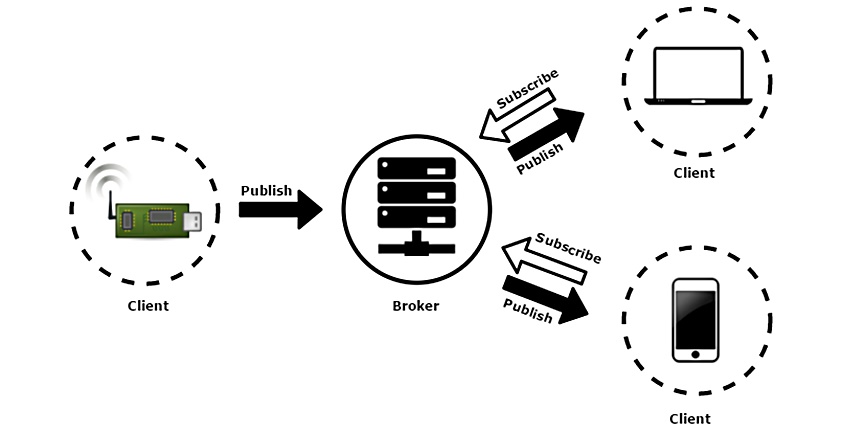
**TT (MQ TELEMETRY TRANSPORT)**

MQTT เป็นโพรโทคอลสื่อสารชั้นแอปพลิเคชั่นที่รันบน TCP/IP ถูกพัฒนาขึ้นในปี 1999 โดย IBM และ Eurotech สำหรับการมอนิเตอร์สถานะท่อส่งน้ำมันส่วนที่วางผ่านเขตทะเลทราย ด้วยการออกแบบให้เป็นการรับส่งข้อความที่มีน้ำหนักเบามากและเป็นโพรโทคอลเปิด ทำให้ในปัจจุบัน MQTT ถูกนำมาใช้แพร่หลายในการสื่อสารแบบ M2M หรือ IoT เพราะเหมาะสมกับอุปกรณ์ปลายทางที่มีขนาดเล็ก/พลังงานจำกัด หรือในการสื่อสารระยะไกลที่ต้องการการใช้งานแบนด์วิดธ์อย่างมีประสิทธิภาพ

โมเดลการสื่อสารของโพรโทคอลแสดงดังรูปที่ 1.3 ประกอบด้วย 2 ส่วนคือไคลเอนต์ และโบรกเกอร์

รูปที่ 1.3 โมเดลการสื่อสารของโพรโทคอล MQTT



**โบรกเกอร์** เป็นจุดศูนย์กลางในการรับส่งข้อความระหว่างไคลเอนต์ วิธีการกำหนดเส้นทาง (Routing) กระทำผ่าน Topic โดยไคลเอนต์ Subscribe ใน Topic ที่ตนต้องการ จากนั้นโบรกเกอร์จะส่งข้อความทั้งหมดที่ถูก Publish ใน Topic นั้นๆ ไปให้ ดังนั้นไคลเอนต์จึงสื่อสารกันได้โดยไม่จำเป็นต้องรู้จักกัน ช่วยลดความเกี่ยวพันระหว่างผู้สร้างข้อมูลและผู้ใช้ข้อมูล ส่งผลให้การขยายตัวของเครือข่ายทำได้ง่าย นอกจากนี้หน้าที่ที่สำคัญอีกประการของโบรกเกอร์คือการรักษาความปลอดภัยของไคลเอนต์ (Authorization, Authentication) ซึ่งในส่วนนี้สามารถขยายเพิ่มเติม หรือนำไปเชื่อมกับกลไกความปลอดภัยของระบบหลังบ้านที่มีอยู่แล้วได้ ช่วยให้นำโบรกเกอร์เข้าไปใช้งานเป็นส่วนหนึ่งของระบบอื่นๆ ได้ ส่วน Authorization ของ NETPIE ซึ่งจะได้กล่าวถึงต่อไปในหัวข้อที่ 1.4 ก็ถือเป็นตัวอย่างหนึ่งของการขยายเพิ่มเติมการรักษาความปลอดภัยของโบรกเกอร์ใน MQTT ปัจจุบันมีโบรกเกอร์ MQTT ที่เปิดให้ดาวน์โหลดไปใช้หรือดัดแปลงอยู่หลายรายได้แก่ Mosquitto, RabbitMQ, Erlang, VerneMQ ฯลฯ

**ไคลเอนต์** จะเป็นได้ทั้ง Publisher หรือ Subscriber หรือ Publisher/Subscriber พร้อมๆ กัน และจะเป็นอุปกรณ์ใดๆ ก็ได้ที่สามารถรัน MQTT Client Library บน TCP/IP Stack การที่ MQTT ใช้โมเดล Publish/Subscribe ตรรกะส่วนใหญ่จึงไปตกอยู่ในฝั่งโบรกเกอร์ ทำให้ Library มีขนาดเล็ก ติดตั้งได้ง่าย ใช้งานได้กับอุปกรณ์ที่มีทรัพยากรจำกัด ไคลเอนต์จำเป็นต้องเปิดการเชื่อมต่อ TCP ไว้ตลอดเพื่อที่โบรกเกอร์จะสามารถผลักข้อความไปให้ได้ หากการเชื่อมต่อถูกตัดขาด โบรกเกอร์จะเก็บข้อความทั้งหมดที่เข้ามาไว้จนกว่าไคลเอนต์จะกลับมาออนไลน์อีกครั้ง

เมื่อเปรียบเทียบ MQTT กับ HTTP (REST) ที่มีสถาปัตยกรรมแบบ Request/Response จะพบว่า MQTT มีความได้เปรียบที่โบรกเกอร์สามารถผลัก (Push) ข้อความไปยังไคลเอนต์ได้ตามเหตุการณ์ (Event-driven) ในขณะที่เมื่อใช้ HTTP ฝั่งไคลเอนต์ต้องคอยโพลข้อมูลเป็นระยะๆ และต้องตั้งค่าคาบเวลาการโพลไว้ก่อนล่วงหน้า โดยแต่ละครั้งต้องมีการสร้างการเชื่อมต่อขึ้นใหม่และอาจจะไม่มีข้อมูลใหม่ใดๆ ให้อัพเดท ดังนั้นหากต้องการให้ระบบทำงานแบบ Real Time หรือใกล้เคียง ย่อมหมายถึงต้องตั้งคาบเวลาการโพลให้สั้น และความสิ้นเปลืองของการใช้ช่องสัญญาณที่ไม่จำเป็นที่ตามมา นี่จึงเป็นอีกเหตุผลสำคัญที่ทำให้ MQTT ได้รับความนิยมเหนือ REST สำหรับการใช้งานแบบ M2M นอกเหนือจากการมีน้ำหนักเบา

**MQTT Topics**

**MQTT Topic** เป็น UTF-8 String ในลักษณะเดียวกับ File Path คือสามารถจัดเป็นลำดับชั้นได้ด้วยการขั้นด้วย “/” ตัวอย่างเช่น myhome/floor-one/room-c/temperature ไคลเอนต์สามารถเลือก Publish หรือ Subscribe เฉพาะ Topic หรือ Subscribe หลาย Topic พร้อมๆ กันโดยใช้ Single-Level Wildcard (+) เช่น myhome/floor-one/+/temperature หมายถึงการขอเขียนหรือรับข้อความ temperature จากทุกๆ ห้องของ myhome/floor-one หรือ Multi-Level Wildcard (#) เช่น myhome/floor-one/# หมายถึงการขอเขียนหรือรับข้อความทั้งหมดที่มี Topic ขึ้นต้นด้วย myhome/floor-one เป็นต้น

เราสามารถกำหนด Topic อย่างไรก็ได้ โดยมีข้อยกเว้นการขึ้นต้น Topic ด้วยเครื่องหมาย “$” ซึ่งจะจำกัดไว้สำหรับการเก็บสถิติภายในของตัวโบรกเกอร์เท่านั้น ดังนั้นไคลเอนต์จะไม่สามารถ Publish หรือ Subscribe ไปยัง Topic เหล่านี้ได้ โดยทั่วไป Topic เหล่านี้จะขึ้นต้นด้วย $SYS

| **MQTT vs Message Queues เรามักพบการสับสนระหว่าง MQTT กับ Message Queues โดยคนจำนวนไม่น้อยเชื่อว่า MQ ใน MQTT มาจากคำว่า Message Queueในความเป็นจริงแล้ว MQ ในที่นี้มาจากชื่อรุ่นผลิตภัณฑ์ที่รองรับโพรโทคอล MQTT ของ IBM ที่เรียกว่า MQ Series และ MQTT ไม่ได้ทำงานในลักษณะ Message Queue กล่าวคือข้อความใน MQTT จะถูกส่งให้กับไคลเอนต์ทั้งหมดที่ Subscribe ใน Topic ในขณะที่ข้อความใน Message Queue สามารถถูกดึงออกไปใช้งานโดยผู้ใช้เพียงรายเดียวเท่านั้น** |
| --- |
|  |

**MQTT Connections**

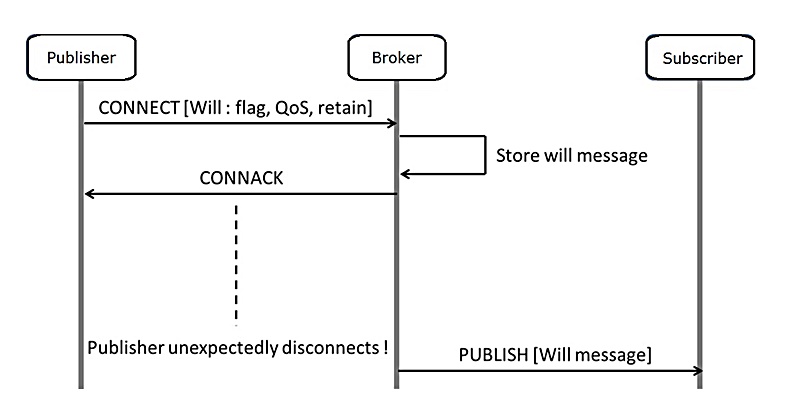
แพ็กเกตควบคุม (Control Packets) ทั้งหมดที่ใช้ในการสื่อสารระหว่างโบรกเกอร์กับไคลเอนต์ใน MQTT มีทั้งสิ้น 14 ชนิด แสดงไว้ดังตารางที่ 1.1

| **แพ็กเกตควบคุม** | **ผู้ส่ง(โบรกเกอร์)** | **ผู้ส่ง(ไคลเอนต์)** | **คำอธิบาย** |
| --- | --- | --- | --- |
| CONNECT |  | X | ขอเชื่อมต่อ |
| CONNACK | X |  | รับทราบการขอเชื่อมต่อ |
| PUBLISH | X | X | ข้อความที่จะขอ Publish |
| PUBACK | X | X | แจ้งว่าได้ Publish แล้ว (QoS Level 1) |
| PUBREC | X | X | แจ้งว่าได้ Publish แล้ว (QoS Level 2) |
| PUBREL | X | X | รับทราบว่าข้อความถูก Publish แล้วและลบให้อีกผู้รับลบค่าสถานะได้ (QoS Level 2) |
| PUBCOM | X | X | แจ้งว่า Publish เสร็จสิ้นและสถานะถูกลบ (QoS Level 2) |
| SUBSCRIBE |  | X | ขอ Subscribe |
| SUBACK | X |  | รับทราบการขอ Subscribe |
| UNSUBSCRIBE |  | X | ขอยกเลิก Subscribe |
| UNSUBACK | X |  | รับทราบการขอยกเลิก Subscribe |
| PINGREQ |  | X | PING Request |
| PINGRESP | X |  | PING Response |
| DISCONNECT |  | X | ขอยกเลิกการเชื่อมต่อ |

การเชื่อมต่อ MQTT จะเริ่มต้นจากฝั่งไคลเอนต์ส่งแพ็กเกตควบคุม CONNECT ไปยังโบรกเกอร์ โบรกเกอร์จะตอบรับด้วยแพ็กเกตควบคุม CONNACK วิธีนี้ช่วยแก้ปัญหาไคลเอนต์ที่ติดตั้งอยู่หลังเราท์เตอร์หรือไม่มีเลขไอพีสาธารณะ เพราะโบรกเกอร์มีเลขไอพีสาธารณะและจะรักษาการเชื่อมต่อสองทางไว้ตลอดหลังจากได้รับแพ็กเกตควบคุม CONNECT หากโบรกเกอร์พบว่าแพ็กเกต CONNECT ที่ได้รับไม่ถูกต้อง หรือไคลเอนต์ใช้เวลานานเกินไปนับตั้งแต่เปิดซ็อกเก็ตจนกระทั่งเริ่มส่งแพ็กเกต โบรกเกอร์จะปิดการเชื่อมต่อเพื่อป้องกันไคลเอนต์ไม่ประสงค์ดีที่ต้องการถ่วงการทำงานของโบรกเกอร์

ไคลเอนต์จะเป็นผู้ระบุค่าการเชื่อมต่อในแพ็กเกตควบคุม CONNECT ค่าเหล่านี้ ได้แก่

* Client ID เพื่อให้โบรกเกอร์ใช้ระบุตัวตนของไคลเอนต์และเก็บค่าสถานะเซสชั่นได้แก่ Subscriptions และข้อความทั้งหมดที่ไคลเอนต์ยังไม่ได้รับไว้ให้ (ส่วนนี้เกี่ยวข้องกับการตั้งค่าระดับ QoS ซึ่งจะกล่าวถึงต่อไป) ดังนั้น Client ID จึงจำเป็นที่จะต้องไม่ซ้ำกัน แต่หากไม่ต้องการให้โบรกเกอร์เก็บค่าสถานะ ไคลเอนต์สามารถเว้นส่วนนี้ว่างไว้ได้
* Clean Session มีค่า True หรือ False เพื่อระบุว่าไคลเอนต์ต้องการให้โบรกเกอร์รักษาค่าสถานะของเซสชั่นไว้ให้หรือไม่ หากต้องการ ให้ระบุค่าเป็น False หากไม่ ให้ระบุค่าเป็น True ดังนั้นหากไคลเอนต์ไม่ระบุค่า Client ID ในแพ็กเกต CONNECT จะต้องระบุ Clean Session เป็น True ด้วย มิฉะนั้นโบรกเกอร์จะไม่รับการเชื่อมต่อ
* Username และ Password เพื่อให้โบรกเกอร์ใช้ในการ Authentication และ Authorization ซึ่งตามมาตรฐานปัจจุบัน (MQTT 3.1.1) โพรโทคอล MQTT เองไม่มีการเข้ารหัสหรือแฮชในส่วนนี้ กล่าวคือ Username/Password จะถูกส่งเป็น Plaintext ดังนั้นจึงมีข้อควรระวังในการใช้ MQTT หากไม่มีชั้นความปลอดภัยเพิ่มเติมเช่น TLS ในชั้นทรานสปอร์ต
* Last Will Topic, Last Will QoS และ Last Will Message มีไว้ให้โบรกเกอร์ Publish ข้อความสั่งเสียสุดท้าย (Last Will Message) ไปยัง Topic ที่ระบุ (Last Will Topic) เพื่อให้ไคลเอนต์อื่นรับรู้ในกรณีการเชื่อมต่อของไคลเอนต์รายนี้ขาดลงแบบไม่เจตนา
* Keep Alive เป็นค่าตัวเลขคาบเวลาที่ไคลเอนต์ตกลงกับโบรกเกอร์ว่าจะส่งแพ็กเกตควบคุม PINGREQ มาเป็นระยะๆ ซึ่งโบรกเกอร์จะตอบด้วยแพ็กเกต PINGRESP เพื่อให้ทั้งสองฝ่ายรับรู้ว่าการเชื่อมต่อยังคงอยู่

รูปที่ 1.4 ผังการรับส่งข้อความสั่งเสียสุดท้าย (Last Will Message) [1](https://netpie.gitbooks.io/doc/content/second-chapter.html#fn_1)  


ไคลเอนต์ Publish ข้อความ โดยบรรจุลงในส่วน Payload ของแพ็กเกตควบคุม PUBLISH ซึ่งจะต้องระบุ Packet ID, ชื่อ Topic, ระดับของ QoS, Duplicate Flag และ Retain Flag โบรกเกอร์จะตอบกลับด้วยแพ็กเกต PUBACK หรือ PUBREC ขึ้นอยู่กับระดับ QoS ที่ระบุในแพ็กเกต PUBLISH ในทางกลับกันเมื่อต้องการรับข้อมูล ไคลเอนต์ส่งแพ็กเกตควบคุม SUBSCRIBE ไปยังโบรกเกอร์ โดยระบุรายชื่อ Topic ที่ต้องการ ซึ่งอาจมีได้มากกว่าหนึ่ง และสามารถเลือกตั้งค่าระดับ QoS ที่แตกต่างกันสำหรับแต่ละ Topic และโบรกเกอร์จะตอบด้วยแพ็กเกต SUBACK โดยยืนยันค่าระดับ QoS ของแต่ละ Topic ที่ไคลเอนต์ขอ Subscribe กลับมาอีกครั้ง หาก Topic ใดที่ไม่อนุญาตหรือไม่ปรากฏบนโบรกเกอร์ โบรกเกอร์จะตอบกลับด้วยค่า 128 แทนที่ค่าระดับ QoS

เมื่อไคลเอนต์ต้องการยกเลิกการรับข้อมูล ทำได้โดยการส่งแพ็กเกตควบคุม UNSCRIBE ไปยังโบรกเกอร์ โดยระบุรายชื่อ Topic ที่ต้องการบอกเลิกในคราวเดียวกันได้มากกว่าหนึ่ง โบรกเกอร์จะยืนยันการยกเลิกด้วยแพ็กเกต UNSUBACK

เมื่อไคลเอนต์ต้องการเลิกการเชื่อมต่อ ทำได้โดยการส่งแพ็กเกตควบคุม DISCONNECT ไปยังโบรกเกอร์ หากไคลเอนต์ CONNECT โดยตั้งค่า Clean Session เป็น True โบรกเกอร์จะยกเลิก Subscription ทั้งหมดของไคลเอนต์ให้เองโดยอัตโนมัติ ในทางกลับกันหาก Clean Session เป็น False โบรกเกอร์จะยังคงเก็บค่าต่างๆ ของเซสชั่นไว้ เมื่อไคลเอนต์เชื่อมต่อเข้ามาใหม่ด้วย Client ID เดิม จึงไม่จำเป็นต้องเริ่ม Subscribe ใหม่อีกครั้ง

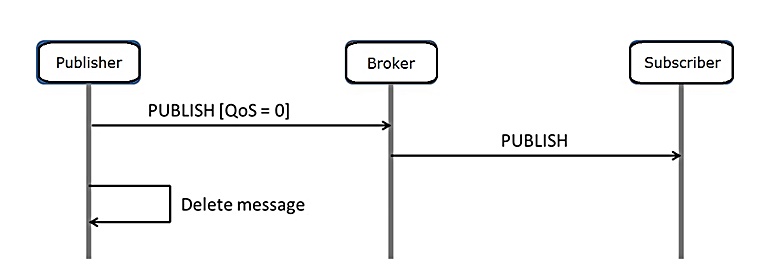
**MQTT Quality of Service (QoS)**

ไคลเอนต์จะเป็นผู้กำหนดระดับของบริการส่งและรับข้อความหรือ QoS ที่ตนต้องการในแต่ละ Topic ในแพ็กเกต PUBLISH หรือ SUBSCRIBE และโบรกเกอร์จะตอบสนองด้วย QoS ระดับเดียวกันสำหรับ Topic นั้นๆ

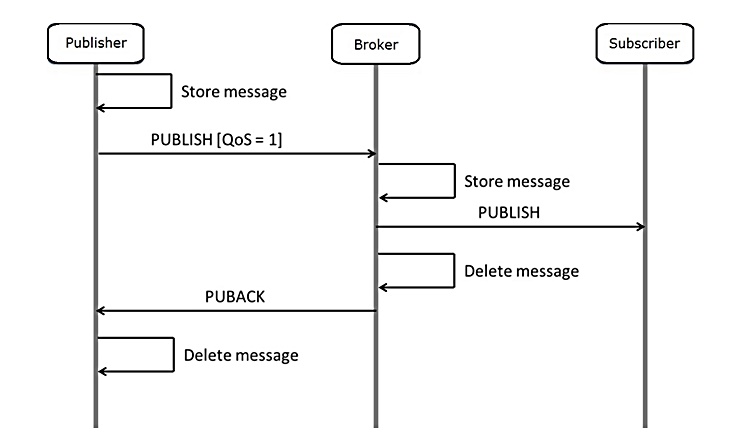
QoS ใน MQTT แบ่งได้เป็น 3 ระดับ คือ

1. อย่างมากหนึ่งครั้ง (At Most Once) แทนด้วยโค้ด 0 QoS 0 เป็นระดับบริการที่ต่ำที่สุด กล่าวคือไม่รับประกันว่าข้อความจะถูกส่งถึงผู้รับใดๆ เลยหรือไม่ หากไคลเอนต์ Publish ข้อความด้วย QoS 0 โบรกเกอร์จะไม่มีการตอบรับใดๆ ว่าได้ Publish ต่อไปให้ผู้รับรายอื่นหรือไม่ หากไม่มีผู้รับข้อความ โบรกเกอร์อาจเก็บข้อความไว้หรือลบทิ้งก็ได้ ขึ้นอยู่กับนโยบายของผู้ให้บริการเซิร์ฟเวอร์ ในทางกลับกันหากไคลเอนต์ที่เป็นผู้รับ Subscribe ไว้ด้วย QoS 0 เมื่อได้รับข้อความจากโบรกเกอร์ ก็ไม่ต้องส่งข้อความตอบรับใดๆ กลับ ทำให้การส่งข้อความแบบนี้รวดเร็วที่สุด เพราะไม่มีโอเวอร์เฮดในการตอบรับ ขณะเดียวกันหากข้อความถูกส่งไม่ถึง ก็ไม่มีทางทราบได้เช่นกัน
2. อย่างน้อยหนึ่งครั้ง (At Least Once) แทนด้วยโค้ด 1 QoS 1 รับประกันว่าข้อความจะถูกส่งถึงผู้รับอย่างน้อยหนึ่งครั้ง การส่งลักษณะนี้ ผู้ส่งจะเก็บข้อความเอาไว้ จนกว่าจะได้รับแพ็กเกต PUBACK จากผู้รับ ในกรณีไคลเอนต์ขอ Publish ผู้รับข้อความซึ่งเป็นโบรกเกอร์จะต้อง Publish ต่อไปยังไคลเอนต์ที่ Subscribe ไว้อย่างน้อยหนึ่งครั้ง จึงจะสามารถส่งแพ็กเกตตอบรับไปกลับยังผู้ส่ง ในกรณี Subscribe ผู้ส่งซึ่งก็คือโบรกเกอร์จะต้องเก็บข้อความไว้จนกว่าไคลเอนต์ที่ตนส่งข้อความไปให้จะยืนยันตอบรับ ดังนั้นแพ็กเกต PUBACK จึงต้องมีหมายเลขไอดีเดียวกับแพ็กเกต PUBLISH เพื่อให้ผู้ส่งทราบว่าข้อความใดถูกส่งถึงแล้วและสามารถลบออกได้
3. หนึ่งครั้งเท่านั้น (Exactly Once) แทนด้วยโค้ด 2 QoS 2 รับประกันว่าแต่ละข้อความจะถูกส่งถึงผู้รับเพียงหนึ่งครั้งเท่านั้น เป็นบริการที่ปลอดภัยที่สุดและช้าที่สุดของโพรโทคอล MQTT เนื่องจากผู้รับและผู้ส่งต้องส่งแพ็กเกตควบคุมไปกลับถึงสองรอบ เริ่มต้นด้วยผู้ส่งส่งข้อความไปในแพ็กเกต PUBLISH เมื่อผู้รับได้รับข้อความจะเก็บแพ็กเกตไว้และยืนยันกลับไปยังผู้ส่งด้วยแพ็กเกต PUBREC ผู้ส่งจึงสามารถลบข้อความนั้นของจากหน่วยเก็บข้อมูลของตนได้ และส่งแพ็กเกต PUBREL ไปยังผู้รับเพื่อให้ผู้รับสามารถลบสถานะการส่งข้อความนี้ออกได้ หากผู้รับเป็นไคลเอนต์ปลายทางที่ Subscribe ข้อความเอาไว้ ผู้รับจะส่งแพ็กเกต PUBCOM เพื่อยืนยันว่าข้อความถูกส่งถึงแล้วเรียบร้อยหนึ่งครั้ง หากผู้รับเป็นโบรกเกอร์ ทันทีที่ได้ Publish ข้อความต่อไปยังไคลเอนต์ปลายทางหนึ่งครั้ง จึงจะลบข้อความนั้นออก และปิดเซสชั่นด้วยการส่งแพ็กเกต PUBCOM กลับไปยังผู้ส่ง

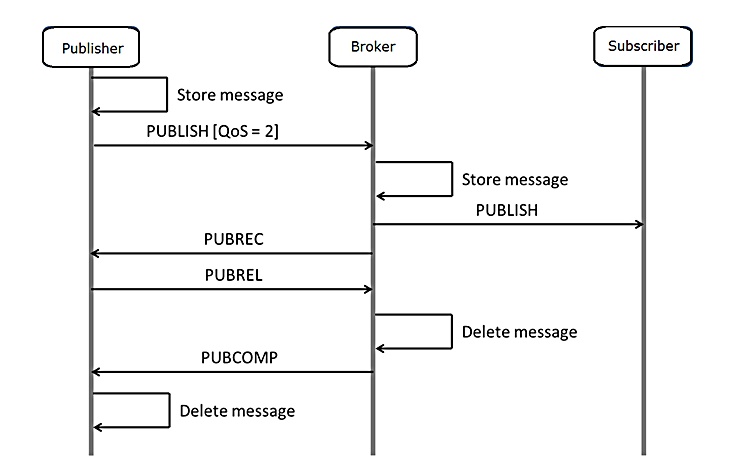
รูปที่ 1.5 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS 0[1](https://netpie.gitbooks.io/doc/content/second-chapter.html#fn_1)



รูปที่ 1.6 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS 1[1](https://netpie.gitbooks.io/doc/content/second-chapter.html#fn_1)



รูปที่ 1.7 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS[1](https://netpie.gitbooks.io/doc/content/second-chapter.html#fn_1)



**Retained Messages**

เมื่อไคลเอนต์ Publish ข้อความแบบ Retained Message โดยการตั้ง Retain Flag เป็น True จะทำให้โบรกเกอร์เก็บข้อความนั้นไว้ใน Topic ที่ระบุอย่างถาวร จนกว่าจะมี Retained Message อื่นที่ถูก Publish ภายหลังมาเก็บแทนที่ โดยโบรกเกอร์จะเก็บ Retained Message ไว้ให้เพียง Topic ละหนึ่งข้อความ ดังนั้นทุกครั้งที่มีไคลเอนต์ Subscribe เข้ามาใหม่ ก็จะได้รับ Retained Message ทันที ไม่ต้องรอจนกว่าจะมี Publication ใหม่เกิดขึ้น จึงมองได้ว่า Retained Message คือข้อมูลสุดท้ายที่ Subscriber ทุกรายควรต้องทราบ ดังนั้น Retained Message จึงเป็นประโยชน์อย่างยิ่งกับแอปพลิเคชั่นที่เกี่ยวข้องกับการอัพเดทสถานะ หากไคลเอนต์ต้องการลบ Retained Message ใน Topic ใดๆ ก็สามารถทำได้ไม่ยาก ด้วยการส่งแพ็กเกต PUBLISH ที่มีไม่มี Payload และตั้ง Retain Flag เป็น True ไปยัง Topic นั้นๆ หรือหากมีข้อมูลจะอัพเดท ก็ไม่มีความจำเป็นต้องลบ Retained Message เก่าออกก่อน แต่สามารถส่ง Retained Message เข้าไปเขียนทับได้เลย

**แบบฝึกหัด: Local MQTT Chat บน Raspberry Pi**

1.ติดตั้งโบรกเกอร์ Mosquitto ลงบน Raspberry Pi โดยพิมพ์ชุดคำสั่งข้างล่าง และแทนที่ xxxx ด้วยชื่อรุ่นระบบปฏิบัติการเช่น wheezy หรือ jessie

curl -0 http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key

sudo apt-key add mosquitto-repo.gpg.key

rm mosquito-repo.gpg.key

cd /etc/apt/sources.list.d/

sudo curl -0 http://repo .mosquitto.org/debian/mosquitto-xxxx.list

sudo apt-get update

2.ติดตั้งไคลเอนต์

sudo apt-get install mosquitto mosquitto-clients

3.ทดลองแชทโดยเปิดหน้าต่าง สร้างหัวข้อ “chat/test1” ลงบนโบรกเกอร์บนเครื่องและ Subscribe โดยพิมพ์

mosquitto\_sub –h localhost –t “chat/test1”

4.เปิดหน้าต่างใหม่โดยยังไม่ปิดหน้าต่างแรก พิมพ์คำสั่งข้างล่างเพื่อ Publish ข้อความ “hello” ไปยังหัวข้อ “chat/test1”

mosquitto\_pub –h localhost –t “chat/test1” –m “hello”

ในหน้าต่างแรกที่ได้ Subscribe เอาไว้ จะปรากฏข้อความ hello

5.สังเกตผลแตกต่างของการตั้งระดับบริการรับส่งข้อมูล (QoS) ที่แตกต่างกัน พิมพ์คำสั่งข้างล่างในหน้าต่างแรกเพื่อ Subscribe ในหัวข้อ “chat/test2” ด้วย QoS 1

mosquitto\_sub –h localhost –v –t “chat/#” –c –q 1 –i “Client ID”

6.ในหน้าต่างที่สอง พิมพ์คำสั่งข้างล่างเพื่อ Publish ข้อความ “You might get this message.” และ “You will get this message.” ด้วย QoS 0 และ QoS 1 ตามลำดับ

mosquitto\_pub –h localhost –t “chat/test2” –m “You might get this message.”

mosquitto\_pub –h localhost –t “chat/test2” –m “You will get this message.” –q 1

7.สังเกตผลลัพธ์ในหน้าต่างแรก

8.หยุดการทำงานของหน้าต่างแรกชั่วคราวโดย ctrl-c

9.ทำซ้ำข้อ 6

10.ทำซ้ำข้อ 5 และสังเกตผลลัพธ์