A Formal Model and Analysis of the MQ Telemetry Transport Protocol

Benjamin Aziz
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

***Abstract*—We present a formal model of the MQ Telemetry Transport version 3.1 protocol based on a timed message-passing process algebra. We explain the modeling choices that we made, including pointing out ambiguities in the original protocol specification, and we carry out a static analysis of the formal protocol model, which is based on an approximation of a name-substitution semantics for algebra. The analysis reveals that the protocol behaves correctly as specified against the first two quality of service modes of operation providing at most once and at least once delivery semantics to the subscribers. However, we find that the third and highest quality of service semantics is prone to error and at best ambiguous in certain aspects of its specification. Finally, we suggest an enhancement of this level of QoS for the protocol.**

***Keywords*-Embedded Systems, Formal Verification, IoT, MQTT, Protocols**

I. Introduction

The MQ Telemetry Transport (MQTT) protocol - version 3.1 [1] is described by OASIS as a lightweight broker-based publish/subscribe messaging protocol that was designed to allow devices with small processing power and storage, such as those which the Internet of Things (IoT) is composed of, to communicate over low-bandwidth and unreliable networks. The publish/subscribe message pattern [2], on which MQTT is based, provides for one-to-many message distribution with three variaties of delivery semantics, based on the level of quality of service expected from the protocol, including “at most once”, “at least once” and “exactly once” semantics.

The protocol defines the message structure needed in communications between *clients*, i.e. end-devices responsible for generating data from their domain (the data source) and *servers*, which are the system components responsible for collating source data from clients/end-devices and distributing these data to interested subscribers.

In the “at most once” case, messages are delivered with the best effort of the underlying communication infrastructure, which is usually IP-based, therefore there is no guarantee that the message will arrive. This protocol, which is termed as the QoS = 0 protocol, is represented by the following flow of messages and actions:

Client → Server	:	**Publish**
Server Action	:	*Publish message to subscribers*

In the second case of “at least once” semantics, certain mechanisms are incorporated to allow for message duplication, and despite the guarantee of delivering the message, there is no guarantee that duplicates will be suppressed. This case is represented by the following flow of messages and actions:

Client → Server	:	**Publish**
Client Action	:	*Store Message*
Server Actions	:	*Store Message*,
		Publish message to subscribers,
		Delete Message
Server → Client	:	**Puback**
Client Action	:	*Discard Message*

The second message **Puback** represents an acknowledgment of the receipt of the first message, and if **Puback** is lost, then the first message is retransmitted by the client (hence the reason why the message is stored at the client). Once the protocol completes, the client discards the message. This protocol is also known as the QoS = 1 protocol.

Finally, for the last case of “exactly once” delivery semantics, also known as the QoS = 2 protocol, the published message is guaranteed to arrive only once at the subscribers. This is represented by the following flow of messages and actions:

Client → Server	:	**Publish**
Client Action	:	*Store Message*
Server Actions	:	*Store Message OR*
		Store Message ID,
		Publish message to subscribers
Server → Client	:	**Pubrec**
Client → Server	:	**Pubrel**
Server Actions	:	*Publish message to subscribers*,
		Delete Message OR
		Delete Message ID
Server → Client	:	**Pubcomp**
Client Action	:	*Discard Message*

In this protocol, **Pubrec** and **Pubcomp** represent acknowledgment messages from the server, whereas **Pubrel** is an acknowledgment message from the client. The loss of **Pubrec** causes the client to recommence the protocol from its beginning, whereas the loss of **Pubcomp** causes the client to retransmit only the second part of the protocol, which starts at the **Pubrel** message. This additional machinery presumably ensures a single delivery of the published message to the subscribers.

The rest of the paper is organised as follows. In Section II, we provide an overview of the TPi process algebra, a timed version of the π-calculus [3] and provide a concrete semantics for the language. In Section III, we abstract this semantics and define our static analysis algorithm. In Section, IV we develop a model of the MQTT protocol based on TPi, and explain the various modeling options that we adopted. In Section V, we give an analysis of the protocol in the context of its three versions of the delivery semantics and we discuss the results of the analysis in Section VI. Finally, in Section VII, we describe related work in current literature and in Section VIII, we conclude the paper and provide directions for future research.

II. TPi: A timed Process Algebra

The model of MQTT that we introduce here is based on a process algebra called TPi, originally inspired by [4] and further developed in [5], which is a synchronous message-passing calculus capable of expressing timed inputs.

A. Syntax and Structural Operational Semantics of TPi

The syntax of the language defines processes, $P, Q \in \mathcal{P}$, based on names $x, y \in \mathcal{N}$ as follows:

$$\begin{aligned} P, Q ::= {} & \overline{x}\langle y \rangle.P \mid \mathtt{timer}^t(x(y).P, Q) \mid\ !P \ \mid\ (\nu x)P \ \mid \\ & (P|Q) \ \mid\ (P{+}Q) \ \mid\ \mathbf{0} \ \mid\ A(x) \end{aligned}$$

The syntax corresponds to that of the standard synchronous π-calculus [3] except for the fact that input actions are placed within a timer, $\mathtt{timer}^t(x(y).P, Q)$, where $t \in \mathbb{N}$ represents a time bound. The input action, $x(y).P$, can synchronise with suitable output actions as long as $t > 0$. Otherwise, when $t = 0$, the timer behaves as Q. There is an assumption that t is decremented by the environment of the process and that t can be any time unit (e.g. tick, second etc.). Finally, we utilised *process definition calls* in the form of $A(x)$. This calls a process definition $A(y) \overset{\mathrm{def}}{=} P$, and at the same time, passes it the value x to replace y. If the definition A does not accept any input parameters, then we simply omit the input parameter y and write $A() \overset{\mathrm{def}}{=} P$.

The structural operational semantics of TPi are given in terms of the structural congruence, $\equiv$, and labeled transition, $\xrightarrow{\mu}$, relations as shown in Figure 1, where *fn*(P) represents the set of free names of P.

The definition of $\equiv$ is standard, except for rules (6), (7), (8) and (9), which deal with expired and infinite timers, and parameterised/non-parameterised process definition calls, respectively. The labels, $\mu \in \{\xrightarrow{\overline{x}\langle y \rangle}, \xrightarrow{\overline{x}(y)}, \xrightarrow{x(z)}, \xrightarrow{\tau}\}$, express free and bound outputs, inputs and silent actions. Again, most of the rules for $\xrightarrow{\mu}$ are straightforward and their explanation can be found elsewhere (e.g. [6, §3.2.2] and [5]) except for rule (19), where a *time-stepping* function, $\eth : \mathcal{P} \to \mathcal{P}$, expresses the ticking of activated timers (i.e. timed inputs that are already at the head of the process waiting to accept a message) by the external environment of the process:

$$\eth(P) = \begin{cases} \mathtt{timer}^t(x(y).Q, Q'), & \text{if} \\ \quad P = \mathtt{timer}^{t+1}(x(y).Q, Q') & \\ \quad \text{and } 0 < t+1 < \infty & \\ \eth(Q) \mid \eth(R), & \text{if } P = Q \mid R \\ \eth(Q) + \eth(R), & \text{if } P = Q + R \\ (\nu x)\eth(Q), & \text{if } P = (\nu x)Q \\ \eth(Q[x/y]), & \text{if } P = A(x) \text{ and} \\ & A(y) \overset{\mathrm{def}}{=} Q \\ \eth(Q), & \text{if } P = A() \text{ and} \\ & A() \overset{\mathrm{def}}{=} Q \\ P, & \text{otherwise} \end{cases}$$

This function reduces the timer's value as long as that value is still a positive number, expressed as $t{+}1$. The function distributes over parallel composition, non-deterministic choice, restrictions and process definition calls, but it has no effect over all other processes. This is interesting for the cases of output and replicated processes since these are considered to be non-active unless they are synchronised or replicated. We assume that rule (19) is applied once every unit of time and that its application (including the call to the $\eth$ function) is completed before a single unit of time elapses.

B. A Name-Substitution Semantics

In this section, we define a non-standard semantics for *TPi* such that it is possible to express the meaning of processes in terms of name substitutions resulting from message passing (note here that we exclude other substitutions, such as those due to α-conversions or renaming of bound names). For example, in:

$$!((\nu y)\overline{x}\langle y \rangle.\mathbf{0}) \mid\ !\mathtt{timer}^{t+1}(x(u).\mathbf{0}, \mathbf{0})$$

we would like to have a meaning that captures the set of substitutions, $\{y_1/u_1, y_2/u_2 \ldots\}$, where y_i is a labeled copy of the fresh name, y, and u_i is a labeled instance of the input parameter, u, assuming that $t + 1 > 0$. (Note: other labeling schemes are also possible as long as they maintain bound name uniqueness).

First however, we need to introduce the notion of *tags* defined as the set, $\ell, \ell' \in \mathcal{L}$. The set $\mathcal{L}$ is then used to tag

Rules of the $\equiv$ relation:

(1) $(\mathcal{P}/\equiv, |, \mathbf{0})$ is a commutative monoid
(2) $(\nu x)\mathbf{0} \equiv \mathbf{0}$
(3) $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
(4) $!P \equiv P\ |!P$
(5) $(\nu x)(P \mid Q) \equiv (P \mid (\nu x)Q)$ if $x \notin \mathit{fn}(Q)$
(6) $\mathtt{timer}^{0}(x(z).P, Q) \equiv Q$
(7) $\mathtt{timer}^{\infty}(x(z).P, Q) \equiv x(z).P$
(8) $A(x) \equiv P[x/y]$, where $A(y) \overset{\mathtt{def}}{=} P$
(9) $A() \equiv P$, where $A() \overset{\mathtt{def}}{=} P$

Rules of the $\xrightarrow{\mu}$ relation:

(10) $\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}\langle y\rangle} P$
(11) $\mathtt{timer}^{t+1}(x(z).P, Q) \xrightarrow{x(z)} P$
(12) $P \xrightarrow{\overline{x}\langle y\rangle} Q \;\;\Rightarrow\;\; (\nu y)P \xrightarrow{\overline{x}(y)} Q$ if $x \neq y$
(13) $P \xrightarrow{\overline{x}\langle y\rangle} P', Q \xrightarrow{x(z)} Q' \;\;\Rightarrow$
$P \mid Q \xrightarrow{\tau} P' \mid Q'[y/z]$
(14) $P \xrightarrow{\overline{x}(y)} P', Q \xrightarrow{x(z)} Q' \;\;\Rightarrow$
$P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q'[y/z])$
(15) $P \xrightarrow{\mu} Q \;\;\Rightarrow\;\; (\nu x)P \xrightarrow{\mu} (\nu x)Q$ if $x \neq \mathit{fn}(\mu)$
(16) $P \xrightarrow{\mu} P' \;\;\Rightarrow\;\; P \mid Q \xrightarrow{\mu} P' \mid Q$
(17) $P \xrightarrow{\mu} P' \;\;\Rightarrow\;\; P + Q \xrightarrow{\mu} P'$
(18) $P \xrightarrow{\mu} P' \;\;\Rightarrow\;\; Q + P \xrightarrow{\mu} P'$
(19) $P \xrightarrow{\tau} \eth(P)$

Figure 1. The structural operational semantics of *TPi* [5].

messages of output actions: $\overline{x}\langle y\rangle.P$ becomes $\overline{x}\langle y^{\ell}\rangle.P$. This tagging is performed uniquely, i.e. no two messages will be assigned the same tag even if the two messages have the same name. This will help distinguish every message in the non-standard interpretation. Additionally, we define the following two functions involving tags:

$\mathit{value\_of} \;:\; \mathcal{L} \rightarrow \mathcal{N}$
$\mathit{tags\_of} \;:\; \mathcal{P} \rightarrow \wp(\mathcal{L})$

where $\mathit{value\_of}(\ell) = y$ signifies that ℓ was assigned to the message y and $\mathit{tags\_of}(P) = \{\ell 1, \ldots, \ell n\}$ signifies the set of tags used in P. Naturally, *value_of* is non-injective and we sometimes write $\mathit{value\_of}(\{\ell, \ell' \ldots\})$ to mean $\{\mathit{value\_of}(\ell), \mathit{value\_of}(\ell') \ldots\}$.

Next, we define the environment, $\phi_{\mathcal{S}} : \mathcal{N} \rightarrow \wp(\mathcal{L})$, such that $\ell \in \phi_{\mathcal{S}}(x)$ implies that the message tagged with ℓ replaces the input parameter, x, at runtime. From $\phi_{\mathcal{S}}$, a semantic domain, $D_{\perp} : \mathcal{N} \rightarrow \wp(\mathcal{L})$, is formed with the following ordering:

$$\forall \phi_{\mathcal{S}1}, \phi_{\mathcal{S}2} \in D_{\perp} \;:\; \phi_{\mathcal{S}1} \sqsubseteq_{D_{\perp}} \phi_{\mathcal{S}2} \;\Leftrightarrow\; \forall x \in \mathcal{N} :\; \phi_{\mathcal{S}1}(x) \subseteq \phi_{\mathcal{S}2}(x)$$

where the bottom element, $\perp$, denotes the null environment, $\phi_{\mathcal{S}0}$, which maps every name in $\mathcal{N}$ to $\emptyset$. From the above definition of $D_{\perp}$ then, we can assign a meaning to process P as a function $\mathcal{S}(\![P]\!)\ \rho\ \phi_{\mathcal{S}} \in D_{\perp}$, defined over the structure of P as shown in Figure 2.

In the rules of this semantics, ρ is a multiset of processes in parallel with the interpreted process along with the standard $\{\!|\, - \,|\!\} : \mathcal{P} \rightarrow \wp(\mathcal{P})$ and $\uplus : \wp(\mathcal{P}) \times \wp(\mathcal{P}) \rightarrow \wp(\mathcal{P})$ operators over ρ. The meaning of ρ is given in $(\mathcal{R}0)$ using the special union, $\cup_{\phi_{\mathcal{S}}}$, defined as:

$$\forall x \in \mathcal{N} :\; (\phi_{\mathcal{S}1} \cup_{\phi_{\mathcal{S}}} \phi_{\mathcal{S}2})(x) = \phi_{\mathcal{S}1}(x) \cup \phi_{\mathcal{S}2}(x)$$

We describe next these rules informally. Rule $(\mathcal{S}1)$ does not affect the value of $\phi_{\mathcal{S}}$ since communications are dealt with in the case of input actions (the next two rules). In $(\mathcal{S}2)$ for active input actions, the rule uses the equivalence of two names, $\overset{\phi_{\mathcal{S}}}{\sim}$, parameterised by $\phi_{\mathcal{S}}$ to determine matching channel names. This is defined for any two names, x and y as:

$$x \overset{\phi_{\mathcal{S}}}{\sim} y \;\;\Leftrightarrow$$
$$(\mathit{value\_of}(\phi_{\mathcal{S}}(x)) \cap \mathit{value\_of}(\phi_{\mathcal{S}}(y)) \neq \emptyset) \;\vee\; (x = y)$$

For each synchronisation, the value of $\phi_{\mathcal{S}}$ is updated with the tag of the communicated message using the *update* operator defined as:

$$\forall \phi_{\mathcal{S}} \in D_{\perp}, y \in \mathcal{N}, \ell \in \mathcal{L} :$$
$$\mathit{update}(\phi_{\mathcal{S}}, y, \ell) = \phi_{\mathcal{S}}[y \mapsto \phi_{\mathcal{S}}(y) \cup \{\ell\}]$$

Rule $(\mathcal{S}2)$ also considers the case where no communications take place. In either case, all active timers are decremented some using the time-stepping defined in the previous section. Rule $(\mathcal{S}3)$ deals with inactive input actions the meaning of the input-guarded process becomes the meaning of the continuation process Q. Rule $(\mathcal{S}4)$ naturally adds two parallel process in the ρ multiset. Rule $(\mathcal{S}5)$ deals with replicated processes using a fixed-point calculation of the higher order functional, $\mathcal{F}$. The rule allows for as many copies of P to be spawned and the number of each copy is used to subscript its bound names and tags in order to maintain their uniqueness. As a result, the interpretation of restricted names in rule $(\mathcal{S}6)$ drops the ν operator in ρ. Rule $(\mathcal{S}7)$ returns the same value of $\phi_{\mathcal{S}}$ when interpreting the null process, and in rule $(\mathcal{S}8)$,

$$
\begin{array}{lll}
(\mathcal{S}1) & \mathcal{S}\llbracket \overline{x}\langle y^{\ell}\rangle.P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \phi_{\mathcal{S}} \\
(\mathcal{S}2) & \mathcal{S}\llbracket \mathtt{timer}^{t+1}(x(y).P,Q) \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ (\displaystyle\bigcup_{\overline{x'}\langle z^{\ell}\rangle.P'\in\rho:\ x \overset{\phi_{\mathcal{S}}}{\sim} x'}{}_{\phi_{\mathcal{S}}} \ \mathcal{R}\llbracket (\biguplus_{R\in\rho} \{\!|\eth(R)|\!\}) \uplus \{\!|P|\!\} \uplus \{\!|P'|\!\} \rrbracket\ \mathit{update}(\phi_{\mathcal{S}},y,\ell)) \cup_{\phi_{\mathcal{S}}} \\
 & & \quad \mathcal{R}\llbracket (\displaystyle\biguplus_{R\in\rho} \{\!|\eth(R)|\!\}) \uplus \{\!|\mathtt{timer}^{t}(x(y).P,Q)|\!\} \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}3) & \mathcal{S}\llbracket \mathtt{timer}^{0}(x(y).P,Q) \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{R}\llbracket \{\!|Q|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}4) & \mathcal{S}\llbracket P \mid Q \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{R}\llbracket \{\!|P|\!\} \uplus \{\!|Q|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}5) & \mathcal{S}\llbracket !P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \mathit{snd}(\mathit{fix}\ \mathcal{F}(0,\perp)) \\
 & \multicolumn{2}{l}{\text{where, } \mathcal{F} = \lambda f \lambda(j,\phi).f\ \ (\mathit{if}\,\phi = \mathcal{R}\llbracket (\displaystyle\biguplus_{i=0}^{j} \{\!|(P)\sigma|\!\})\ \uplus\ \rho \rrbracket\ \phi_{\mathcal{S}}\ \mathit{then}\ j,\phi\ \mathit{else}\ (j+1),(\mathcal{R}\llbracket (\biguplus_{i=0}^{j} \{\!|(P)\sigma|\!\})\ \uplus\ \rho \rrbracket\ \phi_{\mathcal{S}}))} \\
 & \multicolumn{2}{l}{\text{and } \sigma = [\mathit{bn}_i(P)/\mathit{bn}(P)][\mathit{tags\_of}_i(P)/\mathit{tags\_of}(P)],\ \mathit{bn}_i(P) = \{x_i \mid x \in \mathit{bn}(P)\},\ \mathit{tags\_of}_i(P) = \{\ell_i \mid \ell \in \mathit{tags\_of}(P)\}} \\
(\mathcal{S}6) & \mathcal{S}\llbracket (\nu n)P \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{R}\llbracket \{\!|P|\!\} \uplus \rho \rrbracket\ \phi_{\mathcal{S}} \\
(\mathcal{S}7) & \mathcal{S}\llbracket \mathbf{0} \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \phi_{\mathcal{S}} \\
(\mathcal{S}8) & \mathcal{S}\llbracket P + Q \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{S}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{S}}\ \cup\ \mathcal{S}\llbracket Q \rrbracket\ \rho\ \phi_{\mathcal{S}} \\
(\mathcal{S}9) & \mathcal{S}\llbracket A(x) \rrbracket\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{S}\llbracket P[x/y] \rrbracket\ \rho\ \phi_{\mathcal{S}} \text{ where, } A(y) \overset{\mathtt{def}}{=} P \\
(\mathcal{R}0) & \mathcal{R}\llbracket \rho \rrbracket\ \phi_{\mathcal{S}} & = \ \displaystyle\bigcup_{P\in\rho}{}_{\phi_{\mathcal{S}}} \mathcal{S}\llbracket P \rrbracket\ (\rho \backslash \{\!|P|\!\})\ \phi_{\mathcal{S}}
\end{array}
$$

Figure 2. The definition of $\mathcal{S}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{S}}$.

applies the union of the two possible choice interpretations. Finally, rule ($\mathcal{S}9$) replaces a process application with its definition making the necessary name substitution. Note here that we do not capture these substitutions in $\phi_{\mathcal{S}}$ the same way we do with communicated messages, as we are more interested in the latter for the purpose of our analysis.

The following soundness theorem states that name substitutions in the structural operational semantics are captured in the non-standard semantics.

Theorem 1 (Soundness of the non-standard semantics). $\forall P, Q, x, y\ :\ P \xrightarrow{\mu}{}^{*} Q[x/y] \ \Rightarrow\ x \in \mathit{value\_of}(\phi'_{\mathcal{S}}(y))$ *where,* $\phi'_{\mathcal{S}} = \mathcal{S}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{S}}$

Proof: The proof is by induction on the rules of the structural operational semantics in Figure 1. The most interesting cases are rules (10) and (11), where we need to show that if a process, P, exhibits a transition, $P \xrightarrow{\overline{x}\langle y\rangle} P'$, then this will eventually yield a process, $\overline{x}\langle y^{\ell}\rangle.P'' \in \rho$ during the non-standard interpretation. The same can be shown for $Q \xrightarrow{x(z)} Q'$ and $R \xrightarrow{\overline{x}(y)} R'$. From rule ($\mathcal{S}2$), we can then show that $P \mid Q$ and $R \mid Q$ will capture substitutions in the $\phi_{\mathcal{S}}$ environment in each case. ■

III. An Approximated Semantics

The computation of the non-standard semantics of the previous section is not guaranteed to terminate due to the infinite size of $D_{\perp}$ as a result of the presence of replication in processes. Therefore, we need to approximate the meaning of processes by introducing the α_k approximation, which limits the number of copies of fresh names and tags that can be captured by the semantics.

Definition 1 (The α_k-approximation function). *Define* $\alpha_k\ :\ (\mathcal{N} \cup \mathcal{L}) \rightarrow (\mathcal{N}^{\sharp} \cup \mathcal{L}^{\sharp})$ *as follows, where* $\mathcal{N}^{\sharp} = \mathcal{N} \backslash \{x_i \mid i > k\}$ *and* $\mathcal{L}^{\sharp} = \mathcal{L} \backslash \{\ell_i \mid \ell > k\}$*:*

$$
\forall u \in (\mathcal{N} \cup \mathcal{L}) : \alpha_k(u) = \begin{cases} u_k, & \mathit{if}\ u = u_i\ \wedge\ i > k \\ u, & \mathit{otherwise} \end{cases}
$$

□

We write, $\alpha_k(\{u, u', \ldots\})$, to mean $\{\alpha_k(u), \alpha_k(u'), \ldots\}$. The α_k approximation function leads naturally to the appearance of the abstract environment, $\phi_{\mathcal{A}} : \mathcal{N}^{\sharp} \rightarrow \wp(\mathcal{L}^{\sharp})$ and the abstract semantic domain, $D^{\sharp}_{\perp}$ with the following ordering:

$$
\forall \phi_{\mathcal{A}1}, \phi_{\mathcal{A}2} \in D^{\sharp}_{\perp}\ :\ \phi_{\mathcal{A}1} \sqsubseteq_{D^{\sharp}_{\perp}} \phi_{\mathcal{A}2} \quad \Leftrightarrow \quad \forall x \in \mathcal{N}^{\sharp}\ :\ \phi_{\mathcal{A}1}(x) \subseteq \phi_{\mathcal{A}2}(x)
$$

Based on $D^{\sharp}_{\perp}$, we can interpret processes as a new function, $\mathcal{A}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{A}} \in D^{\sharp}_{\perp}$, defined as follows:

$$
\mathcal{A}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{A}} = \mathtt{let}\ \mathit{update} = \mathit{update}^{\mathcal{A}}_{\alpha_k}\ \mathtt{in}\ \mathtt{let}\ \phi_{\mathcal{S}} = \phi_{\mathcal{A}}\ \mathtt{in}\ \mathcal{S}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{S}}
$$

which uses the same algorithm for $\mathcal{S}\llbracket P \rrbracket\ \rho\ \phi_{\mathcal{S}}$ defined in Figure 2 but replacing $\phi_{\mathcal{S}}$ and *update* with their abstract siblings. The $\mathit{update}^{\mathcal{A}}_{\alpha_k}$ operator is defined for all $\phi_{\mathcal{A}} \in D^{\sharp}_{\perp}, y \in \mathcal{N}, \ell \in \mathcal{L}$ as follows:

$$
\mathit{update}^{\mathcal{A}}_{\alpha_k}(\phi_{\mathcal{A}}, y, \ell) = \phi_{\mathcal{A}}[\alpha_k(y) \mapsto \phi_{\mathcal{A}}(\alpha_k(y)) \cup \{\alpha_k(\ell)\}]
$$

The following termination result can be shown to hold.

Theorem 2 (Termination of the Abstract Semantics). *For any process,* P*, the computation of* $\mathcal{A}\llbracket P \rrbracket\ \{\!|\,|\!\}\ \perp_{D^{\sharp}_{\perp}}$ *terminates.*

Proof: The proof relies on two requirements: First, to show that $D^{\sharp}_{\perp}$ is finite. This is true from the definition of α_k. The second is to show that the abstract meaning of a process is monotonic with respect to the number of copies of a replicated process:

$$\mathcal{R}(\![(\biguplus_{i=0}^{j} \{\!|(P)\sigma|\!\}) \uplus \rho]\!)\ \phi_{\mathcal{A}} \sqsubseteq_{D^{\sharp}_{\perp}} \mathcal{R}(\![(\biguplus_{i=0}^{j+1} \{\!|(P)\sigma|\!\}) \uplus \rho]\!)\ \phi_{\mathcal{A}}$$

This latter requirement is proved by showing that the extra copy of P can "only" induce more communications. ■

IV. A Model of MQTTv3.1

We now define a model of the MQTT protocol in TPi as shown in Figure 3, which captures the client/server protocol messages. Although the protocol also describes messages between the server and the subscribers, we only focus on one aspect of these, which is the initial publish message from the server to the subscribers. The model expresses three protocols, one for each of the three levels of the quality of service specified in [1].

A. The Subscribers

Our model of the subscribers is minimal, since we only care about the first step in their behaviour, which is listening to the published messages announced by the server:

$$\mathit{Subscriber}() \stackrel{\text{def}}{=} \quad !\mathit{pub}(x')$$

This definition does not care about what happens to the message after it has been read by the subscriber on the channel *pub*. The main reason for including the replication, !, is to allow for the possibility of accepting multiple messages from the server. This will allow us later in the analysis to validate the different delivery semantics associated with the MQTTv3.1 protocol. The definition can capture the number of times the subscriber will read a message within a single run of the protocol since each instance of x' spawned under the replication is renamed with the labeling system x'_1, x'_2, etc. [7]. The definition also assumes that the subscriber can wait ad infinitum for a message to be published by the server, and if no such message is published, it will do nothing. This is not realistic, but sufficient for our purpose here.

B. The Attacker

The attacker in our case has a very primitive role, which reflects the passive behaviour of an open network that offers the possibility of consuming the exchanged messages in the protocol. In fact, the attacker is only interested in disrupting messages between the client and the server. Therefore, its definition is to listen continuously on the channels c and c' over which the client and the server communicate:

$$\mathit{Attacker}() \stackrel{\text{def}}{=} \quad !(c(y') + c'(u'))$$

Similar to the case of the subscribers, the attacker is not in a rush to obtain an input from the protocol, therefore it can wait ad infinitum for a message to be received on its channels c or c'. It is also possible to run a finite attacker model as follows:

$$\mathit{Attacker}() \stackrel{\text{def}}{=} \quad (c(y'_1) + c'(u'_1)) \mid \ldots \mid (c(y'_n) + c'(u'_n))$$

Where the operator ! is replaced by a finite number n of the input choices all composed in parallel. In this finite model, the attacker is capable of only consuming n messages.

V. Analysis of the Protocol

We now define formally the three message-delivery semantics associated with the MQTT protocol, *at most once*, *at least once* and *exactly once* delivery, and we discuss the results of analysing the protocol in light of these three semantics.

A. QoS = 0 Protocol

The model of QoS = 0 protocol is straightforward. The client process is called from the top level protocol with the *Publish* message. This process is then run in parallel with the server process, which upon receiving the *Publish* message, it publishes it on the *pub* channel where interested subscribers are listening. For simplicity, we assume that the message is published as is. However, a more refined (but not of interest to us) server process would be expected to extract the relevant payload from *Publish* before publishing the actual data. We formalise the semantics of the protocol for the case of QoS = 0 in terms of the following theorem.

Theorem 3 (Delivery Semantics For QoS = 0)**.** *The MQTTv3.1 protocol for the case of QoS = 0 has a delivery semantics of the publish message to the subscribers of "at most once".*

Proof: Given the definition of the subscribers' process in the previous section, a run of this protocol would be equivalent to the following in the absence of any attackers: $(\mathit{Client}(\mathit{Publish}) \mid \mathit{Server}() \mid \mathit{Subscriber}())$. Analysing the process renders the following value of Φ:

$$\phi = \{x \mapsto \{\mathit{Publish}\}, x'_1 \mapsto \{\mathit{Publish}\}\}$$

From this, we can see that the message arrives at the subscriber. However, if we re-run the analysis with the attacker process activated: $(\mathit{Client}(\mathit{Publish}) \mid \mathit{Server}() \mid \mathit{Subscriber}() \mid \mathit{Attacker}())$, we obtain the following interesting outcome:

$$\phi_{atk} = \{y'_1 \mapsto \{\mathit{Publish}\}\}$$

This case shows a run of the protocol, which leads to only y' being instantiated with *Publish*. There is no instantiation of the x or x' variables.

From these results, it is easy to see that there are two possible outcomes. The first value of ϕ represents a normal run where $x' \mapsto \{\mathit{Publish}\}$, whereas in the second value of ϕ_{atk}, we have that $x' \mapsto \{\}$ by the definition of the default state ϕ_0. Hence, it is straightforward to see that the protocol *may* deliver the published message to the subscribers, and therefore, it correctly exhibits the at most once delivery semantics. ■

QoS Level 0 Protocol:

$\mathit{Client}(\mathit{Publish}) \mid \mathit{Server}()$, where:

$$\mathit{Client}(z) \overset{\mathtt{def}}{=} \overline{c}\langle z\rangle$$

$$\mathit{Server}() \overset{\mathtt{def}}{=} c(x).\overline{\mathit{pub}}\langle x\rangle$$

QoS Level 1 Protocol:

$\mathit{Client}(\mathit{Publish}) \mid \mathit{Server}()$, where:

$$\mathit{Client}(z) \overset{\mathtt{def}}{=} \overline{c}\langle z\rangle.\mathtt{timer}^{t}(c'(y), \mathit{Client}(\mathit{Publish}_{DUP}))$$

$$\mathit{Server}() \overset{\mathtt{def}}{=} !c(x).\overline{\mathit{pub}}\langle x\rangle.\overline{c'}\langle \mathit{Puback}\rangle$$

QoS Level 2 Protocol:

$\mathit{Client}(\mathit{Publish}) \mid \mathit{Server}()$, where:

$$\mathit{Client}(z) \overset{\mathtt{def}}{=} \overline{c}\langle z\rangle.\mathtt{timer}^{t}(c(y).\mathit{ClientCont}(y), \mathit{Client}(\mathit{Publish}_{DUP}))$$

$$\mathit{ClientCont}(u) \overset{\mathtt{def}}{=} \overline{c'}\langle \mathit{Pubrel}_{u}\rangle.\mathtt{timer}^{t'}(c'(w), \mathit{ClientCont}(u))$$

$$\mathit{Server}() \overset{\mathtt{def}}{=} !c(l).(\mathit{ServerLate(l)} \ + \ \mathit{ServerEarly}(l))$$

$$\mathit{ServerLate}(x) \overset{\mathtt{def}}{=} (\overline{c}\langle \mathit{Pubrec}_{x}\rangle.c'(v).\overline{\mathit{pub}}\langle x\rangle.\overline{c'}\langle \mathit{Pubcomp}_{v}\rangle.!(c'(v').\overline{c'}\langle \mathit{Pubcomp}_{v'}\rangle))$$

$$\mathit{ServerEarly}(x) \overset{\mathtt{def}}{=} (\overline{\mathit{pub}}\langle x\rangle.\overline{c}\langle \mathit{Pubrec}_{x}\rangle.c'(q).\overline{c'}\langle \mathit{Pubcomp}_{q}\rangle.!(c'(q').\overline{c'}\langle \mathit{Pubcomp}_{q'}\rangle))$$

Figure 3. A model of MQTTv3.1 in TPi considering the three levels of QoS.

B. QoS = 1 Protocol

The QoS = 1 protocol has a semantics of "at least once" delivery. We model this in Figure 3 as a client process, which starts by sending a *Publish* message to the server. The server is capable of inputting this message, publishing it to the subscribers and then replying back to the client with the *Puback* message. Again, for simplicity, we abstract away from the structure of both *Publish* and *Puback*, and point out here that a more refined treatment of these messages (i.e. extracting their payload) does not affect our analysis in the paper.

The next part is the main difference from the QoS = 0 case above. The client will wait for a finite amount of time, t, on its input channel c' for the *Puback* message from the server. If this message delays (for any communication failing reason), the client will choose to re-call its process with a new $\mathit{Publish}_{DUP}$ message. The difference between $\mathit{Publish}_{DUP}$ and *Publish* is that the DUP bit is set in the former to indicate that it is a duplication of the latter. The server on its side is capable of receiving this new publish message since its behaviour is replicated, which means that it can restart its process any number of times required by the context.

The two channels, c and c', distinguish between the two parts of the protocol (*Publish* and *Puback* parts). This is not necessary in practice, however it renders our model much simpler by avoiding unnecessary interferences between these two parts. In practice, there would some message validation mechanisms to prevent such interferences occurring.

We formalise the delivery semantics for this protocol in terms of the following theorem.

Theorem 4 (Delivery Semantics For QoS = 1)**.** *The MQTTv3.1 protocol for the case of QoS = 1 has a delivery semantics of the publish message to the subscribers of "at least once".*

Proof: Again, we first analyse the protocol under no attackers. In this case, we find the following subset value for ϕ:

$$\phi = \{x_1 \mapsto \{\mathit{Publish}\}, y \mapsto \{\mathit{Puback}\}, x'_1 \mapsto \{\mathit{Publish}\}\}$$

This implies normal behaviour, where the published message eventually arrives at the subscriber only once. We now re-run the analysis with the attacker activated, which produces the following subset value of ϕ_{atk}, where k is set to 3 to allow three runs of the protocol to take place:

$$\phi_{atk} = \{x_1 \mapsto \{\mathit{Publish}\}, u'_1 \mapsto \{\mathit{Puback}\}, x'_1 \mapsto \{\mathit{Publish}\},$$
$$x_2 \mapsto \{\mathit{Publish}_{DUP}\}, u'_2 \mapsto \{\mathit{Puback}\},$$
$$x'_2 \mapsto \{\mathit{Publish}_{DUP}\}, x_3 \mapsto \{\mathit{Publish}_{DUP}\},$$
$$u'_3 \mapsto \{\mathit{Puback}\}, x'_3 \mapsto \{\mathit{Publish}_{DUP}\}\}$$

The first subset of name substitutions corresponds to the first run where the attacker interferes with the protocol by consuming the *Puback* message. In the next two subsets, the client will issue a duplicate $\mathit{Publish}_{DUP}$. In both of these subsets, the attacker continues to consume the acknowledgment message and the client will continue to restart the protocol. Examining these results, we can easily see that the subscribers' input x' has more than one instantiation of the message *Publish*, including when the DUP bit is set. This indicates that the message may arrive more than once at the subscriber. ■

C. QoS = 2 Protocol

The last protocol represents the highest quality of service level, indicated by the QoS bit setting of 2. The model of Figure 3 contains again the definitions of the client and the publishing server. Similar to (and for the same reasons above) for the case of QoS = 1, we use two channels for the client: c for the first part ending with the sending of *Pubrec* and c' for the second part ending with the sending of *Pubcomp*.

The client process has two parts. The first could be reiterated, which will result in the *Publish* message being resent with the DUP bit set in case the *Pubrec* message is not received from the server within a time bound of t units. Note here that the standard protocol of [1] is not clear regarding the resent message. There is no explicit mentioning that the resent publish message is considered different from the original one. The assumption we make is that since DUP is set, then the resent message is a "duplicate" of the original one and therefore it is the same message.

The second part of the client process, *ClientCont*, is instantiated by the first part only if *Pubrec* is received from the server within the time bound t. In this case, it will send a *Pubrel* message to the server parameterised by the same message id as received in the previous message (hence we write Pubrel_u). After this, it waits for an amount of time t' for the last message from the server, *Pubcomp*, at which point it terminates once this message is received. If this last message does not arrive within the time bound t', it will re-call itself (i.e. the *ClientCont* part), which will result in the re-commencement of the protocol from the point of the sending of the Pubrel_u message. We believe the above two timed input actions model adequately the requirement that "*If a failure is detected, or after a defined time period, the protocol flow is retried from the last unacknowledged protocol message; either the PUBLISH or PUBREL.*" [1, pp. 38].

Finally, the last part of the protocol represents the server process. This process after receiving the initial publish message splits into a choice of two processes, *ServerEarly* and *ServerLate*. The main difference between these is whether the publish message is published to the subscribers before (i.e. early) or after (i.e. late) sending the second message of the protocol Pubrec_x, which is parameterised by the message id received in the first message from the client.

The standard provides two alternative options for this case [1, pp. 38]. The first follows the sequence of actions (store message, publish message and delete message), whereas the second follows the sequence of actions (store message id, publish message and delete message id). We term the former a *late publish semantics* and the latter an *early publish semantics*. The standard's document states that "The choice of semantic is implementation specific and does not affect the guarantees of a QoS level 2 flow" [1, pp. 38], however, we demonstrate next in terms of the output of our static analysis later that this is not generally true.

The whole server process is replicated in order to be able to receive a repeat publish message from the client in the event that Pubrec_x is not received at the client within the time limit.

The server process, after sending Pubrec_x, goes into the second part of the protocol. In this part, it listens on $c'(v)$ or $c'(q)$ for the incoming *Pubrel* message from the client. It then continues depending on the choice made earlier to either publish the message and send $\mathit{Pubcomp}_v$ or just send $\mathit{Pubcomp}_q$. In both cases, the *Pubcomp* message is parameterised by the message id from the received *Pubrel* message from the client.

The final part now commences, which is a replicated process that again listens for the *Pubrel* message from the client, and once this is received, it sends another *Pubcomp* message back to the client. This last part of the server process is similar in both sides of the choice and it will replicate itself until the client receives successfully the *Pubcomp* message, at which point the client will cease resending *Pubrel* messages.

It is worth noting here that our model above assumes that the implementation of the server will cater for a nondeterministic choice of both the early and late publish semantics. However, it is also possible, as we shall see in the next section, to model and analyse the server assuming only one of the two semantics of message publishing is implemented. This would be equivalent to modeling the server process as either $!c(l).\mathit{ServerLate(l)}$ or $!c(l).\mathit{ServerEarly}(l)$.

We now capture the delivery semantics for this protocol in terms of the following property.

Property 1 (Delivery Semantics For QoS = 2)**.** *The MQTTv3.1 protocol for the case of QoS = 2 has a delivery semantics of the publish message to the subscribers of exactly once.* □

In the first analysis we run, the attacker is deactivated. We obtain the following subset value for Φ when $k = 1$:

$$\phi = \{x_1 \mapsto \{\mathit{Publish}\}, y \mapsto \{\mathit{Pubrec}_x\}, v_1 \mapsto \{\mathit{Pubrel}_u\}, x'_1 \mapsto \{\mathit{Publish}\}, w \mapsto \{\mathit{Pubcomp}_v\}, x_1 \mapsto \{\mathit{Publish}\}, x'_1 \mapsto \{\mathit{Publish}\}, y \mapsto \{\mathit{Pubrec}_x\}, q_1 \mapsto \{\mathit{Pubrel}_u\}, w \mapsto \{\mathit{Pubcomp}_q\}, \ldots\}$$

The substitutions correspond to normal runs of the protocol for the two choices of the late and early publish semantics. Now, let's examine some of the results of the static analysis when the attacker is *activated*. In particular, we consider the case of the early publish semantics where we analyse in the context of the server $!c(l).\mathit{ServerEarly}(l)$ and the simple attacker model $(c(y') + c'(u'))$. We obtain the following interesting subset of the results, with $k = 2$:

$$\phi_{atk1} = \{x_1 \mapsto \{\mathit{Publish}\}, x'_1 \mapsto \{\mathit{Publish}\}, y'_1 \mapsto \{\mathit{Pubrec}_x\}, x_2 \mapsto \{\mathit{Publish}_{DUP}\}, x'_2 \mapsto \{\mathit{Publish}_{DUP}\}, y \mapsto \{\mathit{Pubrec}_x\}, q_1 \mapsto \{\mathit{Pubrel}_u\}, w \mapsto \{\mathit{Pubcomp}_q\}, \ldots\}$$

The result is interesting, as it represents a single interference case by the attacker (since $k = 2$). The attacker manages to consume the *Pubrec* message ($y'_1 \mapsto \{\mathit{Pubrec}_x\}$) before the client does so. As a result, the first part of the protocol is repeated and hence, in addition to the initial publish message ($x'_1 \mapsto \{\mathit{Publish}\}$), this leads to a second instance of this message to be announced to the subscribers ($x'_2 \mapsto \{\mathit{Publish}_{DUP}$).

Next, we re-apply the analysis this time on the case of the full server model and the simple non-replicated attacker model, where again we set $k = 2$ for simplicity:

$$\phi_{atk2} = \{x_1 \mapsto \{\mathit{Publish}\}, x'_1 \mapsto \{\mathit{Publish}\}, y'_1 \mapsto \{\mathit{Pubrec}_x\}, x_2 \mapsto \{\mathit{Publish}_{DUP}\}, y \mapsto \{\mathit{Pubrec}_x\}, v_1 \mapsto \{\mathit{Pubrel}_u\}, x'_2 \mapsto \{\mathit{Publish}_{DUP}\}, w \mapsto \{\mathit{Pubcomp}_v\}, \ldots\}$$

This represents another case of the attacker interfering with the protocol, however unlike the case of the first attack, a different choice of the publish semantics is made here in terms of the re-transmission of first part of the protocol. Here, we find that the Pubrec_x acknowledgment message sent by the server is captured by the attacker after an early publish semantics choice is taken involving announcing the publish message to the subscribers (by means of $x'_1 \mapsto \{\mathit{Publish}\}$). This failure in delivering Pubrec_x to the client causes a restart of the protocol, however, in this case a different choice is made with the late publish semantics. Continuing with this run, the second part of the protocol causes the duplicated publish message $\mathit{Publish}_{DUP}$ to be announced again to the subscribers. Note that this attack would not be possible if either *ServerLate* or *ServerEarly* process only is adopted, but not a choice of both.

VI. Discussion

Considering the results of the above three analyses, we find that the protocol as described in [1] and specified in Section IV is quite sound in the cases of QoS = 0 and QoS = 1, however in the case of QoS = 2, the protocol description in [1] is ambiguous in terms of how the server is supposed to deal with the publication of duplicated messages and whether it may or may not implement both choices of early and late publish semantics.

More specifically, considering the two results of ϕ_{atk1} and ϕ_{atk2} for the case of QoS = 2, we conclude that there is more than one scenario where the protocol fails in adhering to its "exactly once" delivery semantics and where *Publish*/$\mathit{Publish}_{DUP}$ are delivered to the subscribers more than once. As a result, Property 1 defined in the previous section does not hold. It is also noteworthy that this failure occurs only due to more-than-once delivery reasons, and there is no case where the failure occurs due to less-than-once (i.e. zero) delivery, unless we assume a powerful attacker with replicated inputs whose capable of continuously blocking the *Pubrec* message from being delivered to the client.

We propose here the following two enhancements to the MQTT protocol in the case of QoS = 2. First, separate the implementation of the early and late publish semantics. This is currently not explicit in the specification of the protocol in [1]. We suggest that this be made explicit, if this is the intention, so that implementations of the protocol are not confused with the choice.

The second enhancement we suggest is to introduce a conditional guard on the publishing action at the server side, for both choices of early and late publish semantics. This can be described as the following modified exchange of messages and actions:

Client → *Server*	:	**Publish**
Client Action	:	*Store Message*
Server Actions	:	*Store Message*
		OR
		Store Message ID,
		([¬*Published(Message ID)*]
		Publish message to subscribers)
Server → *Client*	:	**Pubrec**
Client → *Server*	:	**Pubrel**
Server Actions	:	([¬*Published(Message)*]
		Publish message to subscribers),
		Delete Message
		OR
		Delete Message ID
Server → *Client*	:	**Pubcomp**
Client Action	:	*Discard Message*

Where *Published(Message)* is a predicate on the local state of the server in which the message or its id is stored. This predicate checks whether a message has already been published. We also overload the predicate in the case of *Published(Message ID)* to check whether a message corresponding to an ID has been already published or not. If this predicate is not true, the publish action is neglected and the next action in line is applied.

In order to incorporate the predicate *Published(Message)* in our TPi-based specification, we need to modify the syntax of the language to include a new term $P \triangleleft p \triangleright Q$, where p is any local predicate, which is True, the process will evaluate as P and if False, it will evaluate as Q. The substitution semantics then will also include an additional rule on this new term,

$$\mathcal{S}([\![P \lhd p \rhd Q]\!])\ \rho\ \phi_{\mathcal{S}} = \left\{ \begin{array}{l} \mathcal{S}([\![P]\!])\ \rho\ \phi_{\mathcal{S}},\ \text{if } p = \textit{True} \\ \mathcal{S}([\![Q]\!])\ \rho\ \phi_{\mathcal{S}}, \text{if } p = \textit{False} \end{array} \right.$$

This will then lead to the redefinition of the early and late publish semantics specification to include the additional expected conditional predicate, *published*(x), as shown in Figure 4. We assume that the predicate is able to consider the contents of the parameter x containing the message and its id.

Repeating the analysis for the case of the presence of the attacker and for the early and late publish semantics renders the following two subsets of the final ϕ environment:

$$\phi_{atk1} = \{x_1 \mapsto \{\textit{Publish}\}, x'_1 \mapsto \{\textit{Publish}\}, y'_1 \mapsto \{\textit{Pubrec}_x\}, x_2 \mapsto \{\textit{Publish}_{DUP}\}, y \mapsto \{\textit{Pubrec}_x\}, q_1 \mapsto \{\textit{Pubrel}_u\}, w \mapsto \{\textit{Pubcomp}_q\}, \ldots\}$$

$$\phi_{atk2} = \{x_1 \mapsto \{\textit{Publish}\}, x'_1 \mapsto \{\textit{Publish}\}, y'_1 \mapsto \{\textit{Pubrec}_x\}, x_2 \mapsto \{\textit{Publish}_{DUP}\}, y \mapsto \{\textit{Pubrec}_x\}, v_1 \mapsto \{\textit{Pubrel}_u\}, w \mapsto \{\textit{Pubcomp}_v\}, \ldots\}$$

Both of which clearly show no instantiation of the second and further copies of the x' input variable for the subscriber process, in which case we assume that the modification of Figure 4 achieves its intended aim.

VII. Related Work

Publish/subscribe is increasingly becoming an important communication paradigm [8], in particular within the domain of sensor device networks and the Internet-of-Things where messages can be communicated with more efficiency and less consumption of the devices' limited computational power. IBM's MQTT-S protocol [9] was one of the first industrially backed lightweight publish/subscribe protocols that was deployed for wireless sensor and actuator networks. This was followed in year 2010 by version 3.1 [1], which is currently undergoing standardisation by OASIS.

There has been very little effort in applying formal analysis tools to low-level communication protocols, mainly due to the novelty of such protocols and their very recent arrival at the scene of communication protocols. An early attempt in [10] was made to model formally publish/subscribe protocols to capture their essential properties such as minimality and completeness, however, without any attempt to incorporate hostile environments within which these protocols may run. One aspect of their model is the use of an incrementing global clock $\mathcal{T}$, similar to our concept of the function $\eth(P)$, which is needed in order to model the passing of time.

In [11], the authors define a formal model of publish/subscribe protocols, within the domain of Grid computing, based on Petri-Nets. Their model offers a mechanism for the composition of existing publish/subscribe protocols with model, hence offering a friendly approach for the validation of such protocols. Nonetheless, the focus of their work is mostly on Grid computing scenarios.

The work of [12] is an early attempt in discussing security properties and requirements desirable in publish/subscribe protocols, in particular within the domain of Internet-based peer-to-peer systems, where such protocols became popular in their early forms.

Within the domain of sensor network protocols, there is more focus of effort on the formal analysis and verification of such protocols. For example, in [13], the authors apply model checking techniques in the verification of a medium access control protocol called LMAC. Similarly, in [14] propose a formal model of flooding and gossiping protocols for analysing their performance probabilistic properties. More recently, [15] proposed a formal model and analysis of clock-synchronised protocols in sensor networks based on timed automata.

Finally, the static analysis technique proposed in this paper builds on previous works by the author, such as [6] and [6]. More specifically, the analysis of the timed π-calculus was defined and used successfully in [5] for detecting in a precise manner man-in-the-middle attacks in timed systems.

VIII. Conclusion and Future Work

We have modeled and analysed in this paper the MQ Telemetry Transport version 3.1 protocol, which is a lightweight broker-based publish subscribe protocol that is used in communications with small devices that exhibit limited computational and storage power. As far as we know, this is the first attempt to formally analyse the MQTT protocol. MQTT is maintained by the Organization for the Advancement of Structured Information Standards (OASIS).

We found that the first two QoS modes of operation in the protocol are clearly specified and their message delivery semantics to subscribers can be easily verified to hold. However, according to the results of the analysis, the last case of an "exactly once" delivery semantics has potential vulnerabilities where a simple attacker model that adheres to the specified threat model of the protocol can cause the semantics to be undermined. At best, this semantics is vaguely specified in the standard [1], particularly in relation to issues to do with the choice of server-side behaviour.

Future research will be focused on studying the properties of the protocol under more aggressive attacker models and we plan to propose refined versions of the protocol, including the use of lightweight cryptography in scenarios where authentication of the small devices is required. In addition, although we carried out a simple modification to the QoS = 2 case that removes the duplicated publish message vulnerability, we would like to further investigate in-depth additional mechanisms for improving further the protocol specification. Such in-depth investigation will most probably utilise automated toolkits such as the AVISPA (www.avispa-project.org) or Rodin (www.event-b.org/) toolkits.

$$\begin{aligned}
&\mathit{Client}(\mathit{Publish}) \mid \mathit{Server}(), \text{ where:}\\
&\mathit{Client}(z) \overset{\texttt{def}}{=} \overline{c}\langle z\rangle.\texttt{timer}^{t}(c(y).\mathit{ClientCont}(y), \mathit{Client}(\mathit{Publish}_{DUP}))\\
&\mathit{ClientCont}(u) \overset{\texttt{def}}{=} \overline{c'}\langle \mathit{Pubrel}_u\rangle.\texttt{timer}^{t'}(c'(w), \mathit{ClientCont}(u))\\
&\mathit{Server}()\overset{\texttt{def}}{=}!c(l).(\mathit{ServerLate(l)} \;+\; \mathit{ServerEarly}(l))\\
&\mathit{ServerLate}(x) \overset{\texttt{def}}{=} \overline{c}\langle \mathit{Pubrec}_x\rangle.c'(v).\\
&(\overline{\mathit{pub}}\langle x\rangle.\overline{c'}\langle \mathit{Pubcomp}_v\rangle.!(c'(v').\overline{c'}\langle \mathit{Pubcomp}_{v'}\rangle)) \lhd (\neg\ \mathit{published}(x)) \rhd (\overline{c'}\langle \mathit{Pubcomp}_v\rangle.!(c'(v').\overline{c'}\langle \mathit{Pubcomp}_{v'}\rangle))\\
&\mathit{ServerEarly}(x)\overset{\texttt{def}}{=}\\
&\overline{\mathit{pub}}\langle x\rangle.\overline{c}\langle \mathit{Pubrec}_x\rangle.c'(q).\overline{c'}\langle \mathit{Pubcomp}_q\rangle.!(c'(q').\overline{c'}\langle \mathit{Pubcomp}_{q'}\rangle) \lhd (\neg\ \mathit{published}(x))\rhd\\
&\overline{c}\langle \mathit{Pubrec}_x\rangle.c'(q).\overline{c'}\langle \mathit{Pubcomp}_q\rangle.!(c'(q').\overline{c'}\langle \mathit{Pubcomp}_{q'}\rangle)
\end{aligned}$$

Figure 4. The modified model for the case of QoS = 2.

Acknowledgment

The author would like to thank the anonymous reviewers for their valuable feedback on the initial form of this paper. Thanks also to Paul Fremantle, WSO2, for providing useful comments on the MQTT specification and for raising the issue to OASIS (https://tools.oasis-open.org/issues/browse/MQTT-209).

References

[1] D. Locke, "MQ Telemetry Transport (MQTT) V3.1 Protocol Specification," 2010.

[2] K. Birman and T. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," *SIGOPS Oper. Syst. Rev.*, vol. 21, no. 5, pp. 123–138, Nov. 1987.

[3] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes," *Information and Computation*, vol. 100(1), pp. 1–77, Sep. 1992.

[4] M. Berger and K. Honda, "The Two-Phase Commitment Protocol in an Extended Pi-Calculus," *Electronic Notes in Theoretical Comp. Science*, vol. 39, no. 1, 2000.

[5] B. Aziz and G. Hamilton, "Detecting Man-in-the-Middle Attacks by Precise Timing," in *Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies*, ser. SECURWARE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 81–86.

[6] B. Aziz, "A static analysis framework for security properties in mobile and cryptographic systems," Ph.D. dissertation, School of Computing, Dublin City University, Dublin, Ireland, 2003.

[7] B. Aziz and G. Hamilton, "A Privacy Analysis for the π-calculus: The Denotational Approach," in *Proceedings of the* 2^{nd} *Workshop on the Specification, Analysis and Validation for Emerging Technologies*, ser. Datalogiske Skrifter, no. 94. Copenhagen, Denmark: Roskilde University, Jul. 2002.

[8] A. J. Stanford-Clark and G. R. Wightwick, "The Application of Publish/Subscribe Messaging to Environmental, Monitoring, and Control Systems," *IBM J. Res. Dev.*, vol. 54, no. 4, pp. 396–402, Jul. 2010.

[9] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks," in *Proceedings of the Third International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2008)*. IEEE, 2008, pp. 791–798.

[10] R. Baldoni, M. Contenti, S. T. Piergiovanni, and A. Virgillito, "Modelling Publish/Subscribe Communication Systems: Towards a Formal Approach," in *8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*. IEEE Computer Society, 2003, pp. 304–311.

[11] L. Abidi, C. Cerin, and S. Evangelista, "A Petri-Net Model for the Publish-Subscribe Paradigm and Its Application for the Verification of the BonjourGrid Middleware," in *Proceedings of the 2011 IEEE International Conference on Services Computing*, ser. SCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 496–503.

[12] C. Wang, A. Carzaniga, D. Evans, and A. Wolf, "Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9 - Volume 9*, ser. HICSS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 303–.

[13] A. Fehnker, L. V. Hoesel, and A. Mader, "Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks," in *Proceedings of the 6th International Conference on Integrated Formal Methods*, ser. IFM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 253–272.

[14] A. Fehnker and P. Gao, "Formal Verification and Simulation for Performance Analysis for Probabilistic Broadcast Protocols," in *Proceedings of the 5th International Conference on Ad-Hoc, Mobile, and Wireless Networks*, ser. ADHOC-NOW'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 128–141.

[15] F. Heidarian, J. Schmaltz, and F. W. Vaandrager, "Analysis of a clock synchronization protocol for wireless sensor networks," *Theor. Comput. Sci.*, vol. 413, no. 1, pp. 87–105, 2012.