三、研究計畫內容（以中文或英文撰寫）：

(一) 研究計畫之背景。請詳述本研究計畫所要探討或解決的問題、研究原創性、重要性、預期影響性及國內外有關本計畫之研究情況、重要參考文獻之評述等。

1. 本研究計畫所要探討或解決的問題、研究原創與重要性

近年來資通訊技術快速發展，各式運算裝置已微型化並可直接或間接與網際網路互連，促成智慧環境 (Smart Environments) 願景實現。智慧環境中的「智慧」是由具計算能力的裝置與各式軟體服務透過網路連結起來，形成的物聯網 (Internet of Things, IoT) 所實現: 由佈建在環境中，相互連接的微型感測裝置 (Sensors) 感知環境及使用者資訊，傳送到具備較強運算與儲存能力的邊緣主機 (Edge Server) 儲存。再藉由這些感測資料推測使用者意圖，最後由環境控制或資訊呈現來滿足使用者的需求。

在智慧環境中，MQTT (Message Queuing Telemetry Transport)(Banks and Gupta, 2014) 是一個受到廣泛採用的應用層通訊機制。根據文獻記錄，MQTT 是在 1999 年由 Stanford-Clark 與 Nipper 為了人造衛星與石油管線感測器間通訊網路所設計的通訊協定 (Oweis et al., 2016)，此協定定義了裝置與 Broker 間通訊格式與功能規範，其訊息格式相當精簡，適用於處理器資源及網路頻寬受限的物聯網裝置。MQTT 是 ISO[1]與 OASIS (Organization Advancement Structured Information Standards)[2]標準，由 Eclipse 基金會 M2M Industry Working Group 管理。目前業界以 MQTT 3.1.1(Banks and Gupta, 2014) 為應用主流，最新版本為 MQTT 5(Banks et al., 2019)。MQTT 為適用於物聯網，具有幾項特殊的設計。首先，它具有 (樹狀) 階層式的 Topics，傳送到父節點的訊息會被直接轉送到所有子節點。這種特性讓訊息繞送具有更大的彈性，例如 Liao and Fu (2018) 的研究中，便藉由階層方式的特性實現了依據智慧空間的包含關係來繞送或區隔訊息的功能。此外，當節點失效時，broker 可透過遺囑 (Last Will) 功能，進行善後處理，具有讓整體系統 graceful degrading 的效果，增進了系統的強健度。最後，因應智慧環境的多變性，在不同應用情況下，有不同的傳送品質需求，MQTT 訊息中，包含了可動態調整的 QoS(Quality of Service) 機制，非常具有彈性。

從軟體架構角度來看，採用 MQTT 做為通訊基礎設施的應用程式採用的是 Messaging 樣式 (Hohpe and Woolf, 2004)。如圖 1右，一個基於 Messaging 樣式建構的系統，所有端點間的通訊都經過 MQTT Broker(在後面簡稱為 Broker) 進行交換，發佈訊息的端點將訊息發送到 Broker 中的特定主題 (Topic)，而訂閱訊息的端點只要訂閱特定主題就可以收到這些訊息。此種架構看似簡單，但它有效達成了端點間的時間解耦 (temporal decoupled) 與空間解耦 (spatial decoupled)。圖 1左是傳統以 RPC(Remote Procedure Call) 進行遠端呼叫 (X 呼叫 Y 與 Z)，然而，X 要能成功呼叫，Y 與 Z 必須在呼叫的同時皆正常運作; 反之，圖 1右的 Messaging 系統中，Y 或 Z 暫時失效時，X 傳送的訊息可暫時保留在 Broker，待失效端點修復後再送出訊息，從而達成了時間的解耦。此外，在圖 1右，X 傳訊息給 Y 與 Z 時，X 不需要知道 Y、Z 的參考 (如 IP、URL)，Y、Z 也不需知道 X 的參考，訊息均由 Broker 來轉送，因此具有空間解耦的特性。此外，Messaging 系統另一項好處是，連接 Broker 的各端點

[1]ISO/IEC 20922:2016 https://www.iso.org/standard/69466.html
[2]OASIS Message Queuing Telemetry Transport TC https://www.oasis-open.org/committees/mqtt/

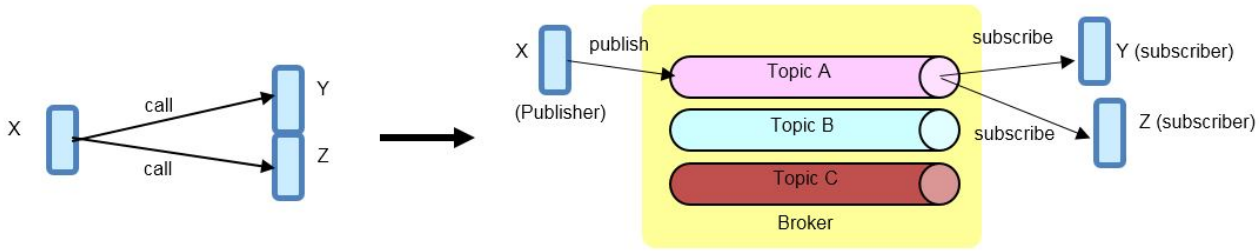

圖 1: RPC(Remote Procedure Call，左) 與 Messaging/Publisher-Subscriber(右) 對照示意圖

可以很容易使用不同語言實作並互通。

時間與空間解耦，一方面為系統帶來彈性與強健性，另一方面也對交易 (Transaction) 機制設計帶來非常大的挑戰。「交易」指的是一連串不可分割系統行為所形成的基本單位 (Unit of Work)，不是全部成功，不然就是全部放棄，通常須符合 ACID(Atomic, Consistent, Isolated, Durable) 屬性。交易在實務應用上是不可或缺的重要機制，以租車服務為例，選車、認證使用者、付費、解鎖等一系列步驟，就是一個交易，任何一個步驟失敗都必須徹消全部的動作。交易機制在單機資料庫系統已非常成熟 (Bernstein and Newcomer, 2009); 分散式交易雖然較為複雜，但也已經有廣被採用的標準做法，稱為兩階段確認 (Two-Phase Commit, 2PC)(Lampson, 1981)。在 RPC 分散式交易機制中，交易參與者要明確知道彼此位址，以同步 (Synchronized) 的方式進行，大部份情況下會預期短期內就可得知交易的結果。不幸的是，在 Messaging 系統中，進行分散式交易處理 (後面簡稱為「訊息式交易」) 時，時間與空間解耦: 交易參與者通常不會得知對方位址，彼此溝通也以非同步方式 (Asynchronized) 進行。換句話說，交易成員難以得知交易對象，也不能確保何時會有交易結果。更糟的是，對同一個 Broker 上的 Topic 來說，即使簡單如 request-reply 的訊息交換，也可能會互相交錯 (interleaved)，在交易的場合，對同一端點中資料做修改時，易形成 race condition，可能導致資料錯誤。上面議題只是就表面上對時間與空間解耦的影響推論得出，可想而之，訊息式交易機制有更多細節需要挑戰。值得注意的是，雖然目前 Messaging 系統無論在 Cloud 環境 (例如 Kafka[3]與 RabbitMQ[4]的普及、淘寶採用 RocketMQ[5]等) 或是 Edge 環境 (如 MQTT) 都非常重要，但訊息式交易相對於一般以 RPC 為基礎的分散式交易技術，在學術研究上的數量較少、進展較慢，技術也較不成熟完備 (Vargas et al., 2007; Liebig and Tai, 2001; Shatsky et al., 2008; Tai and Rouvellou, 2000)。訊息式交易的 ACID 屬性，甚至直到 2018 年才被明確定義 (Jergler et al., 2018)。

訊息式交易機制的設計，過去在 Vargas et al. (2007) 與 Shatsky et al. (2008) 中曾針對重要原則與一般性做法提出討論。首先要處理空間解耦的問題，換句話說，就是要確認有誰要參與交易，及這些參與端點的資訊 (例如: 存取位址、端點 id)。常見的做法是採用 Vargas et al. (2007) 提出的三階段交易: Census、Transaction 與 Commitment[6]。其中 Census 階段

[3]https://kafka.apache.org/
[4]https://www.rabbitmq.com/
[5]https://rocketmq.apache.org/
[6]「三階段交易」容易和前面提到的「兩階段交易」(2PC) 有名詞上的混淆。事實上，二個機制只是恰好名稱類似，它們是處於不同層次。2PC 可以看成是三階段交易中的 Transaction 與 Commitment 階段的一種實現方法。若不採用 2PC，也可以採用 SAGAS(Garcia-Molina and Salem, 1987) 來實現。

最主要的任務就是尋找並確認參與交易的端點。接下來要根據上述三個階段及相關設計決策，設計交易控制訊息格式。最後，還需設計交易處理與監控 (Transaction Processing and Monitoring, TPM) 機制，用戶端會向負責 TPM 的元件提交交易，完成後，TPM 會將結果回傳給用戶。此外，交易的對象端點上也必須部署 TPM(與部署在發起端的 TPM 邏輯會有些許不同)，這樣各端點才能依特定格式與規則接受、處理與回傳交易結果訊息。

前述文獻的不足，可分為兩個層次來陳述。首先，這些研究提出解決方案都是植基於特定的訊息系統所設計 (Pietzuch, 2004)，要讓物聯網的 Messaging 系統規格，如 MQTT，支援訊息式交易，只能參考其較高層次的設計理念，具體的細節尚待進一步設計釐清。其次，智慧環境與物聯網應用場域有其特殊性，這些特殊性只靠上述一般性原則並不足以因應，例如，在智慧境境/物聯網場域中，裝置運處與儲存能力的差異相當大，並非所有的端點 (裝置) 都有能力負擔與交易相關的運算與儲存空間，計算能力較弱的裝置，若無適當支援，根本無法參與交易。此外，部份物聯網裝置服務涉及環控，改變的是實體環境，沒辦法和資料庫的欄位值一般，可以立即修改、還原，這也是上述相關研究提出之機制所未能處理的。

由上述討論，我們知道訊息式交易的重要性，又因為它比起傳統 RPC 為主分散式交易，只有較少的研究被提出，設計在專屬的訊息系統上，而且這些研究並未考量物聯網場域的特殊需求。因此本計畫以發展基於物聯網標準規格設計訊息式交易機制為主軸，考量上述議題，預期技術貢獻如下:

- **基於 MQTT 5 的訊息式交易:** 考量 MQTT 規格在物聯網的普及，我們將參考現有文獻提出的訊息式交易架構，以 MQTT 5 (Banks et al., 2019) 為基底加以擴充，設計兼顧相容性的訊息式交易機制。目前大部份基於 MQTT 的改良研究都還以 MQTT 3.1.1 (Banks and Gupta, 2014) 為主，然 MQTT 3.1.1 未考量擴充或使用者自訂 header 的可能性，導致大部份關於 MQTT 改良的相關研究，或多或少都破壞了相容性，幾乎沒辦法用在實際場景; 反之，MQTT 5 則提供了通用的擴充機制。因此，本計畫將採用 MQTT 5 為基底來設計，其原因一方面是目前此版本研究尚少，可使研究成果在相關領域居於較領先地位; 另一方面，MQTT 5 許多新功能，提供設計較好解決方案的契機，又可不破壞協定的相容性。

- **支援部份與完整提交:** 在訊息式交易中，一個參與交易的端點是否可以在未完成一個交易前，就先處理下一個交易，開發人員必須在交易效率與正確性之間做權衡。有時為了提升效能，也有可能採用 SAGAS (Garcia-Molina and Salem, 1987)(而非 2PC)，在 prepare 階段就先提交，若最後 abort 再將所做的修改進行補償動作 (Compensation)。在失敗定義方面，交易失敗除了業務邏輯失敗外，也可能因為網路或 Broker 造成的訊息漏失、訊息版本錯誤或格式錯誤。

- **讓資源受限裝置也可參與交易的彈性部署機制:** 物聯網中並非所有的端點 (裝置) 都有能力負擔與交易相關的運算與儲存空間，因此我們希望交易能支援二種機制，依據「交易管理」邏輯實現的地方，區分為「端點主控交易 (Endpoint-Managed Transaction, EMT)」與「Broker 主控交易 (Broker-Managed Transaction, BMT)」。針對計算能力較弱的節點，由 Broker 上依需求建立交易代理人參與交易。

- **考量物聯網特殊性的交易模式:** 大部份現有研究都假設服務在交易確認後，都可以藉由補償 (compensation) 回復，但實務上並不一定，例如租用物品一旦解鎖，就可能被使用或取走。此外，在部份執行失敗時，有些具備 idempotent 性質的服務可提供重試 (retry) 的機會，以增加交易成功率。智慧環境中有些動作失敗時是可以重覆嘗試的，例如解鎖/上鎖，或是更新到同一個狀態。但有些動作是不可重覆執行 (non-idempotent) 的，例如環控的加溫、加濕、計數器、付費處理，這些動作在邏輯上做一次和做多次其結果是有區別的。上述這些交易的細節也會影響到智慧環境中交易的效能與正確性，應該予以考量。

綜合上述，本計畫主要目的為發展一個適用於智慧環境/物聯網應用場域的訊息式交易機制，此交易機制因應物聯網應用的多變需求，支援多種交易 (部份與完整提交) 與部署型式 (EMT/SMT)。同時，考慮智慧環境裝置提供服務的特殊性，在設計時也將交易放棄或失敗後的交易補償 (compensation)、錯誤重試與服務衝突提供了相關的處理機制。為提高設計的通用性，將基於 MQTT 5 加以擴充，提出兼顧相容性的訊息式交易機制，並將研究成果實現為原型系統與設計案例，供未來研究人員或業界研發團隊使用。敝研究團隊深耕物聯網/智慧環境相關協定 (如 UPnP、mDNS/DNS-SD、CoAP 與 MQTT) 多年，對相關通訊協定與實作熟悉，2017 年與 2019 年所提之專題研究計畫，均以 MQTT 為主題，研究成果亦已整理發表成為期刊技術貢獻之一部份 (Liao and Chen, 2019)，所累積的研究經驗與技術成果，預期將有助計畫進行。

2. 預期影響性

- MQTT 目前已成為普及的物聯網應用層通訊機制，在顧及相容性，不改變通訊協定的前提下，本計畫擬設計並實現 MQTT 5 上的訊息式交易機制，未來在業界相關實務應用，有機會成為關鍵技術 (如租用服務付款)。

- 初步文獻探討顯示，目前尚無類似技術被提出，且目前 MQTT 5 剛被提出 1 年多，相關研究仍少，成果若儘速發表，可望於相關領域居於較領先地位。

- 對於產業來說，目前較少智慧環境、IoT 等新興領域應用研究與技術開發人員，本計畫預期可針對此一方向加以探索與實作，使參與研究的成員累積更多專業知識與實務經驗，訓練更多具相關產業技術之人才。

3. 國內外有關本計畫之研究情況及重要參考文獻評述

在電子與通訊技術高度發展的今日，在市場上也出現愈來愈多將小型控制與感測裝置嵌入的日常生活物品，這些物品由通常具備一定程度的計算、儲存與通訊能力，因此又稱為智慧物品 (Smart Things)。這些智慧物品之間，通常藉由各式異質網路彼此通訊，並可藉由閘道器與 IP 網路上的其它服務連結，形成物聯網 (Internet of Things, IoT)，配合適當的應用程式加與整合，陸續實現了許多智慧環境，例如智慧家庭、智慧化博物館、智慧教室、智慧工廠與停車場。

要建置一個完整物聯網/智慧環境系統，各式裝置與通訊協定還是經由軟體加以整合。其架構大致可分二類。第一類架構依循傳統分散式系統 RPC(Remote Procedure Call) 方式進行元件間通訊及整合，通常會由一協調程序依序呼叫位於遠端的服務。早期許多智慧環境與物聯網系統均以此方式設計。正如 Saif and Greaves (2001) 所指出，此類架構會產生時間和空間的耦合，難以符合智慧環境動態改變的特性，系統的穩定性及彈性較低。反之，另一類架構主要的構想是在元件之間引入一個抽象的資料交換暫存區，所有元件均依賴中央資料交換區的位置，而不需綁定彼此的位置，此外，中央資料交換區也具備 buffer 的效果，可先暫時保留資料，因此可使得系統具有非同步的行為，從而降低了時間和空間上的相依性。這類系統主要有 Tuple Space(Gelernter, 1985) 和訊息式系統 (Messaging System)(Hohpe and Woolf, 2004) 二種方式。以智慧環境的特質來考量，訊息式系統在效能及相容性 (Interoperability) 上被認為優於 Tuple Space(Grimm, 2004)，因此近年來 Messaging 漸成為智慧環境與物聯網領域的主流架構。

本計畫聚焦於 MQTT 技術研究，目前針對 MQTT 的研究可分成三類，第一類著重於 MQTT 的實作與改良，Zare and Iqbal (2020) 和 Kodali (2016) 分別在 ESP32 和 CC3200 特特定硬體平台實作了輕量級的 MQTT Broker。Pipatsakulroj et al. (2017) 則利用 Linux 核心的 epoll 機制實作了稱為 muMQ 的 MQTT Broker，在不同流量下，其效能比 MQTT 的參考實作 Mosquitto[7]快 1.5 到將近 4 倍。第二類研究為針對 MQTT 協定本身加以修改或擴充，藉以加強資料即時性 (Timeliness)(Jo and Jin, 2015)、延展性 (Scalability)(Longo et al., 2020)、安全性 (Dikii, 2020)、吞吐量 (Throughput)(Banno et al., 2017) 與封包丟失率 (Wu and Li, 2018)。第三類則為基於 MQTT 的應用與實現，例如智慧工廠 (Iglesias-Urkia et al., 2017)、公眾存取開放資料集 (Open Data)(Tantitharanukul et al., 2017)、地理資訊系統 (Kawaguchi and Bandai, 2020) 與具自主設定與管理 MQTT 功能的智慧家庭閘道器 (Kim et al., 2015)。

本計畫將在 MQTT 上設計訊息式交易機制。有關一般性 RPC 為主的交易機制的設計與實現，Bernstein and Newcomer (2009) 已提出非常完備的參考指引，在此書中有一節提到 Queued Transaction，然而它只描述的最基本的單節點與單服務間交易機制。相較之下，訊息式交易機制，一般來說研究數量較少且進展較慢。Tai and Rouvellou (2000) 是相關議題較早期的研究，主要探討的重點是如何將 Messaging System 與分散物件交易系統整合，提出 MMT(Middleware Mediated Transactions) 機制 (Liebig and Tai, 2001)。針對時間與空間的解耦這個經典問題，比較重大的突破是出自於 Vargas et al. (2007)，提出透過 Census 協商階段來解決無法認知交易對象問題，但它是植基於特定的產品所設計。Shatsky et al. (2008) 延續此研究，提出了 TOPS(Transaction-Oriented Publish/Subscribe) 機制，它比較值得注意的改善是在於限制交易的參與者與交易的影響範圍 (Transaction Scope)，相關機制也是基於特定的產品所設計。基於問題的複雜性，相關研究在過去長期停留在實作性的探索，直到 2018 年才由 Jergler et al. (2018) 提出訊息式交易的 ACID 屬性的正規定義。本計畫主要探索的範圍是適用於智慧環境/物聯網的訊息式交易，就我們所知，到計畫提出為止，並沒有相關議題的研究成果被提出。

[7]https://mosquitto.org/

(二) 研究方法、進行步驟及執行進度。請分年列述：1. 本計畫採用之研究方法與原因及其創新性。2. 預計可能遭遇之困難及解決途徑。3. 重要儀器之配合使用情形。4. 如為須赴國外或大陸地區研究，請詳述其必要性以及預期效益等。

本節首先提供 MQTT 技術背景資訊，接下來簡介正規表述式及如何延伸 MQTT，設計訊息式交易機制構想、研究步驟/階段的進行方法與原因，最後說明可能遭遇之困難及解決途徑。

1. MQTT 技術背景

本節主要目的為就之前未詳述，但在後面會用到的 MQTT 技術細節加以介紹。如同之前提過的，MQTT 定義了裝置與 Broker 間通訊格式與功能規範，其訊息格式相當精簡，因此適合用於處理器資源及網路頻寬受限的物聯網裝置。其主要特色為 (1) 透過 publish 與 subscribe 的方式進行通訊，且 (2) 具備 QoS 選定機制，可依封包的重要性動態調整；(3) 其封包標頭長度為 2 bytes，因此可以減少封包在傳送時的額外負載，另外，當某個端點異常斷線時，MQTT 具有遺囑 (Last Will) 功能，會由 Broker 代替通知由斷線端點事先選定的 topic，告知訂閱者端點斷線之事實。因此在智慧環境應用上，MQTT 具有高度得靈活性與實用性，且在諸多實際情況的應用中，都可以透過其本身提供的相關機制，輕易地解決問題。MQTT 為了彈性對應各種不同傳輸需求，除了傳統常見的不可靠 (QoS-0, At most once) 與可靠 (QoS-1, At least once) 兩種傳輸方法外，MQTT 還多了另一種更為精確的傳遞方法-確定一次 (QoS-2, Exactly once) 的傳輸方式。MQTT 5.0 (Banks et al., 2019) 針對前版 MQTT 的可擴充性、錯誤回報能力與安全性的議題進行了改良。在此只針對和研究目標相關的二個項目簡要說明:

- **Request-response 模式:** Request-response 主要的用途是在 Messaging 系統上模擬遠端程序函式的叫用。這個概念舊版 MQTT 並無直接支援，MQTT 5 提供了 Response Topic 與 Corelation ID 等機制來支援 Request-response 的實現。舉例來說，假設要模擬端點 A 呼叫端點 B 上的 int add(int i, int j) 的服務，計算 add(1,1) 的值，只要由 A 發送 PUBLISH 給 B，其中帶入 $i = 1, j = 1$，並在 user properties 中設定 response topic 與 Corelation ID 屬性，B 收到訊息並計算出答案後，回傳到 response topic，並附帶 Corelation ID 於訊息中，如此一來 A 就能透過訂閱 Response Topic，與使用 Corelation ID 來辨識出之前送出 Request 的回應。基於規格相容的考量，本計畫提出的交易機制採用 MQTT 5 的 Request-response 模式做為主要的服務使用模式。換句話說，一個交易會視為由多個「request-response」的「序對 (ordered pairs)」組成，且以 MQTT 5 的 Request-response 機制實現。

- **User Properties:** 新規格在標頭中引入了 User Properties 機制，定義一系列預設封包的控制屬性，例如 Session expiry interval、Message expiry interval、Maximum QoS 等，並可讓使用者自訂，大幅增進了 MQTT 的可擴充性。本計畫為實現交易所提出的相關擴充模式、部署策略相關指令，即是以此為載具。為防此封包因自訂屬性變得過大，可以使用預定義的 Maximum Packet Size 來設限。

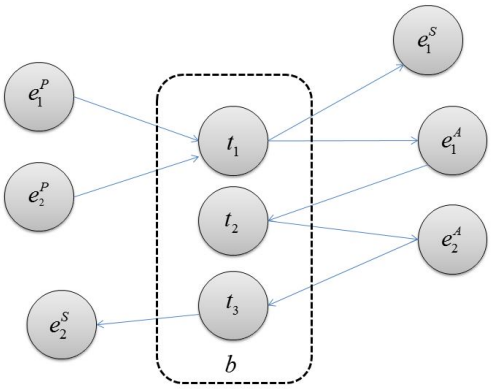

圖 2: MQTT 系統模型示意圖

2. MQTT 通訊機制的正規表示式

過去不少研究試圖分析或修改 MQTT，但在表述此端點互動過程中，大都直接呈現封包內容 (Happ and Wolisz, 2016)，使用流程圖 (Wu and Li, 2018) 或使用循序圖 (Sequence Diagram)(Rausch et al., 2018)。使用流程圖很難清楚呈現比較出改善的優勢，而呈現封包內容或使用循序圖又必須耗掉大量文字說明篇幅來說明其設計。我們基於之前將 REST 正規化的經驗 (Liao and Chen, 2017)，提出一個可用於描述 MQTT 的正規表示式，以期精簡呈現，並能從高層次比較與分析確保送達機制的設計原理。此表示式可做為正式規格 (Formal Specification) 使用，開發人員很容易地可以清楚明確地根據設計人所描述的規格，以程式碼加以實現[8]。以下我們先介紹此一表示式如何描述 MQTT 及其通訊行為。

首先，令一 MQTT 系統以 Tuple: (E, B, T, M) 表示，其中 B 代表系統中 MQTT Brokers 所形成的集合，理論上一個系統中可有一至多個 Brokers，邏輯上可視為一個，我們用 b 代表，其中 $b \in B$; E 代表透過 b 彼此通訊的端點 (Endpoints)，其中，E^P, E^S 分別代表訊息發佈者 (Publisher, $e^P \in E^P$) 與訊息訂閱者 (Message Subscriber, $e^S \in E^S$)，同時訂閱和發佈訊息的端點稱為 Agent，以 e^A 表示，而其宇集合以 E^A 表示。由上可知，$E = E^S \cup E^P \cup E^A$。此外，$M = \{m_1, m_2, ...\}$ 代表透過 b 傳送的訊息集合，通常訊息會針對特定的主題 (topic) 來發佈與訂閱，我們用 $T = \{t_1, t_2, ...\}$ 來代表 topic 集合。其關係如圖 2所示，在圖中所呈現的系統包含三個主題 (t_1, t_2, t_3)、二個訂閱者 (e_1^S, e_2^S)、二個發佈者 (e_1^P, e_2^P) 與二個 Agents(e_1^A, e_2^A)。

接下來定義 MQTT 訊息格式，主要包含二種形式，第一種是單純只包含標頭 (H) 和 Payload(β)，可定義如下:

$$m(H, \beta) \equiv \kappa(H, \beta)$$

同義符號 $(\equiv)$ 的左方為應用程式、API 的觀點，我們稱為「設計層次」; 右方為實際傳送的訊息，屬於實作面、網路層次的表述，我們稱為「實作層次」。由這樣的高低層次並陳表述方式，可以表示出，一個「設計層次」的行為，在「實作層次」上，是如何具體透過 MQTT 端點間訊息交換來完成。上式中 κ 代表 MQTT 定義的 Control Packet Type，例如:PUBLISH、CONNECT 等。另一種 MQTT 訊息類型則額外包含回應碼 (reason code)，主要用於明確表

[8]我們在前期計畫書詳述了如何由此表示式對應到實作程式碼的層次

述訊息解讀與執行的結果:

$$m^{'}(H,\beta) \equiv \kappa^{'}(r,H^{'},\beta^{'})$$

基於上面的定義，我們可以描述訊息傳送如何被實現。例如圖 2中，由端點 e_1^P 送訊息 m 給 Broker b，如式 (1)，接下來 b 再將此訊息轉送給 e_1^S，如式 (2)，與 e_1^A，如式 (3):

$$e_1^P.m(H,\beta) \rhd b \equiv e_1^P.\kappa(H,\beta) \rhd b \tag{1}$$

$$b.m(H,\beta) \rhd e_1^S \equiv b.\kappa(H,\beta) \rhd e_1^S \tag{2}$$

$$b.m(H,\beta) \rhd e_1^A \equiv b.\kappa(H,\beta) \rhd e_1^A \tag{3}$$

在需要回傳確認 (如 PUBACK) 的情況下，每次訊息傳送都需要二個步驟才能完成，其實作層次的定義如下:

$$\begin{aligned} &e^P.\kappa(H,\beta) \rhd b \rightarrow b.\kappa^{'}(r,H^{'},\beta^{'}) \rhd e^P \\ &b.\kappa(H,\beta) \rhd e^S \rightarrow e^S.\kappa^{'}(r,H^{'},\beta^{'}) \rhd b \end{aligned} \tag{4}$$

其中，$\rightarrow$ 為時序運算元，例如 $x \rightarrow y$ 代表 x 先發生之後 y 才發生。

3. 基本交易模型與機制設計

本節將說明對於適用於智慧環境/物聯網應用場域的訊息式交易機制的設計，因為訊息式交易機制設計較為複雜，為在呈現上更為清晰，本節先由較基本的單服務、多服務的情況開始討論。基於此基礎之上，接下來的章節呈現如何考量物聯網中所有的端點 (裝置) 都有能力負擔與交易相關的運算與儲存空間的情況，稱為「端點主控交易 (Endpoint-Managed Transaction, EMT)」。若參與交易中的部份端點 (裝置) 無法負擔交易相關的運算與儲存空間，但又需要參與交易時，就需要使用較複雜的「Broker 主控交易 (Broker-Managed Transaction, BMT)」，由 Broker 依需求建立代管節點，做為物聯網裝置的 proxy 參與交易。

3.1 單服務訊息式交易

我們先從包含二個交易端點的狀況開始，如圖 3，我們定義參與交易的端點中，要使用服務的端點 X 稱為 Client，提供服務的 Y 端點稱為 Service。X(Client) 想要存取 Y(Service) 的服務，因此傳送訊息給 Y，這個訊息的內容包含了對 Y 所需提供服務的具體指示與描述，Y 根據訊息內容加以處理後，透過 response 回傳結果 (可能是成功或失敗)，上述過程形成一個交易，交易的範圍 (Transaction Boundary) 由虛線標示，在交易的範圍內，若何一個環節失敗 (Abort)，則之前的節點必須回復原始的狀態。具體而言，X 透過 service topic 傳送服務請求訊息到 Y，接下來 Y 執行邏輯後，將結果回傳到 response topic。

交易通訊機制將透過 MQTT 5 的 request-response 來實現，response topic 每次隨著 service topic 動態建立，所以在圖中標記為 transient。整個過程中可能發生 Abort 的環節包含:(1)X 到 Broker(2)Broker 到 Y(3)Y 收到訊息後處理的過程 (4)Y 到 Broker(5)Broker 到 X(6)X 收到回傳訊息後的處理。在 MQTT 中若以 QoS-2 傳送訊息，則可確保端點到 Broker 間的訊息傳送為 exactly once，如此一來，可能會 abort 的狀況只會發生在 (3)Y 收到訊息後

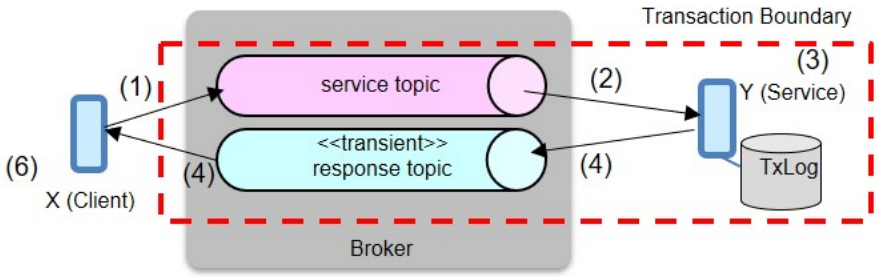

圖 3: 訊息式交易: 單一服務

處理的過程與 (6)X 收到回傳訊息後的處理[9]。其中，在 (6) 的狀況下，若 X 傳入的訊息，是去修改 Y 的狀態，則此時需要讓 Y 進行回復，稱為補償動作 (Compensation)。其主要運作機制是 Y 進行變動前，將變動儲存在本地的 transaction log(TxLog) 中，X 發出 abort 後，Y 根據 event log 中找出上次儲存的變動，加以復原。根據以上的互動描述，其正規表示式如式 (5)-(7) 所示。

$$X.m^{request}(H,\beta) \triangleright b^{Y} \rightarrow b^{Y}.m^{request}(H,\beta) \triangleright Y \tag{5}$$

$$Y.m^{respeonse}(H',\beta') \triangleright b^{Y'} \rightarrow b^{Y'}.m^{respeonse}(H',\beta') \triangleright X \tag{6}$$

$$X.m^{finish}(H'',\beta'') \triangleright b^{Y} \rightarrow b^{Y}.m(H'',\beta'') \triangleright Y \tag{7}$$

式 (5)-(7) 中 X 與 Y 的交易處理邏輯整理如虛擬碼 1與 2，在此我們採用 Cachin et al. (2011) 提出的分散式演算法虛擬碼格式。其中，虛擬碼 1在第 2、18 與 24 行檢查 tx_version 主要目的為避免 consistency 問題，在未完成目前交易前，暫不參與後續交易。虛擬碼 1支援了之前提及的二種交易方式: 2PC 和 SAGAS，Client 可以在傳來的 message header 中做設定。其中，2PC 要進行二輪動作，第一輪 (虛擬碼 1, 8-10 行) 詢問全部交易參與成員都接受並認為可成功執行 (prepare)，第二輪 (虛擬碼 1, 18-21 行) 再發出訊息確認 (commit)，若中間有任何失敗則放棄交易。SAGAS 採取比較樂觀的機制，接受 partial commit，也就是所有服務接到訊息就 commit(虛擬碼 1, 4-7 行)，若之後有任何失敗，由 Client 統一要求所有服務進行補償機制 (compensation)(虛擬碼 1, 23-25 行)。在實際寫作程式時，尚應檢查每個進來的訊息是否來自同一個 Client，這部份必須與 MQTT 本身的認證機制結合，因為篇幅有限，並在呈現上聚焦於交易處理，虛擬碼 1並未呈現此一部份。

基於上述設計，我們可以進一步處理幾個特殊議題。首先是服務是否可補償 (compensable) 的問題。目前我們假設每一個服務在交易確認後，都可以藉由補償 (compensation) 回復，但實務上並不一定，例如租用物品一旦解鎖，就可能被使用或取走。在無法回復時，有二種可能的處理方法，首先是禁止 SAGAS 的使用，如此一來會等到全交易確認完成才 commit，若採用此方法應修改虛擬碼 1的 5-7 行，遇到 SAGAS 就 abort 交易，同時也要刪除 24-25 行; 另一個可能性是仍允許 SAGAS，但在補償需求發生時，發出系統警告，人為介入處理，此時就必須將 25 行置換為警告訊息的發出。第二個問題是服務是否可重覆執行

[9]在此先排除 MQTT Broker 與端點間傳送過程出錯的可能性 (MQTT 訊息在 Broker 與端點間傳送過程，是網路層級的議題，前一次專題計畫的主題就是針對此議題予以深入討論。

```
upon message<request> m do:
  if(m.tx_id == tx_version + 1):
    try:
      if(m.tx_mode == SAGAS):
        results = prepare(m) // handle business logic, write to event log
        commit(m)          // SAGAS: commit first
        m' = createMessage(committed, results)
      elif(m.tx_mode == 2PC):
        results = prepare(m)
        m' = createMessage(prepared, results)
    catch(e):
       m' = createMessage(abort, e)
    finally:
       tx_version += 1
       send(m')

upon message<commit> m do:
  if(m.tx_id == tx_version): // 2PC Commit
    results = commit(m)
    m' = createMessage(committed, results)
    send(m')

upon message<abort> m do:
  if(m.tx_id == tx_version && tx_mode == SAGAS):
    compensate(m) // react to client abort
```

虛擬碼 1: 式 (5)-(7) 服務端點 (Y) 的虛擬碼

```
main:
  send(createMessage(request, tx_mode=SAGAS or 2PC)) // create and send tx request to the
      service
                           // tx_mode is either SAGAS or 2PC

upon message<prepared> m do:
  send(createMessage(commit))

upon message<abort> m do:
  // deal with transaction failure

upon message<committed> m do:
  send(createMessage(finish)) // or send(createMessage(abort)) if necessary
```

虛擬碼 2: 式 (5)-(7) 呼叫端點 (X) 的虛擬碼

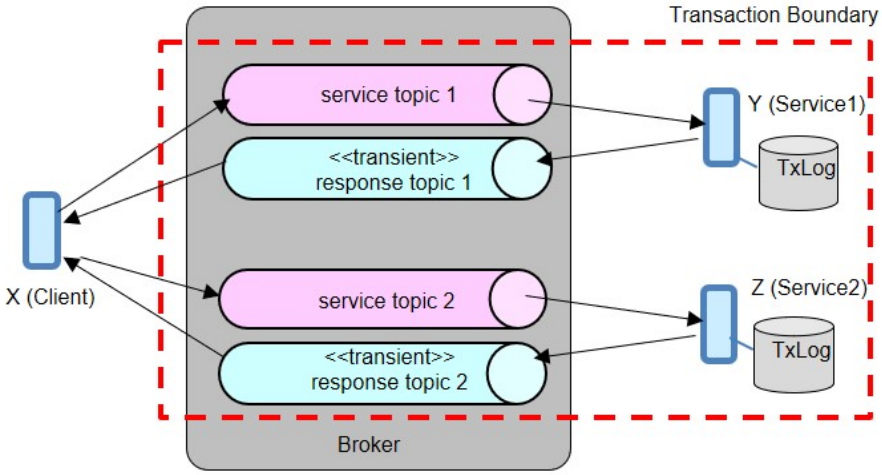

圖 4: 訊息式交易: 多服務

(idempotent)，在智慧環境中有些動作失敗時是可以重覆嘗試的，例如解鎖/上鎖，或是更新到同一個狀態。但有些動作是不可重覆執行 (non-idempotent) 的，例如環控的加溫、加濕、計數器、付費處理，這些動作在邏輯上做一次和做多次其結果是有區別的。在我們的設計上，服務若聲明是可重覆執行 (idempotent) 的動作，則可允許一定次數的重試，增加系統的強健性。在實作上，可在虛擬碼 1中的第 12 行加入 send 指令配合重試的計數器，在一定次數的嘗試後再提出 abort。

3.2 多服務訊息式交易

接下來，我們將討論擴充到包含多個服務的訊息式交易。如圖 4，在這個場景中，X(Client) 想進行的交易包含 Y(Service1) 與 Z(Service2) 二個服務，虛線部份為交易範圍，交易範圍內若何一個環節失敗 (Abort)，則之前的節點必須回復原始的狀態。單一服務交易與多服務交易最大的不同在於 Client 端的控制邏輯會較為複雜。其訊息傳送的模式為:

$$\Sigma_{s_i \in S}[X.m_{s_i}^{request}(H,\beta) \rhd b^{s_i}] \rightarrow \Sigma_{t_i \in T}[b^{s_i}.m^{t_i.request}(H,\beta)] \rhd s_i \tag{8}$$

$$s_i.m_{s_i}^{respeonse}(H',\beta') \rhd b^{s_i'} \rightarrow b^{s_i'}.m^{respeonse_{s_i}}(H',\beta') \rhd X \tag{9}$$

$$\Sigma_{s_i \in S}[X.m_{s_i}^{finish}(H,\beta) \rhd b^{s_i}] \rightarrow \Sigma_{t_i \in T}[b^{s_i}.m^{t_i.finish}(H,\beta)] \rhd s_i \tag{10}$$

其中 $\Sigma_{s_i \in S}$ 代表 $\forall i, s_i \in S$ 都做 [...] 間的行為。$s_i \in S$ 代表參與交易的服務，$t_i \in T$ 代表每個服務的子交易，b^{s_i} 代表 broker 上第 i 個服務的 service topic，$b^{s_i'}$ 代表 broker 上第 i 個服務的 response topic。虛擬碼 3列出了多交易時 X 端的處理機制，在多服務交易的情況下，需要每個交易都成功才算是成功，因此較為複雜的部份在於一對多通知的處理，及服務交易狀態的等待，也就是需要統計比對是否所有交易已完成準備，才能送出 commit。在失敗時，也需要一併通知其它相關服務 abort 的訊息。Services 的部份則和單交易相同，因此可直接沿用虛擬碼 1的設計。

```
main:
   tx = createTransaction(tx_id, tx_mode) //transaction mode is either SAGAS or 2PC
   tx.services = [request_1, request_2,...,request_n]
   for each request in services:
       send(createMessage(request, [service_topic_1, service_topic_2,...service_topic_n])

upon message<prepared> m do:
   tx.services[m.service_id].status = prepared
   if(all services.status in tx are prepared):
      send(createMessage(commit, [service_topic_1, service_topic_2,...service_topic_n]))

upon message<abort> m do:
   // deal with transaction failure
   ...
   // abort all participating transactions
   send(createMessage(abort, [service_topic_1, service_topic_2,...service_topic_n]))

upon message<committed> m do:
   send(createMessage(finish), m.service_topic)
   // or send(createMessage(abort, [service_topic_1, service_topic_2,...service_topic_n]))
       if necessary
```

虛擬碼 3: 多服務交易呼叫端點 (X) 的虛擬碼

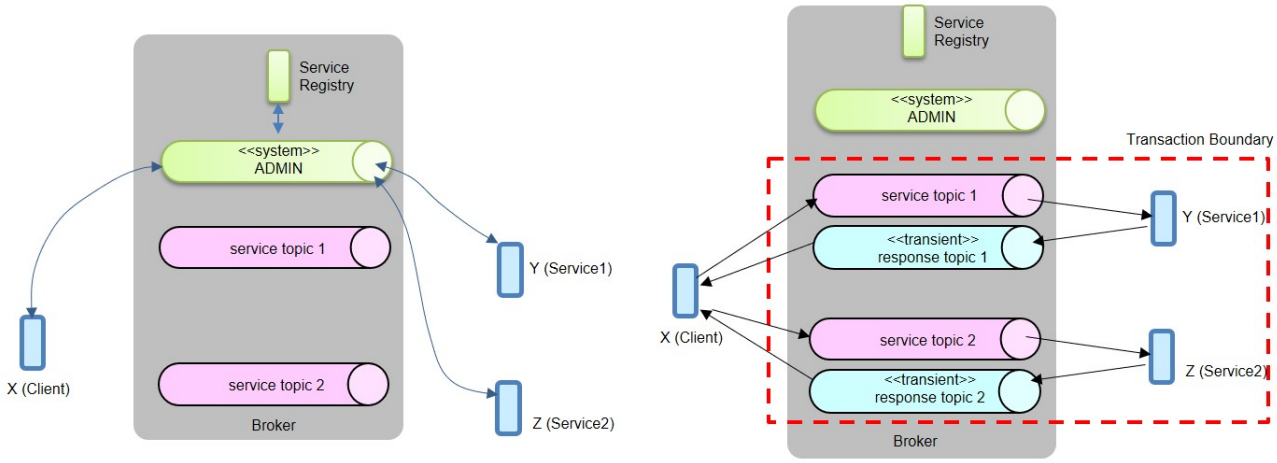

圖 5: 訊息式三階段交易，圖左:census 階段; 圖右:transaction 與 commit 階段

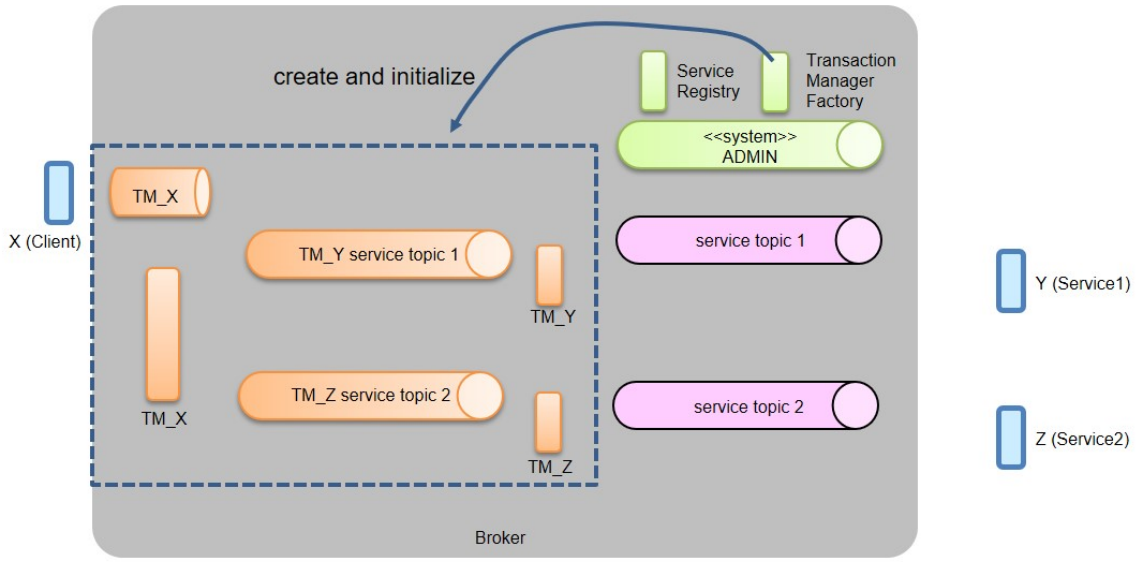

圖 6: Broker 主控式訊息交易: 交易代理人 (proxy) 與 adaptation topics 之建構

4. Endpoint-Managed Transaction (EMT) 與 Broker-Managed Transaction (BMT)

訊息式交易比較傳統交易機制最大的挑戰來自於訊息式系統 loosely coupled 的特性，造成交易參與者都不知道彼此的資訊。對交易的發起節點來說尤其困擾，因為它連有沒有人要參與交易，誰要來參與交易都無法輕易得知。由於這個問題，Vargas et al. (2007) 提出了三階段交易的構想。整個交易過程分成 census、transaction 和 commitment 三個階段。

其中 transaction 和 commitment 階段可透過 4.2 中提出的多端點交易機制來處理，這裡針對和傳統交易機制較為不同 census 階段說明。Census 階段主要用來讓交易參與成員協商誰要來參與交易，並交換必要的資訊。圖 5左描述了我們所設計在 MQTT 上實現 census 的方式。我們在 Broker 中，新增了一個專門用來進行交易協商的 topic，稱為 ADMIN(慣例上會冠以 $ 號表示系統管理專用)。首先當物聯網服務節點 Y 與 Z 加入環境後，會先到 Service Registry 進行註冊，登記包含它的 ID、名稱、idempotent 與 compensable、服務類型、service_topic 及對 EMT(Endpoint-Managed Transaction)/BMT(Broker-Managed Transaction) 的偏好。其中 EMT/BMT 是本計畫提出的概念，在我們所設計的交易機制中，開發者可以決定將「交易管理」的功能，在服務節點處理 (EMT)，或是委由 Broker 來處理 (BMT)，開發人員事先考量該節點的計算與儲存能力加以設計後，在啟動時連到 ADMIN topic 註冊。

首先，我們先考量 EMT，在 X 發起交易前，可透過 ADMIN topic 和 Service Registry 交易資訊，查詢適合參與交易的節點，並透過 ADMIN 來邀請加入交易。在圖 5左中，X 邀請了 Y 和 Z 加入交易，此時 Y 和 Z 考量自身狀態 (例如目前是否有同時參與其它交易) 接受或拒絕交易的邀請。圖 5右中，Y 和 Z 都接受了交易的邀請，接下來 X 就可以透過 4.2 中說明的多交易機制，進行交易的處理。

要支援 BMT，在 Broker 中必須新增一個稱為 TransactionManagerFactory(TMF) 的元件，用來建立代理資源不足的物聯網裝置進行交易的代理人 (proxy) 與輔助用的 topics(adaptation topics)。在 census 階段中，TransactionManagerFactory 若發現登記為 BMT

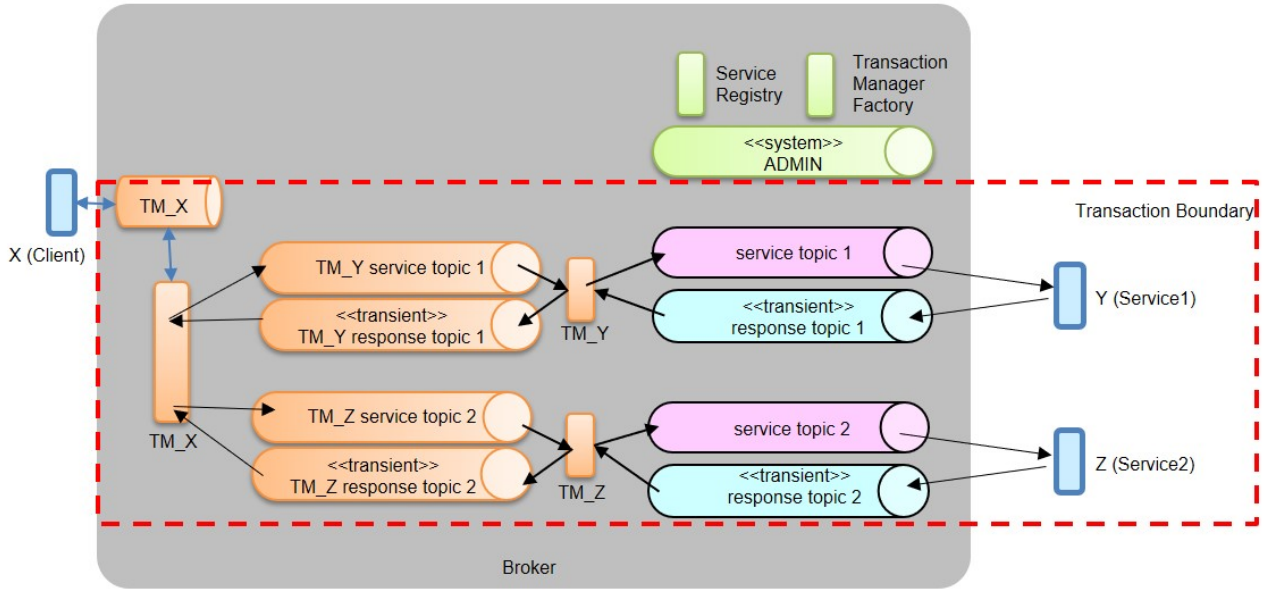

圖 7: Broker 主控式訊息交易: 透過代理人與 adaptation topics 進行交易

的服務節點接受了交易的邀請，就會啟動 BMT 的交易程序。這個程序大致上可以分成二個部份，首先是 proxy 和 adaptation topics 的建立，在圖 6中，我們假設交易發起人 (X) 與交易的參與服務 (Y 和 Z) 都使用了 BMT，此時 TMF 建立了 TM_X、TM_Y 和 TM_Z 等三個 proxy。同時，為了讓他們和原服務透通的連接，還要建立臨時輔助用的 topics: TM_X、TM_Y service topioc 1 和 TM_Z service topic 2。完成後就可進入後續的 transaction 和 commitment 階段，如圖 7所示。從圖 7可以觀察到，proxy 本質上是透過 code gen 出來的元件，它的正確運作，取決於參與節點一開始在 registry 登錄的資訊是否正確完備，這部份在實際實作時，預期應會產生更多的研究議題。此外，和圖 5比較，也可以觀察到，對交易參與者 Y 和 Z 來說，它們和 Broker 互動的方式，和 EMT 並沒有差別; 然而對交易發起者 (X) 來說，互動方式是不同的，在 EMT 中，X 必須親自處理所有交易互動，所以會直接和 service_topic 有訊息交換，但若它變成 BMT，則必須將所有交易處理邏輯委派給 TM_X。就這個角度來看交易發起者的 BMT proxy 實作將較為複雜。

5. 原型實作

為評估可行性，我們將對所設計的訊息式交易機制實現在不同端點與 Broker 上，並實際佈署到區域網路與嵌入式硬體及感測節點。以下分別說明硬體與軟體實作規畫。硬體方面，我們希望藉由計畫經費所購置工業電腦來實現 MQTT Broker 與 BMT 相關服務，以 Raspberry Pi (版本: 4 或 3 B+) 嵌入板，來實現參與交易的 Client 端點與部份 Service。而使用 Arduino WiFi (或 What's next) 實現功能精簡交易參與節點。若經費許可，我們也將建立 ZigBee/802.15.4 網路及其與 IP 網路的介接 Gateway，來模擬 LLN 與家用網路的整合。軟體方面，經由前幾年計畫的經驗，我們發現透過正規表示式，除了可以在較高層次思考規畫系統，實作時亦可以很容易根據表示式的內容，轉化為程式碼。此外，為節省實作時間，

本計畫將採用 MQTT 社群所提供的開放源碼函式庫:mqtt-packet[10]與 mqtt-connection[11]為基礎來實作。

6. 系統評估與討論

在此階段將針對研究在正確性、系統效能、相容性及安全性等面向做評估。根據驗證結果，可能回到原型實作步驟或系統設計步驟再做調整。一定的成果產生之後，再將數據與成果加以整理。以下詳述系統評估方式。

正確性評估 為了驗證系統確實完成預期目標，我們將根據 Jergler et al. (2018) 提出的方法進行 ACID 的驗證，評估設計是否符合交易要件。若發現設計上的錯誤，則針對所設計的機制加以改正並重新進行驗證，直到通過為止。此外，也將評估交易失敗處理的正確性，將列出各類不同交易失敗情境，製作為腳本來測試所發展的機制是否能成功停止所有交易並復原原始狀態。最後，也擬針對每功能項目將設計 2-3 個場景進行實作與整合性驗證。

效能評估實驗 主要目的為驗證本計畫提出的交易機制設計，其效能是否符合需求。基於上一個測試中的各類不同使用案例，探討本計畫所設計之交易機制，對整體系統效能的影響。目前初步規畫三組實驗，實驗進行時預期使用獨立的區域網路與自行建構的 LLN 來進行測試 (視經費核定情況可能會建構 1-2 個 LLN)，並將使用二台電腦 (後簡稱為 A 與 B)。其中電腦 A 規格為一般 Intel-based PC，安裝本計畫研制的 MQTT Broker，電腦 B 則將以 Raspberry Pi 3 實作，將則安裝我們實作的閘道器原型。我們將實作由 10 個節點形成的網路，分二組，第一組將進行 EMT 交易，量測交易發送率對完成時間 (turnaround time) 的影響; 第二組進行 BMT，進行相同量測，並予以比較。此外，也將根據量測數據計算二種形式的 throughput (transactions/second)。

相容性 為了避免因相容性不佳造成欠缺實用價值的後果，我們必須驗證所設計機制與 MQTT 相容性。主要進行方式為將所設計的 broker 與端點，混合其它既有 (由第三方實作) 的 MQTT 5 的端點在同一個區網中進行測試，觀察是否有因不符規範的封包丟失情形。

基本安全性評估 我們將對資訊安全四個重要特性 (資料與資源的機密性、完整性、不可否認性、身份鑑別與存取權限控制)，對系統原型做一質性分析的初步評估。目前我們初步對安全性的設計與與評估方式詳述如下:

- **機密性 (Confidentiality):** 機密性表示只有訊息的傳送目標能夠閱讀訊息內容，由於本計畫成果是基於 MQTT 實作。考量到與效能的平衡，在 MQTT 規格中，建議以 DES 或 AES 進行對稱加密。在計畫執行時，我們也將一併評估採用 TLS/SSL 時，對效能的影響。

[10]https://github.com/mqttjs/mqtt-packet
[11]https://github.com/mqttjs/mqtt-connection

- **完整性與不可否認性 (Integrity and Non-repudiation):** 安全機制中定義的完整性表示訊息傳送的過程中必須證明內容未遭到竄改或偽造；不可否認性表示使用者不可否認其所做過的事，包括送出、接收訊息，存取資料等，而透過數位簽章的方式可以達到完整性及不可否認性。在傳送訊息前，簽章與相關密文隨標頭傳送，Endponint 與閘道器生成的保密密鑰 (private key) 可用來進行解密以及簽章，若有人欲更改訊息內容，都需要進行數位簽章，否則簽名無效，即表示訊息被竄改。

- **認證與授權控制 (Authentication and Authorization):** MQTT 5 新增 AUTH control packet type 來進行驗證，因此將直接採用其設計。

7. 可能遭遇之困難及解決途徑

交易機制設計複雜度: 本研究涉及分散式交易的設計與處理，原本就較為複雜，再加上 BMT 的實作的難度較高，這部份將對機制的設計是否完備可用造成風險。此外，MQTT 最新規格 (第五版) 和 IETF 系列規格相比，文字寫作風格較抽象艱深。在計畫一年期程內要完全理解並實作規格，進而設計完備交易機制有一些挑戰性。目前規畫的解決方法是本團隊必須建立良好的知識管理機制，將規格的簡要版本以文件方式用於研究團隊成員的交接與教育訓練用途，大約了解規格要旨後，再直接閱讀規格原文，可加快理解。為快速驗證修改設計後的正確性，也需要針對重要功能，額外建立若干 Test cases，在修改後自動進行驗證，降低修正設計的額外負擔。

設計系統時程緊湊 由於系統原型實作與各項評估均需要廣泛的實作技能 (含硬體實作與通訊協定知識)，同時，研究過程將涉及 Broker 的實作或修改，具有一定門檻，在一年內完成有一定風險。為加速實作，我們採用 JavaScript 進行原型實作，近年來由於 Node.js 的普及與成熟，JavaScript 變成類似以往 Unix 系統上的 Tck/tk 的角色，可以在 C/C++ 的原生碼基礎上，寫作主要程式邏輯，因此較可在開發速度與系統效能取得平衡，且大部份嵌入系統均支援 Node.js，是不錯的選擇。Broker 實作方面，Mosca[12]是市面上唯一 JavaScript 寫成的開源 MQTT Broker，但它目前不支援 MQTT 5，因此這也是可能的困難點，初步的解決方案是直接使用其底層的 mqtt-connection[13]與 mqtt-packet[14](支援 MQTT 5) 為基礎來實作。

規格快速更新，或具有競爭性之新解決方案出現: 近年來 IoT 與智慧環境技術快速發展，相關研究與系統規格更新的速度非常快，在設計開發過程中，可能面臨到剛設計好，就有更新的解決方案出現、新的規格版本出現的問題。因此，我們在研究過程中，必須隨時留意較為相關規格與最新研究，甚至於也必須注意其它新興 IoT 規格，隨時在設計過程中因應此變化來調整研究內容。

系統安全性之不足: 近年來資訊安全議題受到很大的重視，因此所有設計必須考量基本的安全性，必須與現有的「認證」與「授權」規範配合。然「認證」與「授權」涉及資安之專

[12]Mosca: a node.js MQTT broker. https://github.com/mcollina/mosca
[13]Barebone Connection object for MQTT, https://github.com/mqttjs/mqtt-connection
[14]A library for encoding and decoding MQTT 3.1.1, 5.0 packets. https://github.com/mqttjs/mqtt-packet

業與細節設計，非敝團隊專精領域，故本計畫著重於應用層面之設計，在系統安全層面將只考慮基礎設計。如同上面所提，我們也在系統評估階段，對資料與資源的機密性、完整性、不可否認性、身份鑑別與存取權限控制等基本面向進行評估與改善。

(三) 預期完成之工作項目及成果。請分年列述：1. 預期完成之工作項目。2. 對於參與之工作人員，預期可獲之訓練。3. 預期完成之研究成果（如實務應用績效、期刊論文、研討會論文、專書、技術報告、專利或技術移轉等質與量之預期成果）。4. 學術研究、國家發展及其他應用方面預期之貢獻。

1. 預期可完成之工作項目

- MQTT 5 規格之理解與分析
- 實現基本交易機制的 MQTT Broker 建置
- 實作 BMT、EMT 交易機制，並實際使用嵌入式系統與感測板建置網路加以測試
- 實驗設計、進行、結果分析、討論及實驗報告寫作
- 期刊及會議論文之寫作與發表、期末報告之寫作及繳交

2. 對於參與之工作人員，預期可獲之訓練

- 熟悉 Messaging System 運作與 MQTT 協定
- 了解 Message Broker 的建置原理
- 理解 Transaction Processing 的理論與實務
- 分析國內外相關研究內容、特長與缺失
- 熟悉軟體系統設計研究實驗方法

3. 預期完成之研究成果（如實務應用績效、期刊論文、研討會論文、專書、技術報告、專利或技術移轉等質與量之預期成果）

- 系統原型及其程式碼 1 份
- 國際期刊或國際會議論文至少 1 篇; 國內會議論文至少 1 篇
- 碩士研究生訓練 3 人次、大學專題生 4 人次

4. 學術研究、國家發展及其他應用方面預期之貢獻。

- 近年來，物聯網商業化的需求日增，彈性且穩定的交易機制非常關鍵，本計畫基於 MQTT 上提出支援 EMT/BMT 二種模式的訊息式交易機制，可彈性因應物聯網應用的多變需求。在業界相關領域有機會成為關鍵技術 (如租用服務付款)。目前 MQTT 5 定案不久，相關研究仍少，研究成果若儘速發表，亦可望於相關領域居於較領先地位。

- 研究標的技術是進行 MQTT 物聯網服務分散交易技術開發的前提。若計畫得以順利進行，可促進 MQTT 端點間的分散式交易的研發，此類技術可望成為基於 MQTT 的物聯網服務付款機制的重要基石。
- 對於產業來說，目前較少智慧環境、IoT 等新興領域應用研究與技術開發人員，本計畫預期可針對此一方向加以探索與實作，使參與研究的成員累積更多專業知識與實務經驗，訓練更多具相關產業技術之人才。

References

Banks, A., Briggs, E., Borgendale, K., and Gupta, R. (2019). Mqtt version 5. *OASIS standard*.

Banks, A. and Gupta, R. (2014). Mqtt version 3.1.1. *OASIS standard*, 29:89.

Banno, R., Sun, J., Fujita, M., Takeuchi, S., and Shudo, K. (2017). Dissemination of edge-heavy data on heterogeneous mqtt brokers. In *Cloud Networking (CloudNet), 2017 IEEE 6th International Conference on*, pages 1–7. IEEE.

Bernstein, P. and Newcomer, E. (2009). *Principles of Transaction Processing*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science.

Cachin, C., Guerraoui, R., and Rodrigues, L. (2011). *Introduction to Reliable and Secure Distributed Programming*. Springer Berlin Heidelberg.

Dikii, D. I. (2020). Remote access control model for mqtt protocol. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 288–291. IEEE.

Garcia-Molina, H. and Salem, K. (1987). Sagas. *ACM Sigmod Record*, 16(3):249–259.

Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112.

Grimm, R. (2004). One. world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, 3(3):22–30.

Happ, D. and Wolisz, A. (2016). Limitations of the pub/sub pattern for cloud based iot and their implications. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6. IEEE.

Hohpe, G. and Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional.

Iglesias-Urkia, M., Orive, A., Barcelo, M., Moran, A., Bilbao, J., and Urbieta, A. (2017). Towards a lightweight protocol for industry 4.0: An implementation based benchmark. In

Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017 IEEE International Workshop of, pages 1–6. IEEE.

Jergler, M., Zhang, K., and Jacobsen, H.-A. (2018). Multi-client transactions in distributed publish/subscribe systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 120–131. IEEE.

Jo, H.-C. and Jin, H.-W. (2015). Adaptive periodic communication over mqtt for large-scale cyber-physical systems. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2015 IEEE 3rd International Conference on*, pages 66–69. IEEE.

Kawaguchi, R. and Bandai, M. (2020). Edge based mqtt broker architecture for geographical iot applications. In *2020 International Conference on Information Networking (ICOIN)*, pages 232–235. IEEE.

Kim, S.-M., Choi, H.-S., and Rhee, W.-S. (2015). Iot home gateway for auto-configuration and management of mqtt devices. In *Wireless Sensors (ICWiSe), 2015 IEEE Conference on*, pages 12–17. IEEE.

Kodali, R. K. (2016). An implementation of mqtt using cc3200. In *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2016 International Conference on*, pages 582–587. IEEE.

Lampson, B. W. (1981). Atomic transactions. In *Distributed Systems - Architecture and Implementation, An Advanced Course*, page 246–265, Berlin, Heidelberg. Springer-Verlag.

Liao, C.-F. and Chen, K. (2019). A service platform for streamlining the production of cyber–physical interactive performance art. *Service Oriented Computing and Applications*, 13(3):221–236.

Liao, C.-F. and Chen, P.-Y. (2017). Rosa: Resource-oriented service management schemes for web of things in a smart home. *Sensors*, 17(10):2159.

Liao, C.-F. and Fu, H.-Y. (2018). Spatial-aware service management in a pervasive environment. *Service Oriented Computing and Applications*, 12(2):95–110.

Liebig, C. and Tai, S. (2001). Middleware mediated transactions. In *Proceedings 3rd International Symposium on Distributed Objects and Applications*, pages 340–350. IEEE.

Longo, E., Redondi, A. E., Cesana, M., Arcia-Moret, A., and Manzoni, P. (2020). Mqtt-st: A spanning tree protocol for distributed mqtt brokers. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

Oweis, N. E., Aracenay, C., George, W., Oweis, M., Soori, H., and Snasel, V. (2016). Internet of things: Overview, sources, applications and challenges. In *Proceedings of the Second International Afro-European Conference for Industrial Advancement AECIA 2015*, pages 57–67. Springer.

Pietzuch, P. R. (2004). Hermes: A scalable event-based middleware. Technical report, University of Cambridge, Computer Laboratory.

Pipatsakulroj, W., Visoottiviseth, V., and Takano, R. (2017). mumq: A lightweight and scalable mqtt broker. In *Local and Metropolitan Area Networks (LANMAN), 2017 IEEE International Symposium on*, pages 1–6. IEEE.

Rausch, T., Nastic, S., and Dustdar, S. (2018). Emma: Distributed qos-aware mqtt middleware for edge computing applications. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 191–197. IEEE.

Saif, U. and Greaves, D. J. (2001). Communication primitives for ubiquitous systems or rpc considered harmful. In *Distributed Computing Systems Workshop, 2001 International Conference on*, pages 240–245. IEEE.

Shatsky, Y., Gudes, E., and Gudes, E. (2008). Tops: A new design for transactions in publish/subscribe middleware. In *Proceedings of the Second International Conference on Distributed Event-Based Systems*, DEBS '08, page 201–210, New York, NY, USA.

Tai, S. and Rouvellou, I. (2000). Strategies for integrating messaging and distributed object transactions. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 308–330. Springer.

Tantitharanukul, N., Osathanunkul, K., Hantrakul, K., Pramokchon, P., and Khoenkaw, P. (2017). Mqtt-topics management system for sharing of open data. In *Digital Arts, Media and Technology (ICDAMT), International Conference on*, pages 62–65. IEEE.

Vargas, L., Pesonen, L. I., Gudes, E., and Bacon, J. (2007). Transactions in content-based publish/subscribe middleware. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pages 68–68. IEEE.

Wu, X. and Li, N. (2018). Improvements of mqtt retain message storage mechanism. In *2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 957–961. IEEE.

Zare, A. and Iqbal, M. T. (2020). Low-cost esp32, raspberry pi, node-red, and mqtt protocol based scada system. In *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pages 1–5. IEEE.