณ์

void resetRetransmissions ()

ลบค่าจำนวนการส่งซ้ำให้เป็น 0

Router API

bool init ()

ตั้งค่าเริ่มต้นการทำงาน และสั่งอุปกรณ์ให้เริ่มทำงาน

void setIsaRouter (bool isa_router)

ตั้งค่าระบุการทำงานของตัวอุปกรณ์ว่าให้หรือไม่ให้ทำหน้าที่เป็น router

void setMaxHops (uint8_t max_hops)

ตั้งค่าจำนวนการส่งต่อสูงสุด ข้อมูลจะยังถูกส่งต่อออกไปถ้าจำนวนครั้งของการส่งต่อของข้อมูลที่ได้รับเข้ามาไม่มากกว่าค่านี้

void addRouteTo (uint8_t dest, uint8_t next_hop, uint8_t state=Valid)

เพิ่มแอดเดรสของอุปกรณ์ที่ทำหน้าที่ router ตัวถัดไปเข้าในผังตารางการส่งต่อที่เก็บไว้ในตัวอุปกรณ์ ถ้าพื้นที่เก็บผังข้อมูลไม่เพียงพอ จะลบของมูลที่เก่าที่สุดออก

RoutingTableEntry * getRouteTo (uint8_t dest)

ตอบกลับค่าจากผังตารางการส่งต่อข้อมูลที่ได้เรียกดูมาจากอุปกรณ์ที่มีแอดเดรสตามที่ระบุ

bool deleteRouteTo (uint8_t dest)

ลบแอดเดรสที่ระบุไว้ออกจากผังตารางการส่งต่อข้อมูลที่อยู่ภายในตัวอุปกรณ์

void retireOldestRoute ()

ลบรายการในผังตารางการส่งต่อข้อมูลที่เก่าสุดออกจากผังตาราง

void clearRoutingTable ()

ลบรายการผังตารางการส่งต่อข้อมูลทั้งหมด

void printRoutingTable ()

แสดงค่าผังตารางการส่งต่อข้อมูลออกทางช่องทาง serial communication (เฉพาะในอุปกรณ์ที่รองรับการส่งค่าออกทางช่องทาง serial communication)

bool getNextValidRoutingTableEntry (RoutingTableEntry *RTE_p, int *lastIndex_p)

ค้นหาข้อมูลจากผังตารางการส่งต่อข้อมูล เริ่มต้นจากตำแหน่งที่กำหนดไว้โดย lastindex_p หรือเริ่มต้นจากรายการแรกของผังตารางถ้า lastindex_p ถูกกำหนดค่าเป็น -1

uint8_t sendtoWait (uint8_t *buf, uint8_t len, uint8_t dest, uint8_t flags=0)

ส่งข้อมูลไปยังอุปกรณ์ที่กำหนดแอดเดรสตามที่ระบุ โดยจะส่งไปยังอุปกรณ์ที่มีแอดเดรสตามที่ผังตารางเส้นทางได้ระบุไว้ และรอการตอบรับจากอุปกรณ์ส่งต่อตัวดังกล่าว

โครงสร้างการทำงาน

bool recvfromAck (uint8_t *buf, uint8_t *len, uint8_t *sourceL, uint8_t *dest=NULL, uint8_t *id, uint8_t *flags, uint8_t *hops)

รอรับข้อมูลขาเข้า ถ้าอุปกรณ์ไม่ได้อยู่ในสถานะรอรับข้อมูล จะสั่งการให้อุปกรณ์อยู่ในสถานะดังกล่าวก่อนการรอรับข้อมูลขาเข้า เมื่อได้รับข้อมูลเข้ามา จะตอบรับการได้รับข้อมูลกลับไปยังผู้ส่งตัวสุดท้าย และส่งกลับข้อมูลที่ได้รับ พร้อมทั้งค่าความยาวของข้อมูล และค่าแอดเดรส FROM, แอดเดรส TO, ID และ FLAG ที่อยู่ใน header ของข้อมูลที่ได้รับ และค่าจำนวนอุปกรณ์ router ที่ส่งต่อข้อมูลตั้งแต่ต้นทางมาถึงอุปกรณ์นี้

bool recvfromAckTimeout (uint8_t *buf, uint8_t *len, uint16_t timeout, uint8_t *source=NULL, uint8_t *dest, uint8_t *id, uint8_t *flags, uint8_t *hops)

ทำงานเช่นเดียวกับ recvfromAck แต่สามารถระบุระยะเวลาที่จะรอรับข้อมูล และจะตอบกลับไปยังโปรแกรมเรียกใช้งานถ้าระยะเวลาดังกล่าวหมดลง

Mesh API

uint8_t sendtoWait (uint8_t *buf, uint8_t len, uint8_t dest, uint8_t flags)

ส่งข้อมูลไปยังผู้รับตามแอดเดรสที่ระบุ และรอการตอบรับยืนยันการได้รับข้อมูลจากอุปกรณ์ที่ทำหน้าที่ route ตัวถัดไป

bool recvfromAck (uint8_t *buf, uint8_t *len, uint8_t *source, uint8_t *dest, uint8_t *id, uint8_t *flags, uint8_t *hops)

รอรับข้อมูลขาเข้า ถ้าอุปกรณ์ไม่ได้อยู่ในสถานะรอรับข้อมูล จะสั่งการให้อุปกรณ์อยู่ในสถานะดังกล่าวก่อนการรอรับข้อมูลขาเข้า เมื่อได้รับข้อมูลเข้ามา จะตอบรับการได้รับข้อมูลกลับไปยังผู้ส่งตัวสุดท้าย และส่งกลับข้อมูลที่ได้รับ พร้อมทั้งค่าความยาวของข้อมูล และค่าแอดเดรส FROM, แอดเดรส TO, ID และ FLAG ที่อยู่ใน header ของข้อมูลที่ได้รับ และค่าจำนวนอุปกรณ์ router ที่ส่งต่อข้อมูลตั้งแต่ต้นทางมาถึงอุปกรณ์นี้

bool recvfromAckTimeout (uint8_t *buf, uint8_t *len, uint16_t timeout, uint8_t *source, uint8_t *dest, uint8_t *id, uint8_t *flags, uint8_t *hops)

ทำงานเช่นเดียวกับ recvfromAck แต่สามารถระบุระยะเวลาที่จะรอรับข้อมูล และจะตอบกลับไปยังโปรแกรมเรียกใช้งานถ้าระยะเวลาดังกล่าวหมดลง

3. รูปแบบข้อมูล

Datalink Frame Format

ในแต่ละเฟรมข้อมูลที่ส่งออกอากาศ จะประกอบด้วยข้อมูลสองส่วนคือ Header และ Payload โดยข้อมูลแต่ละเฟรมจะมีจำนวนทั้งสิ้นไม่เกิน 256 ไบท์ แยกเป็นแต่ละส่วนดังนี้

Header	Payload
4byte	252byte

Header ข้อมูลกำหนดเส้นทางการรับส่ง และควบคุมการรับส่ง

Payload ข้อมูลที่ต้องการรับส่งระหว่างอุปกรณ์ต้นทางและปลายทาง

Header จะประกอบด้วยข้อมูลต่างๆ ดังนี้

TO-addr	FROM-addr	ID	Control flag
1byte	1byte	1byte	1byte

TO-addr แอดเดรสปลายทาง (destination address)

FROM-addr แอดเดรสต้นทาง (source address)

ID ตัวเลขเฉพาะ (unique ID) ที่ใช้เพื่อตรวจสอบการซ้ำกันของข้อมูล

Control flag ข้อมูลควบคุมการทำงาน

Payload ประกอบด้วยข้อมูลต่างๆ ดังนี้

Message	fcs
1..250byte	1byte

Message ข้อมูลที่ต้องการรับส่งระหว่างอุปกรณ์ในระดับถัดขึ้นไป

fcs ข้อมูลตรวจสอบความถูกต้องชนิด CRC-8 ข้อมูลที่มี fcs ถูกต้องเท่านั้น ที่จะได้รับการดำเนินการต่อไป

ข้อกำหนดของแอดเดรส TO-addr และ FROM-addr

0 ยังไม่มีการใช้งาน

1-247 อุปกรณ์ในเครือข่าย

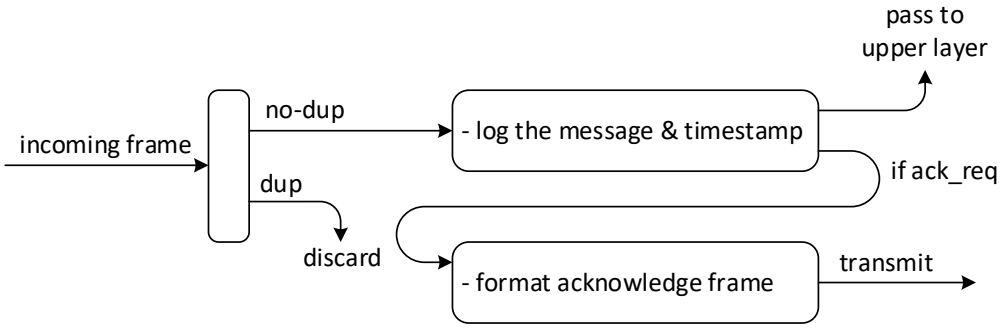
248-254 ยังไม่มีการใช้งาน

255 ไม่เจาะจงผู้รับ (broadcast address)

รูปแบบข้อมูล

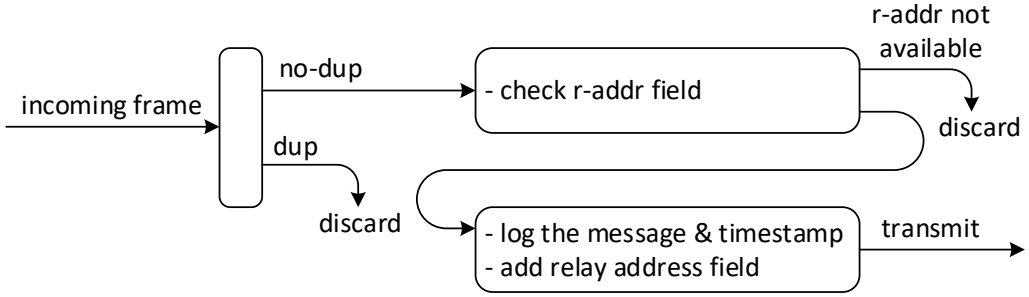
การส่งและรับข้อมูล

1. ผู้ส่งส่งข้อมูลโดยระบุหมายเลขแอดเดรสปลายทางในเฟรมข้อมูล และระบุ ack_req พร้อมกับ id
2. ข้อมูลได้รับโดย
   a. ผู้รับที่มีหมายเลขแอดเดรสประจำตัวตรงกับที่ระบุในเฟรมข้อมูล



ตรวจสอบว่าเป็นข้อมูลที่เคยรับเข้าภายในระยะเวลา 10 วินาทีที่ผ่านมาหรือไม่ ถ้าเป็นข้อมูลเดิม ให้ทิ้งข้อมูลดังกล่าว ถ้าเป็นข้อมูลใหม่ ให้ตรวจสอบข้อมูล ถ้ามี ack_req ให้ส่งข้อมูลยืนยันการรับข้อมูลกลับไปที่แอดเดรสต้นทางตามที่ระบุในเฟรมข้อมูล

   b. ผู้รับที่มีหมายเลขแอดเดรสประจำตัวไม่ตรงกับที่ระบุในเฟรมข้อมูล



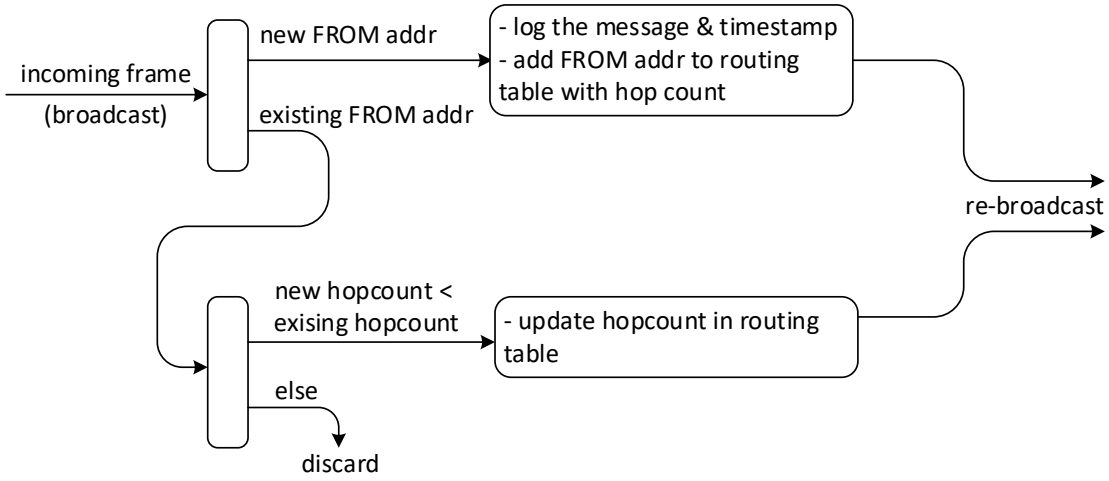
ตรวจสอบว่าเป็นข้อมูลที่เคยรับเข้าภายในระยะเวลา 10 วินาทีที่ผ่านมาหรือไม่ ถ้าเป็นข้อมูลเดิม ให้ทิ้งข้อมูลดังกล่าว ถ้าเป็นข้อมูลใหม่ ให้เก็บข้อมูลดังกล่าวลงใน local buffer พร้อมกับลง timestamp และเพิ่มแอดเดรสของตัวเองเข้าไปใน relay-addr ก่อนทำการส่งข้อมูลออกไป

การหาเส้นทาง

1. ผู้ส่งส่งข้อมูลโดยระบุหมายเลขแอดเดรสปลายทางในเฟรมข้อมูลเป็น 255 (broadcast address)
2. ผู้รับที่สามารถรับข้อมูลดังกล่าว ทำการตรวจสอบแอดเดรส FROM กับผังตารางเส้นทาง

รูปแบบข้อมูล

a. ถ้าไม่มีข้อมูลดังกล่าว ให้ทำการเพิ่มแอดเดรส FROM และแอดเดรส TO ลงในผังตารางเส้นทาง พร้อมกับค่าจำนวน hop count ที่อยู่ในข้อมูลที่ได้รับ แล้วส่งต่อข้อมูลดังกล่าวโดยใช้แอดเดรสของอุปกรณ์เป็นแอดเดรส FROM และเพิ่มค่า hop count ก่อนที่จะส่งต่อข้อมูล
b. ถ้ามีข้อมูลดังกล่าวอยู่แล้ว ให้ตรวจสอบค่า hop count ที่อยู่ในตารางเทียบกับค่าที่ได้รับเข้ามาใหม่ ถ้าค่าที่เข้ามาใหม่น้อยกว่าค่าที่มีอยู่ ให้ปรับค่า hop count ในตารางให้เป็นค่าใหม่ และส่งต่อข้อมูลดังกล่าวโดยใช้แอดเดรสของอุปกรณ์เป็นแอดเดรส FROM และใส่ค่า hop count ตามที่ได้รับเข้ามา แต่ถ้าค่าที่ได้รับเข้ามาใหม่เท่ากับ หรือมากกว่าค่าที่มีอยู่เดิม ให้ทิ้งข้อมูลดังกล่าวและไม่ต้องดำเนินการใดๆ



หมายเหตุ การหาเส้นทางควรจะดำเนินการเป็นช่วงๆ ตามความเหมาะสมกับการใช้งาน และระดับอัตราการใช้งานช่องความถี่ การดำเนินการหาเส้นทางเป็นช่วงระยะเวลาห่าง อาจจะทำให้ผังตารางเส้นทางไม่สอดคล้องกับลักษณะเครือข่ายปัจจุบัน ถ้ามีอุปกรณ์เพิ่ม หรือหายไปจากเครือข่ายในระยะเวลาดังกล่าว แต่ถ้าดำเนินเป็นช่วงระยะเวลาถี่จะทำให้อัตราการใช้งานช่องความถี่มากขึ้น และมีผลทำให้การส่งข้อมูลอาจจะมีความล่าช้ามาก เนื่องจากต้องรอช่องความถี่ว่างนานขึ้น

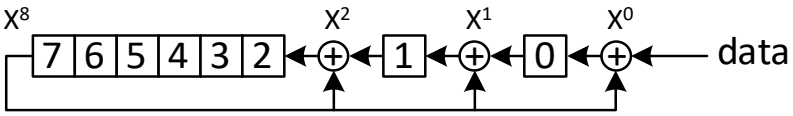
4. บรรณานุกรม

AN1200.22 LoRa Modulation Basic (https://lora-developers.semtech.com/documentation/product-documents/)

5. ภาคผนวก

การคำนวณ CRC

CRC เป็นการคำนวณโดยมองข้อมูลที่รับส่งเป็น polynomial แล้วทำการหารข้อมูลดังกล่าวด้วย polynomial ตามที่กำหนดไว้ ผลลัพธ์คงเหลือของการหารดังกล่าวคือค่า CRC ที่จะใช้เป็น fcs ประกอบกับการส่งข้อมูลแต่ละเฟรมต่อไป สำหรับ CRC-8 จะมี polynomial ตัวหารเป็น x^8+x^2+x+1 ขบวนการหารดังกล่าวสำหรับเลขฐาน 2 จะเป็นการ shift และ xor เท่านั้น ดังแสดงในรูปข้างล่าง



ในขั้นตอนดังกล่าว data คือข้อมูลที่ต้องการคำนวณหา CRC ที่เพิ่มบิท 0 จำนวน 8 บิทต่อท้ายข้อมูล หลังจากขบวนการ shift และ xor ข้อมูลทั้งหมดแล้ว ข้อมูลที่คงอยู่ใน shift register ก็คือ CRC ผลลัพธ์ ในการทำงานจริงบางครั้งจะมีการกำหนดค่าเริ่มต้นของข้อมูลใน shift register เป็นค่าอื่นที่ไม่ใช่ 0 และ/หรือมีการ xor ค่า CRC ผลลัพธ์ด้วยค่าคงที่ก่อนนำไปใช้งาน รวมทั้งอาจจะมีการกลับบิทของข้อมูลขาเข้าและ/หรือขาออก ขบวนการสร้างและขบวนการตรวจสอบจะต้องใช้วิธีการแบบเดียวกันเสมอ

การตรวจสอบ CRC ก็จะทำในลักษณะเดียวกัน คือนำค่าที่ได้รับมาทั้งหมดโดยไม่รวมข้อมูล fcs ไปทำการคำนวณหา CRC เปรียบเทียบข้อมูล fcs ที่ได้รับเข้ามา ซึ่งถ้าข้อมูลที่ได้รับเข้ามาถูกต้องค่าทั้งสองจะต้องตรงกัน

สำหรับการคำนวณด้วยซอฟท์แวร์ สามารถทำได้ดังนี้

วิธีที่ 1 คำนวณด้วยการ shift และ xor ตามปรกติ

```
uint8_t crc8(void *data, size_t size, bool reflectIn, bool reflectOut)
{
  uint8_t crc = INITIAL_REMAINDER;
  uint8_t *pos = (uint8_t *)data;
  uint8_t *end = pos + size;

  while (pos != end)
  {
    crc ^= (reflectIn ? Reverse(*pos) : *pos);
    for (int bit = 8; bit > 0; --bit)
    {
      if (crc & 0x80)
      {
        crc = (crc << 1) ^ polynomial;
      }
      else
      {
        crc <<= 1;
      }
    }
    pos++;
  }
  crc = (reflectOut ? Reverse(crc) : crc);
  return crc ^ FINAL_XOR_VALUE;
}
```

ภาคผนวก

วิธีที่ 2 ใช้การสร้างตารางข้อมูลเข้ามาช่วยในการคำนวณ เพื่อลดระยะเวลาที่ใช้คำนวณ

```
/* this table is generated using polynomial=0x07, reflectIn=false, reflectOut=false */
static const uint8_t crcTable[256] = {
    0x00,0x07,0x0E,0x09,0x1C,0x1B,0x12,0x15,0x38,0x3F,0x36,0x31,0x24,0x23,0x2A,0x2D,
    0x70,0x77,0x7E,0x79,0x6C,0x6B,0x62,0x65,0x48,0x4F,0x46,0x41,0x54,0x53,0x5A,0x5D,
    0xE0,0xE7,0xEE,0xE9,0xFC,0xFB,0xF2,0xF5,0xD8,0xDF,0xD6,0xD1,0xC4,0xC3,0xCA,0xCD,
    0x90,0x97,0x9E,0x99,0x8C,0x8B,0x82,0x85,0xA8,0xAF,0xA6,0xA1,0xB4,0xB3,0xBA,0xBD,
    0xC7,0xC0,0xC9,0xCE,0xDB,0xDC,0xD5,0xD2,0xFF,0xF8,0xF1,0xF6,0xE3,0xE4,0xED,0xEA,
    0xB7,0xB0,0xB9,0xBE,0xAB,0xAC,0xA5,0xA2,0x8F,0x88,0x81,0x86,0x93,0x94,0x9D,0x9A,
    0x27,0x20,0x29,0x2E,0x3B,0x3C,0x35,0x32,0x1F,0x18,0x11,0x16,0x03,0x04,0x0D,0x0A,
    0x57,0x50,0x59,0x5E,0x4B,0x4C,0x45,0x42,0x6F,0x68,0x61,0x66,0x73,0x74,0x7D,0x7A,
    0x89,0x8E,0x87,0x80,0x95,0x92,0x9B,0x9C,0xB1,0xB6,0xBF,0xB8,0xAD,0xAA,0xA3,0xA4,
    0xF9,0xFE,0xF7,0xF0,0xE5,0xE2,0xEB,0xEC,0xC1,0xC6,0xCF,0xC8,0xDD,0xDA,0xD3,0xD4,
    0x69,0x6E,0x67,0x60,0x75,0x72,0x7B,0x7C,0x51,0x56,0x5F,0x58,0x4D,0x4A,0x43,0x44,
    0x19,0x1E,0x17,0x10,0x05,0x02,0x0B,0x0C,0x21,0x26,0x2F,0x28,0x3D,0x3A,0x33,0x34,
    0x4E,0x49,0x40,0x47,0x52,0x55,0x5C,0x5B,0x76,0x71,0x78,0x7F,0x6A,0x6D,0x64,0x63,
    0x3E,0x39,0x30,0x37,0x22,0x25,0x2C,0x2B,0x06,0x01,0x08,0x0F,0x1A,0x1D,0x14,0x13,
    0xAE,0xA9,0xA0,0xA7,0xB2,0xB5,0xBC,0xBB,0x96,0x91,0x98,0x9F,0x8A,0x8D,0x84,0x83,
    0xDE,0xD9,0xD0,0xD7,0xC2,0xC5,0xCC,0xCB,0xE6,0xE1,0xE8,0xEF,0xFA,0xFD,0xF4,0xF3
};

uint8_t crc8(const void *data, size_t size, bool reflectIn, bool reflectOut)
{
  uint8_t crc = INITIAL_REMAINDER;
  uint8_t *pos = (uint8_t *)data;
  uint8_t *end = pos + size;

  while (pos != end)
  {
    crc = crcTable[crc ^ n];
    pos++;
  }
  return crc ^ FINAL_XOR_VALUE;
}
```

ซึ่งตาราง `crcTable` ข้างต้นสามารถสร้างได้ดังนี้

```
void GenerateTable(uint8_t polynomial, bool reflectIn, bool reflectOut)
{
  for (int byte = 0; byte < 256; ++byte)
  {
    uint8_t crc = (reflectIn ? Reverse(byte) : byte);
    for (int bit = 8; bit > 0; --bit)
    {
      if (crc & 0x80)
      {
        crc = (crc << 1) ^ polynomial;
      }
      else
      {
        crc <<= 1;
      }
    }
    crcTable[byte] = (reflectOut ? Reverse(crc) : crc);
  }
}

uint8_t Reverse(uint8_t value)
{
  value = ((value & 0xAA) >> 1) | ((value & 0x55) << 1);
  value = ((value & 0xCC) >> 2) | ((value & 0x33) << 2);
  value = (value >> 4) | (value << 4);
  return value;
}
```

6. การทดสอบ และผลการทดสอบ

Setup:

Hardware:

- TonySpace ESP32S3
- TS-COM-S76S(ACSip S76S)
- เสาอากาศ 900 MHz

Software:

- Libraries: TonyS_S3.h, TonySpace_LoRaMesh.h, RHRouter.h, RHMesh.h
- IDE: Arduino IDE
- Compiler/Platform: ESP32 environment

LoRa Configuration

- Frequency 924.0 MHz
- TX power 12
- Spreading Factor 8
- Bandwidth 125
- Preamble length 10
- Coding Rate 4/5

Node type:

- Sender
- Relay
- Receiver

Procedure:

Sender Node ทำหน้าที่หลักในการส่งข้อมูลไปยัง Node ตัวอื่น

ข้อมูลที่ใช้ทดสอบ(example payload) คือ “Hello from Node 1!”(ส่งจาก Node 1) และ “Hello from Node 2!”(ส่งจาก Node 2) โดยก่อนจะทำการส่งไปจะมีการแปลงเป็น ASCII code แล้วเก็บไว้เป็น uint8_t array หรือ type cast เป็น uint8_t pointer

Initialization:

1. เรียกใช้คำสั่ง Tony.begin() เพื่อใช้งาน IO Expander
2. เริ่มใช้งานโมดูล LoRa ด้วย TonyLORA.init(SLOTX) โดย SLOTX คือช่องที่ติดตั้งโมดูล LoRa
3. เรียกใช้งานตัวจัดการจาก RHMesh ตัวจัดการนี้จะทำหน้าที่จัดการรับส่งข้อมูลในระบบ โดยจะใช้ NODE_ADDRESS เป็นเลขประจำตัวของอุปกรณ์

การทดสอบ และผลการทดสอบ

Main flow:

1. Node จะพยายามส่งข้อมูลไปที่ปลายทาง (DEST_ADDRESS) ด้วยคำสั่ง sendtowait() ทุก 60 วินาที หากส่งข้อมูลสำเร็จฟังก์ชั่นจะคืนค่า RH_ROUTER_ERROR_NONE หมายความว่าไม่เจอความผิดพลาดในการส่งใดๆ แต่ถ้าทำการส่งไม่สำเร็จจะคืนค่า error แล้วลดเวลาในการรอส่งในรอบถัดไป
2. Node จะทำการจะทำการฟังข้อความที่รับได้ด้วยคำสั่ง recvfromAck() เมื่อมีข้อความเข้ามาจะทำการประมวลผล ถ้าหากเป็นข้อความที่เป็นของเราจะแสดงข้อความนั้นออกมา

Relay Node ทำหน้าที่ระบบการการส่งต่อข้อมูลเป็นหลัก ไม่ได้มีการทำส่งข้อความของตัวเอง

Initialization:

1. เรียกใช้คำสั่ง Tony.begin() เพื่อใช้งาน IO Expander
2. เริ่มใช้งานโมดูล LoRa ด้วย TonyLORA.init(SLOTX) โดย SLOTX คือช่องที่ติดตั้งโมดูล LoRa
3. เรียกใช้งานตัวจัดการจาก RHMesh ตัวจัดการนี้จะทำหน้าที่จัดการรับส่งข้อมูลในระบบ โดยจะใช้ NODE_ADDRESS เป็นเลขประจำตัวของอุปกรณ์

Main flow:

1. Node จะทำการจะทำการฟังข้อความที่รับได้ด้วยคำสั่ง recvfromAck() เมื่อมีข้อความเข้ามาจะทำการประมวลผล ถ้าหากเป็นข้อความที่เป็นของเราจะแสดงข้อความนั้นออกมา

Receiver Node เป็นปลายทางของการส่งข้อมูล

Initialization:

2. เรียกใช้คำสั่ง Tony.begin() เพื่อใช้งาน IO Expander
3. เริ่มใช้งานโมดูล LoRa ด้วย TonyLORA.init(SLOTX) โดย SLOTX คือช่องที่ติดตั้งโมดูล LoRa
4. เรียกใช้งานตัวจัดการจาก RHMesh ตัวจัดการนี้จะทำหน้าที่จัดการรับส่งข้อมูลในระบบ โดยจะใช้ NODE_ADDRESS เป็นเลขประจำตัวของอุปกรณ์

Main flow:

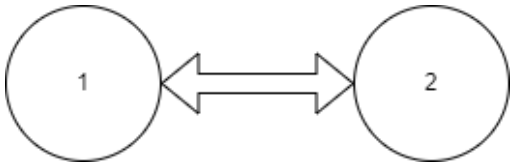
1. Node จะทำการจะทำการฟังข้อความที่รับได้ด้วยคำสั่ง recvfromAck() เมื่อมีข้อความเข้ามาจะทำการประมวลผล ถ้าหากเป็นข้อความที่เป็นของเราจะแสดงข้อความนั้นออกมาและกระพริบไฟ

การทดสอบ และผลการทดสอบ

Test case #1 การรับส่งข้อมูลระหว่างอุปกรณ์ 2 ตัว

Procedure:

จะใช้ Sender Node 1 ตัวและ Receiver Node 1 ตัว ให้ Sender Node มี NODE_ADDRESS = 1 และ Receiver Node มี NODE_ADDRESS = 2 ส่งข้อมูลไปที่ตัวข้างๆ โดยไม่ต้องกระโดด



Result:

Node 1 ต้องการส่ง payload ไปยัง Node 2 แต่ยังไม่มีเส้นทางจำทำการ broadcast discovery msg : ff010100ff0100056000010102 โดยข้อความจะแบ่งเป็น 3 ส่วนคือ Header, Routing header และ payload โดย header จะมีความยาว 4 byte และ routing header มีความยาว 5 byte ส่วน payload มีขนาดตาม message type

Header	Routing header	Payload
ff 01 01 00	ff 01 00 05 60 00	01 01 02

```
Sending to bridge n .2 res=ff010100ff01005600010102
```

Node 2 ได้รับ discovery msg จึงตอบด้วย discover response -> 0102010001020020000020102 การตอบกลับด้วย discovery response จะเปลี่ยน message type จาก 01(discovery request) เป็น 02(discovery response) แต่ข้อมูลอื่นจะคงเดิม

Header	Routing header	Payload
01 02 01 00	01 02 00 20 00	02 01 02

```
>> radio_rx ff010100ff01005600010102 -37 8

Clear buffer:
242
0
0102010001020002000020102
```

Node 1 ตอบ acknowledge Node 2 แล้วว่าได้รับเส้นทางแล้วจึงส่ง จะส่ง application message ไปยัง Node 2

Header	Acknowledge
02 01 01 80	21

การทดสอบ และผลการทดสอบ

```
>> radio_rx 0102010001020002000201O2 -27 8

Clear buffer:
0201018021
020102000201005700044865 6c6c6f2066726f6d204e6f6465203121
```

Node 2 ได้รับข้อความและตอบ Acknowledge Node 1 ว่าได้รับ payload แล้วก่อนที่จะแสดงผล payload ผ่าน serial monitor

Header	Routing header	Payload
02 01 02 00	02 01 00 57 00	04 48 65 6c 6c 6f 20 66 72 6f 6d 20 4e 6f 64 65 02 31 21

```
>> radio_rx 0201018021 -43 8

Clear buffer:

>> radio_rx 020102000201005700044865 6c6c6f2066726f6d204e6f6465203121 -34 8

Clear buffer:
0102028021
request from node n.1: Hello from Node 1! dest: 2 id: 87 flag: 0 rssi: -34
```

ข้อความที่ได้รับคือ “Hello from Node 1” จุดหมายปลายทางคือ Node address 2 (ตัวรับ)

Header	Acknowledge
02 01 01 80	21

```
>> radio_rx 0102028021 -20 8

Clear buffer:
0
Success route
```

เมื่อ Node 1 ได้รับ Acknowledge จาก Node 2 ถือว่าการส่งเสร็จสมบูรณ์และฟังก์ชั่น sendToWait() จะส่ง 0 คืนกลับมา โดย 0 มีความหมายว่า RH_ROUTER_ERROR_NONE

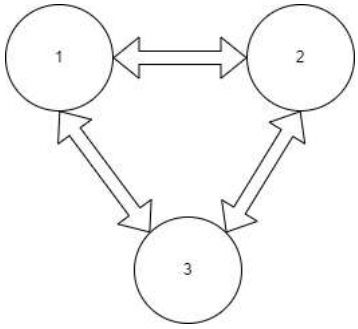
หากส่งไม่สำเร็จจะได้รับ response แบบอื่นจาก sentToWait()

- 1 = RH_ROUTER_ERROR_INVALID_LENGTH เมื่อข้อความยาวเกินกว่าที่จะส่งไปได้
- 2 = RH_ROUTER_ERROR_NO_ROUTE เมื่อหาเส้นทางไปยัง address ปลายทางไม่ได้
- 3 = RH_ROUTER_ERROR_TIMEOUT เมื่อใช้เวลานานเกินไป
- 4 = RH_ROUTER_ERROR_NO_REPLY เมื่อ address ปลายทางไม่ตอบกลับ
- 5 = RH_ROUTER_ERROR_UNABLE_TO_DELIVER เมื่อพยายามส่งไปที่ address ปลายทางที่เคยส่งได้แล้วไม่สำเร็จ

การทดสอบ และผลการทดสอบ

Test case #2 การรับส่งข้อมูลระหว่างอุปกรณ์ 3 ตัว (Triangle)

จะใช้ Sender Node 2 ตัวและ Receiver Node 1 ตัว ให้ Sender Node มี NODE_ADDRESS = 1,2 และ Receiver Node มี NODE_ADDRESS = 3 โดยจะให้ Sender Node ทั้ง 2 ตัวไปที่ Receiver Node ตัวเดียวกันและในเวลาเดียวกัน



Result:

Sender Node ทั้ง 2 ต้องการส่งข้อมูลไปที่ Node 3 ทั้ง 2 ตัวจึงพยายาม broadcast discovery message ไปที่ Node 3 เพื่อหาเส้นทางส่ง

```
Sending to bridge n .3 res=ff010200ff01005700010103
```

```
Sending to bridge n .3 res=ff020100ff02000200010103
```

Node 3 ได้ discovery message ของ Node 2 ก่อนจึงทำการติดต่อกับ Node 2 ก่อนด้วยการตอบ discovery response ไป

```
>> radio_rx ff020100ff02000200010103 -24 7

Clear buffer:
242
0
020301000203004d00020103
```

Node 2 เมื่อได้ discovery response จึงส่ง Acknowledge กลับไปที่ Node 3 ก่อนที่จะส่ง payload ไปที่ Node 3

```
>> radio_rx 020301000203004d00020103 -37 8

Clear buffer:
0302018021
0302020003020003000448656c6c6f2066726f6d204e6f6465203221
```

การทดสอบ และผลการทดสอบ

Node 3 ได้รับ Acknowledge จาก discovery response ที่ส่งไปที่ Node 2 จากนั้นก็ได้รับ payload จาก Node 2 และตอบ Node 2 ว่าได้รับ payload แล้วก่อนที่จะแสดงผล payload ผ่าน serial monitor

```
>> radio_rx 0302018021 -40 6

Clear buffer:

>> radio_rx 0302020003020003000448656c6c6f2066726f6d204e6f6465203221 -28 6

Clear buffer:
0203028021
request from node n.2: Hello from Node 2! dest: 3 id: 3 flag: 0 rssi: -28
```

Node 2 ได้รับ Acknoledge จาก Node 3 ถือว่าจบการส่งในส่วนของ Node 2

```
>> radio_rx 0203028021 -41 6

Clear buffer:
```

เมื่อได้รับข้อความของ Node 2 แล้วแต่ buffer ของโมดูล LoRa ยังมี discovery message ของ Node 1 อยู่จึงทำงานต่อเนื่องด้วยการตอบ discovery response กลับไปยัง Node 1

```
request from node n.2: Hello from Node 2! dest: 3 id: 3 flag: 0 rssi: -28

>> radio_rx ff010200ff01005700010103 -40 7

Clear buffer:
242
0
0103020001030004e00020103
0103024001030004e00020103
```

เนื่องจากการตอบ Node 2 ก่อนหน้าทำให้ discovery message ตัวก่อนหน้าไม่สามารถทำงานได้สำเร็จแต่ discovery response ที่ตอบมาช้ายังสามารถเพิ่มเส้นทางได้ ทำให้เมื่อลองส่งข้อความใหม่ไม่ต้องทำ discovery อีกรอบและสามารถส่งไปยังปลายทาง

```
sendtoWait failed. Are the bridge/intermediate mesh nodes running?

>> radio_rx 0103024001030004e00020103 -40 7

Clear buffer:
0301028021
Sending to bridge n .3 res=0301030003010058000448656c6c6f2066726f6d204e6f6465203121
```

Node 3 ได้รับข้อความที่ Node 1 ส่งมาและส่ง Acknowledge ไปยัง Node 1 ว่าได้รับข้อความแล้ว

```
>> radio_rx 0301030003010058000448656c6c6f2066726f6d204e6f6465203121 -36 8

Clear buffer:
010303c021
request from node n.1: Hello from Node 1! dest: 3 id: 88 flag: 0 rssi: -36
```

Node 1 ได้รับ Acknowledge จาก Node 3 เรียบร้อย จบการส่ง

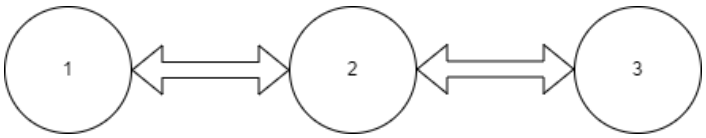
การทดสอบ และผลการทดสอบ

```
>> radio_rx 010303c021 -46 7

Clear buffer:
0
Success route
```

Test case #3 การรับส่งข้อมูลระหว่างอุปกรณ์ 3 ตัว แบบจำกัดการส่ง (1-1-1)

จะใช้ Sender Node 1 ตัว, Relay Node 1 ตัว และ Receiver Node 1 ตัว ทำการทดสอบโดยการให้ Sender Node กับ Receiver Node โดยทำให้ไม่สามารถส่งข้อความได้โดยตรง



Result:

เมื่อต้องการส่งไปที่ Node 3 แต่ยังไม่มีเส้นทาง Node 1 จึงทำการ Discovery ด้วย Broadcast discovery message ตามแผนผังการติดตั้งทำให้ Node 1 ไม่ส่งไป Node 3 ได้โดยตรง จึงต้องส่งผ่าน Node 2

```
Sending to bridge n . 3 es=ff010100ff01005600010103
```

Node 2 จะได้รับ Discovery message จาก Node 1 เมื่อตรวจสอบข้อความแล้วไม่ได้มีปลายทางเป็นตัวมันเองจึงเอา address ของตัวมันไปต่อกับ payload แล้วจึง Re-broadcast discovery msg ออกไป

```
>> radio_rx ff010100ff01005600010103 -24 8

Clear buffer:
0
Num route: 0
ff020100ff0100020001010302
```

Node 3 ได้รับ Discovery message จาก Node 2 เมื่อตรวจข้อความแล้วมีปลายทางเป็น Node 3 เมื่อทราบแล้วว่า Node 1 ต้องการเส้นทางไปที่ Node 3 ด้วย Discovery message จึงตอบกลับด้วย Discovery response ที่มีเส้นทางกลับไปยัง Node 1 โดยผ่าน Node 2

```
>> radio_rx ff020100ff0100020001010302 -41 8

Clear buffer:
1
Num route: 1
020301000103004d0002010302

>> radio_rx 0302018021 -44 5

Clear buffer:
```

Node 2 เมื่อได้รับข้อความจึงตอบ Acknowledge กลับไปที่ Node 3 ว่าทำการส่งสำเร็จแล้ว ข้อความที่ได้มาจาก Node 3 มีปลายทางคือ Node 1 จึงส่งต่อไปโดยไม่ได้ยุ่งกับข้อความที่จะส่งต่อไป

การทดสอบ และผลการทดสอบ

```
>> radio_rx 0203010001030004d0002010302 -43 8

Clear buffer:
0302018021
0102020001030014d0002010302
```

การทำ Discovery ของ Node 1 สำเร็จเมื่อได้รับเส้นทางจาก Discovery response จาก Node 3 เมื่อทราบแล้วว่าเส้นทางแล้วจึงส่งข้อความที่ต้องการไปยัง Node 3

```
>> radio_rx 0102020001030014d0002010302 -20 7

Clear buffer:
0201028021
020102000301005700044865 6c6c6f2066726f6d204e6f6465203121

>> radio_rx 0102028021 -15 7

Clear buffer:
0
Success route
```

Node 2 ตอบ Acknowledge Node 1 ว่าการส่งสำเร็จและส่งข้อความนี้ต่อไปที่ Node 3

```
>> radio_rx 020102000301005700044865 6c6c6f2066726f6d204e6f6465203121 -25 6

Clear buffer:
0102028021
030203000301015700044865 6c6c6f2066726f6d204e6f6465203121
```

Node 3 ตอบ Acknowledge Node 2 ข้อความที่ได้มาเป็นข้อความของมันเอง

```
>> radio_rx 030203000301015700044865 6c6c6f2066726f6d204e6f6465203121 -36 5

Clear buffer:
0203038021
request from node n.1: Hello from Node 1! dest: 3 id: 87 flag: 0 rssi: -36
```

Node 2 ถือว่าเป็นการส่งจบสมบูรณ์

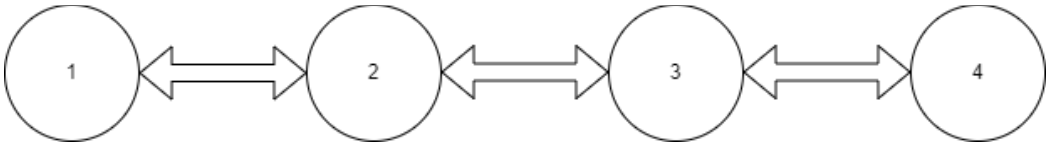
```
>> radio_rx 0203038021 -46 7

Clear buffer:
```

การทดสอบ และผลการทดสอบ

Test case #4 การรับส่งข้อมูลระหว่างอุปกรณ์ 4 ตัว แบบจำกัดการส่งแบบที่ 1 (1-1-1-1)

จะใช้ Sender Node 1 ตัว, Relay Node 2 ตัว และ Receiver Node 1 ตัว ทำการทดสอบโดยการให้ Sender Node กับ Receiver Node โดยทำให้ไม่สามารถส่งข้อความได้โดยตรง จะต้องวิ่งผ่าน relay node ทั้ง 2 ตัว



Result:

เมื่อต้องการส่งไปที่ Node 4 แต่ยังไม่มีเส้นทาง Node 1 จึงทำการ Discovery ด้วย Broadcast discovery message ตามแผนผังการติดตั้งทำให้ Node 1 ไม่ส่งไป Node 4 ได้โดยตรง จึงต้องส่งผ่าน Node 2 และ 3

```
Sending to bridge n .4 res=ff010200ff01005700010104
```

Node 2 ได้รับ broadcast discovery message จาก Node 1 แต่ตัวมันไม่ใช้ปลายทางจึงเพิ่ม address ของตัวเองไปที่ payload ส่วน route แล้วจึงทำการ Re-broadcast discovery message

```
>> radio_rx ff010200ff01005700010104 -23 6

Clear buffer:
0
Num route: 0
ff020100ff0100030001010402
```

Node 3 ได้รับ re-broadcast discovery message จาก Node 2 แต่ตัวมันไม่ใช้ปลายทางจึงทำการ Re-broadcast แต่ตัวมันไม่ใช้ปลายทางจึงเพิ่ม address ของตัวเองไปที่ payload ส่วน route แล้วจึงทำการ Re-broadcast discovery message อีกที

```
>> radio_rx ff020100ff0100030001010402 -40 7

Clear buffer:
1
Num route: 1
ff030100ff01006d000101040203
```

Node 4 ได้รับ re-boardcast ที่ถูกส่งผ่าน relay node 1 และ 2 (Node 2 และ 3) และตัวมันเป็นปลายทางจึงส่ง response ตอบกลับไปโดยจะส่งกลับไปตามทิศทางที่ส่งมา

```
>> radio_rx ff030100ff01006d000101040203 -25 6

Clear buffer:
2
Num route: 2
030401000104009600020104020 3
```

การทดสอบ และผลการทดสอบ

Node 3 ได้รับ discovery response จาก Node 4 และตอบ Acknowledge Node 4 ก่อนที่จะส่งต่อไปที่ Node 2

```
>> radio_rx 03040100010400960002010402O3 -36 8

Clear buffer:
0403018021
020302000104019600020104020 3
```

Node 4 ได้รับ Acknowledge จากการส่ง discovery response ไป Node 3

```
>> radio_rx 0403018021 -19 6

Clear buffer:
```

Node 3 ได้รับ discovery response จาก Node 4 และตอบ Acknowledge Node 4 ก่อนที่จะส่งต่อไปที่ Node 2

```
>> radio_rx 020302000104019600020104020 3 -38 7

Clear buffer:
0302028021
010202000104029600020104020 3

>> radio_rx 0201028021 -17 8

Clear buffer:
```

Node 1 ได้รับเส้นทางจาก discovery response และตอบ Acknowledge Node 2 ก่อนทำการทำการส่ง application message ไปทางที่ส่งผ่านไปทาง Node 2

```
>> radio_rx 010202000104029600020104020 3 -20 6

Clear buffer:
0201028021
Sending to bridge n .4 res=020103000401005800044865 6c6c6f2066726f6d204e6f6465203121
```

Node 2 ได้รับ Acknowledge จากการ่างต่อ discovery response

```
>> radio_rx 0201028021 -17 8

Clear buffer:
```

Node 2 ได้รับ application message ที่มีปลายทางเป็น Node 4 และตอบ Acknowledge Node 1 ก่อนที่จะส่งต่อไปที่ Node 3

การทดสอบ และผลการทดสอบ

```
>> radio_rx 020103000401005800044865 6c6c6f2066726f6d204e6f6465203121 -27 8

Clear buffer:
0102038021
0302030004010158000448656c6c6f2066726f6d204e6f6465203121
```

Node 1 ได้รับ Acknowledge จากการส่ง application message ไปที่ Node 2

```
>> radio_rx 0102038021 -17 6

Clear buffer:
0
Success route
```

Node 3 ได้รับ application message ที่มีปลายทางเป็น Node 4 และตอบ Acknowledge Node 2 ก่อนที่จะส่งต่อไปที่ Node 4

```
>> radio_rx 0302030004010158000448656c6c6f2066726f6d204e6f6465203121 -37 7

Clear buffer:
0203038021
0403030004010258000448656c6c6f2066726f6d204e6f6465203121
```

Node 2 ได้รับ Acknowledge จากการส่ง application message ไปที่ Node 3

Node 4 ได้รับ application message จาก Node 3 และตอบ Acknowledge Node 3

```
>> radio_rx 0403030004010258000448656c6c6f2066726f6d204e6f6465203121 -29 6

Clear buffer:
0304038021
request from node n.1: Hello from Node 1! dest: 4 id: 88 flag: 0 rssi: -29
```

Node 3 ได้รับ Acknowledge จากการส่ง application message ไปที่ Node 4

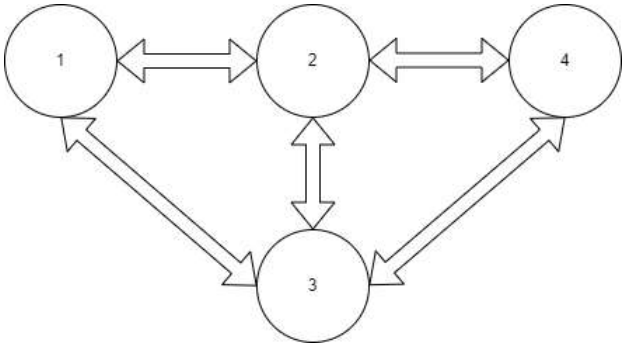
```
>> radio_rx 0304038021 -44 8
```

การทดสอบ และผลการทดสอบ

Test case #5 การรับส่งข้อมูลระหว่างอุปกรณ์ 4 ตัว แบบจำกัดการส่งแบบที่ 2 (1-2-1)

Step 1:

จะใช้ Sender Node 1 ตัว, Relay Node 2 ตัว และ Receiver Node 1 ตัว ทำการทดสอบโดยการให้ Sender Node กับ Receiver Node โดยทำให้ไม่สามารถส่งข้อความได้โดยตรง จะต้องวิ่งผ่าน relay node ตัวใดตัวหนึ่ง relay node สามารถส่งต่อกันได้หากใช้เวลาส่งน้อยกว่า



(Step 2):

เมื่อทำการส่งสำเร็จแล้วจะทำการปิด relay node ที่ส่งข้อมูลระหว่าง Node 1 และ 4 เพื่อให้ Node 1 ทำ discovery ใหม่ผ่าน relay node อีกตัวที่ไม่สามารถส่งได้ในตอนแรก

Result:

(Step 1):

เมื่อต้องการส่งไปที่ Node 4 แต่ยังไม่มีเส้นทาง Node 1 จึงทำการ Discovery ด้วย Broadcast discovery message ตามแผนผังการติดตั้งทำให้ Node 1 ไม่ส่งไป Node 4 ได้โดยตรง ไม่ได้มีการระบุไว้ว่า Node 2 หรือ Node 3 เพราะทั้ง 2 ตัวสามารถใช้งานเป็น relay node ได้

```
Sending to bridge n .4 res=ff010100ff01005600010104
```

Node 2 ได้รับ broadcast discovery message จาก Node 1 แต่ address ของ Node 2 ไม่ใช่ปลายทางจึงเพิ่มเพิ่ม address ของตัวเองก่อนจะ re-broadcast discovery message

```
>> radio_rx ff010100ff01005600010104 -27 7

Clear buffer:
0
Num route: 0
ff020100ff0100020001010402
```

Node 4 ได้รับ broadcast discovery message จาก Node 2 และ address ปลายทางเป็น address ของ Node 4 จึงตอบกลับด้วย discovery response ไปทาง Node 2

การทดสอบ และผลการทดสอบ

```
>> radio_rx ff020100ff0100020001010402 -9 -3

Clear buffer:
1
Num route: 1
020401000104001600020010402
```

Node 2 ได้รับ discovery response จาก Node 4 และตอบ Acknowledge Node 4 ก่อนที่จะส่งต่อไปที่ Node 1

```
>> radio_rx 020401000104001600020010402 -19 7

Clear buffer:
0402018021
0102020001040116000201040 2
```

Node 4 ได้รับ Acknowledge จาก Node 2

```
>> radio_rx 0402018021 -13 8

Clear buffer:
```

Node 1 ได้รับ discovery response และตอบ Acknowledge Node 2 เมื่อได้รับเส้นทางแล้วจึงส่งข้อความ application message ไปผ่าน Node 2 ไปยัง Node 4

```
>> radio_rx 0102020001040116000201040 2 -23 8

Clear buffer:
0201028021
020102000401005700044865 6c6c6f2066726f6d204e6f6465203121
```

Node 2 ได้รับ application mesasge จาก Node 1 จึงตอบ Acknowledge Node 1 และส่งต่อ application message ไปที่ Node 4

```
>> radio_rx 0201028021 -40 7

Clear buffer:

>> radio_rx 020102000401005700044865 6c6c6f2066726f6d204e6f6465203121 -29 7

Clear buffer:
0102028021
040203000401015700044865 6c6c6f2066726f6d204e6f6465203121
```

Node 1 ได้รับ Acknowledge จากการส่ง application message ไปที่ Node 2

การทดสอบ และผลการทดสอบ

```
>> radio_rx 0102028021 -17 8

Clear buffer:
0
Success route
```

Node 4 ได้รับ application message ที่ส่งต่อจาก Node 2 และตอบ Acknowledge Node 2 โดยตัวข้อความมี address ปลายทางตรงกับ Node 4

```
>> radio_rx 040203000401015700044865 6c6c6f2066726f6d204e6f6465203121 -20 8

Clear buffer:
0204038021
request from node n.1: Hello from Node 1! dest: 4 id: 87 flag: 0 rssi: -20
```

Node 2 ได้รับ Acknowledge จากการส่งต่อข้อความ application message

```
>> radio_rx 0204038021 -15 8

Clear buffer:
```

(Step 2) : Action: ทำการปิด Node 2

Node 1 ที่พยายามจะส่ง application message ไปที่ Node 4 ผ่าน Node 2 แต่เมื่อ Node 2 ถูกปิดไปทำให้ข้อความไม่สามารถไปถึง Node 4 และได้ฟังก์ชั่น sentToWait คืนค่า 5 ออกมาซึ่งหมายถึงไม่สามารถส่งข้อความได้สำเร็จ

```
Sending to bridge n .4 res=020103000401005800044865 6c6c6f2066726f6d204e6f6465203121
020103400401005800044865 6c6c6f2066726f6d204e6f6465203121
020103400401005800044865 6c6c6f2066726f6d204e6f6465203121
020103400401005800044865 6c6c6f2066726f6d204e6f6465203121
5
sendtoWait failed. Are the bridge/intermediate mesh nodes running?
```

Node 4 จึงต้องทำ discovery ใหม่เพิ่มหาเส้นทางในการส่งใหม่

```
Sending to bridge n .4 res=ff010500ff01005a00010104
```

Node 3 ที่ตอนแรกที่ไม่สามารถทำการส่งต่อ broadcast discovery message ได้เร็วกว่า Node 2 จึงเขามาหน้าที่ส่งต่อไปที่ Node 4 แทน

การทดสอบ และผลการทดสอบ

```
>> radio_rx ff010500ff01005a00010104 -45 8

Clear buffer:
0
Num route: 0
ff030300ff01004f0001010403
```

Node 4 ได้รับ re-broadcast discovery message จาก Node 3 และตอบ discovery response กลับไปที่ Node 3

```
>> radio_rx ff030100ff01004d0001010403 -25 7

Clear buffer:
1
Num route: 1
030401000104001600020104 03
```

Node 3 ได้รับ discovery response จาก Node 4 และตอบ Acknowledge Node 4 แล้วจึงส่งต่อ discovery response ไปที่ Node 1

```
>> radio_rx 030402000104001800020104 03 -24 8

Clear buffer:
0403028021
010304000104011800020104 03
```

Node 4 ได้รับ Acknowledge จากการส่ง discovery response ไปที่ Node 3

```
>> radio_rx 0403018021 -19 6

Clear buffer:
```

Node 1 ได้รับ discovery response และตอบ Acknowledge Node 3 เมื่อได้รับเส้นทางแล้วจึงส่งข้อความ application message ไปผ่าน Node 3 ไปยัง Node 4

```
>> radio_rx 010304000104011800020104 03 -44 8

Clear buffer:
0301048021
030106000401005b000448656c6c6f2066726f6d204e6f6465203121
```

Node 3 ได้รับ application message จาก Node 1 และตอบ Acknowledge Node 1 ก่อนที่จะส่งต่อ application message ไปยัง Node 4

การทดสอบ และผลการทดสอบ

```
>> radio_rx 030106000401005b000448656c6c6f2066726f6d204e6f6465203121 -41 8

Clear buffer:
0103068021
040305000401015b000448656c6c6f2066726f6d204e6f6465203121
```

Node 1 ได้รับ Acknowledge จากการส่ง application message ไปยัง Node 3

```
>> radio_rx 0103068021 -48 8

Clear buffer:
0
Success route
```

Node 4 ได้รับ application message ที่ส่งต่อจาก Node 3 และตอบ Acknowledge Node 3 โดยตัวข้อความมี address ปลายทางตรงกับ Node 4

```
>> radio_rx 040305000401015b000448656c6c6f2066726f6d204e6f6465203121 -26 8

Clear buffer:
0304058021
request from node n.1: Hello from Node 1! dest: 4 id: 91 flag: 0 rssi: -26
```

Node 3 ได้รับ acknowledge จาก Node 4 จากการส่งต่อ application message

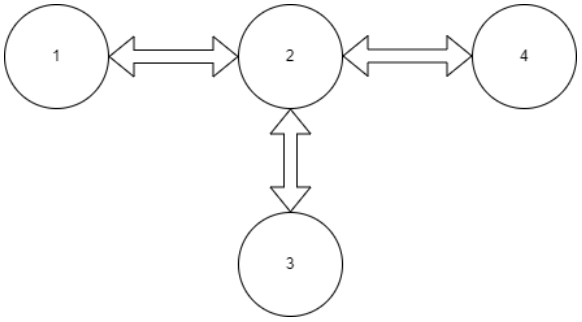
```
>> radio_rx 0304068021 -38 7

Clear buffer:
```

การทดสอบ และผลการทดสอบ

Test case #6 การรับส่งข้อมูลระหว่างอุปกรณ์ 4 ตัว แบบจำกัดการส่งแบบที่ 3 (Star)

จะใช้ Sender Node 1 ตัว, Relay Node 1 ตัว และ Receiver Node 2 ตัว ทำการทดสอบโดยการให้ Sender Node กับ Receiver Node โดยทำให้ไม่สามารถส่งข้อความได้โดยตรง จะต้องวิ่งผ่าน relay node



การส่งข้อความจะทำการส่งไปที่ Node 3 ก่อนเมื่อส่งสำเร็จจะทำการส่งไปที่ Node 4 เพื่อแสดงให้เห็นว่า

Result:

Node 1 ต้องการส่งไปที่ Node 3 แต่ยังไม่มีเส้นจึงทำการส่ง broadcast discovery message

```
Sending to bridge n .3 res=ff010100ff01005600010103
```

Node 2 ได้รับ broadcast discovery message จาก Node 1 และทำการ re-broadcast discovery message

```
>> radio_rx ff010100ff01005600010103 -23 8

Clear buffer:
ff020100ff0100020001010302
```

Node 3 ได้รับ discovery message จาก Node 2 และตอบกลับด้วย discovery response ไปยัง Node 2

```
>> radio_rx ff020100ff0100020001010302 -24 7

Clear buffer:
020301000103004d0002010302
```

Node 2 ได้รับ discovery response จาก Node 3 และตอบ Acknowledge Node 3 และส่งต่อ discovery response ไปที่ Node 1

```
>> radio_rx 020301000103004d0002010302 -34 7

Clear buffer:
0302018021
010202000103014d0002010302
```

Node 3 ได้รับ Acknowledge จากการส่ง discovery response ไปยัง Node 2

การทดสอบ และผลการทดสอบ

```
>> radio_rx 0302018021 -17 8

Clear buffer:
```

Node 1 ได้รับ discovery response จาก Node 2 และตอบ acknowledge Node 2 ก่อนที่จะส่งต่อไปที่ application message ไปยัง Node 4 ผ่าน Node 2

```
>> radio_rx 0102020001030l4d0002010302 -20 6

Clear buffer:
0201028021
02010200030100570004486566c6c6f2066726f6d204e6f6465203121
```

Node 2 ได้รับ Acknowledge จากการส่งต่อ discovery response และได้รับ application message จาก Node 1 จึงตอบ acknowledge Node 1 แล้วจึงส่งต่อไปที่ Node 3

```
>> radio_rx 0201028021 -18 8

Clear buffer:

>> radio_rx 02010200030100570004486566c6c6f2066726f6d204e6f6465203121 -28 8

Clear buffer:
0102028021
030203000301015700044866c6c6f2066726f6d204e6f6465203121
```

Node 1 ได้รับ acknowledge จากการส่ง application message ไปที่ Node 2

```
>> radio_rx 0102028021 -17 6

Clear buffer:
0
Success route
```

Node 3 ได้รับ application message จาก Node 2 และตอบ acknowledge ไปยัง Node 2 โดยตัวข้อความมี address ปลายทางตรงกับ Node 3

```
>> radio_rx 030203000301015700044866c6c6f2066726f6d204e6f6465203121 -33 7

Clear buffer:
0203038021
request from node n.1: Hello from Node 1! dest: 3 id: 87 flag: 0 rssi: -33
```

เมื่อส่งไปที่ Node 3 เสร็จแล้วจึงจะส่งไปที่ Node 4 จึงทำการหาเส้นทางโดยการ broadcast discovery message

```
Sending to bridge n .4 res=ff010300ff01005800010104
```

Node 2 ได้รับ broadcast discovery message จาก Node 1 และทำการ re-broadcast discovery message

```
>> radio_rx ff010300ff01005800010104 -21 8

Clear buffer:
ff020400ff0100030001010402
```

Node 4 ได้รับ discovery message จาก Node 2 และตอบกลับด้วย discovery response ไปยัง Node 2

```
>> radio_rx ff020400ff0100030001010402 -23 8

Clear buffer:
020401000104001600020104 02
```

Node 2 ได้รับ discovery response จาก Node 4 และตอบ Acknowledge กลับ Node 2 แล้วส่งต่อ discovery response ไปยัง Node 1

```
>> radio_rx 020401000104001600020104 02 -13 1

Clear buffer:
0402018021
0102050001040116000201040 2
```

Node 4 ได้รับ acknowledge จากการส่ง discovery response ไปยัง Node 2

```
>> radio_rx 0402018021 -17 8

Clear buffer:
```

Node 1 ได้รับ discovery response จาก Node 2 และตอบ Acknowledge กลับไปยัง Node 2 เมื่อได้เส้นทางแล้วจึงส่ง application message ไปยัง Node 4 ผ่าน Node 2

```
>> radio_rx 0102050001040116000201040 2 -21 6

Clear buffer:
0201058021
020104000401005900044865 6c6c6f2066726f6d204e6f6465203121
```

Node 2 ได้รับ acknowledge จากการส่ง discovery response ยัง Node 1

การทดสอบ และผลการทดสอบ

Node 2 ได้รับ appliacatione message จาก Node 1 และตอบ acknowledge Node 1 แล้วจึงส่งต่อไปที่ Node 4

```
>> radio_rx 0201058021 -19 8

Clear buffer:

>> radio_rx 020104000401005900044865 6c6c6f2066726f6d204e6f6465203121 -28 9

Clear buffer:
0102048021
040206000401015900044865 6c6c6f2066726f6d204e6f6465203121
```

Node 1 ได้รับ acknowledge จากการส่ง application ไป Node 2

```
>> radio_rx 0102048021 -17 6

Clear buffer:
0
Success route
```

Node 4 ได้รับ application message จาก Node 2 และตอบ acknowledge ไปยัง Node 2 โดยตัวข้อความมี address ปลายทางตรงกับ Node 4

```
>> radio_rx 040206000401015900044865 6c6c6f2066726f6d204e6f6465203121 -26 8

Clear buffer:
0204068021
request from node n.1: Hello from Node 1! dest: 4 id: 89 flag: 0 rssi: -26
```

Node 2 ได้รับ acknowledge จากการส่งต่อ application message ไปยัง Node 4

```
>> radio_rx 0204068021 -20 8

Clear buffer:
```