A Utility-based Reputation Model for the Internet of Things

Benjamin Aziz[*], Paul Fremantle[*], Rui Wei[†], and Alvaro Arenas[‡]

[*]School of Computing
University of Portsmouth
Portsmouth, United Kingdom
{benjamin.aziz,paul.fremantle}@port.ac.uk
[†]Department of Computer and Information Technology
Beijing Jiaotong University
Beijing, China
12120463@bjtu.edu.cn
[‡]IE Business School
IE University
Madrid, Spain
alvaro.arenas@ie.edu

Abstract. The MQTT protocol has emerged over the past decade as a key protocol for a number of low power and lightweight communication scenarios including machine-to-machine and the Internet of Things. In this paper we develop a utility-based reputation model for MQTT, where we can assign a reputation score to participants in a network based on monitoring their behaviour. We mathematically define the reputation model using utility functions on participants based on the expected and perceived behaviour of MQTT clients and servers. We define an architecture for this model, and discuss how this architecture can be implemented using existing MQTT open source tools, and we demonstrate how experimental results obtained from simulating the architecture compare with the expected outcome of the theoretical reputation model.

Keywords: Internet of Things, Utility Reputation, Trust Management

1 Introduction

The Internet of Things (IoT) is an area where there is significant growth: both in the number of devices deployed and the scenarios in which devices are being used. One of the challenges for the Internet of Things is supporting network protocols which utilise less energy, lower bandwidth, and support smaller footprint devices. One such protocol is the MQ Telemetry Transport (MQTT) protocol [15], which was originally designed to support remote monitoring and Supervisory Control And Data Acquisition (SCADA) scenarios but has become popular for the IoT.

Another challenge with IoT networks is that small devices may not perform as well as needed due to a number of factors including: network outages or poor network performance due to the use of 2G or other low bandwidth networks, power outages for devices powered by batteries, deliberate vandalism or environmental damage for devices placed in public areas, and many other such challenges. Therefore we identified that a reputation model for devices connecting by MQTT would be a useful construct to express consumers' (applications') trust in the behaviour and performance of these devices as well as measure the level of performance of the server aggregating data from such devices according to some predefined Service Level Agreement (SLA). In addition, we implemented the reputation model to demonstrate that it could be used in real MQTT networks.

Our model of reputation is based on the notion of a *utility function*, which formally expresses the consumer's level of satisfaction related to various issues of interest against which the reputation of some entity is measured. In the case of MQTT networks, one notable such issue is the Quality of Service (QoS) with regards to the delivery of messages; whether messages are delivered exactly once, more than once or at most once to their consumers. The model, inspired by previous works [6, 19], is general enough to be capable of defining the reputation of client devices and servers at various levels of abstraction based on their level of performance in relation to the delivery of messages issue of interest.

The paper starts with an overview of the MQTT protocol (Section 2). From this, we then mathematically define the reputation model for MQTT clients and server (Section 3), based on their ability to keep to the requirements of the protocol. We then outline a system architecture (Section 4) for monitoring the MQTT protocol and thereby being able to calculate the reputation by observing the behaviour of MQTT clients and server in a real network. We show how this system was implemented and we demonstrate the results of this implementation (Section 5). Finally we look at related work (Section 6) and conclude the paper outlining areas for further research (Section 7).

2 MQTT Overview

MQTT [9] is described 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 IoT is composed of, to communicate over low-bandwidth and unreliable networks. The publish/subscribe message pattern [10], on which MQTT is based, provides for one-to-many message distribution with three varieties of delivery semantics, based on the level of QoS expected from the protocol. 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 is termed the QoS = 0 protocol. In the second case of "at least once" semantics, certain mechanisms are incorporated to allow for message duplication. Despite the guarantee of delivering the message, there is no guarantee that duplicates will be suppressed. 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. The protocol also defines message structures 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. Servers are often also referred to as *brokers*, as they intermediate between the data publishers and subscribers.

3 A Reputation Model for MQTT

We show in this section how the model of reputation defined for business processes in [7, 8] can be adapted, with minimum changes, to the MQTT protocol to obtain the reputation of client devices and the server.

3.1 Monitoring Events

Central to the model defined by [7, 8] was the notion of an *event*, which is a signal produced by an independent *monitor system*, which is monitoring the interactions occurring between the different entities in the monitored environment, in this case the client and server entities participating in the MQTT protocol. An event is defined as follows:

$$\mathit{Event} : \mathit{TimeStamp} \times \mathit{Ag} \times \mathit{Msg} \times \mathit{Id} \times \mathbb{N}$$

where *TimeStamp* is the timestamp of the event generated by the monitor system issuing it, *Ag* is the identity of the agent (client device or server) to whom the event is related, *Msg* is the specific message of interest (e.g. *Publish* and *Pubrel* messages), *Id* is an identity value of the protocol instance and finally, $\mathbb{N}$ is a natural number representing the number of times the message *Msg* has been monitored, i.e. was sent.

For example, the following event, issued at monitor system's local time:

$$ev_{ex1} = (12\!:\!09\!:\!52, \mathit{temp\_sensor}, \mathit{Publish}, 1234, 2)$$

denotes that the *temp_sensor* device has been monitored, within the instance number 1234 of the protocol, to have sent twice the *Publish* message to the server responsible for collecting environment temperature data. On the other hand, the following event issued at local time 12 : 19 : 02:

$$ev_{ex2} = (12\!:\!19\!:\!02, \mathit{temp\_server}, \mathit{Publish}, 1234, 1)$$

denotes that the server responsible for the environment temperature, *temp_server*, has been monitored, within the same instance number 1234 of the protocol, to have published only once the specific message *Publish* to the subscribers of the temperature topic. In both these examples, the assumption is that the monitor system is capable of detecting that the protocol instance being monitored has terminated before it issues any events related to that instance. Although theoretically this is impossible due to the halting problem, in practical terms, the monitor system can assume the protocol to have terminated after some reasonable amount of time has elapsed since the last protocol message.

The monitor generates events in the above form, which are used by a *reputation engine* to determine the reputation values for client devices and servers in an MQTT-based environment. The reputation engine will then use a *utility function* pre-supplied to the engine by subscribers to determine the level of satisfaction of a subscriber with regards to the results reported within an event:

$$\begin{array}{|l} \mathit{utility} : \mathit{Event} \times \mathit{SLA} \rightarrow [0,1] \\ \hline \forall (t,a,m,i,n) \in \mathit{Event}, \mathit{sla} \in \mathit{SLA} \bullet \mathit{utility}((t,a,m,i,n),\mathit{sla}) = r \in \mathbb{R} \end{array}$$

This utility function will consider a SLA, defined as follows:

$$\begin{array}{|l} \mathit{SLA} : \mathit{Ag} \times \mathit{Top} \times \mathit{Iss} \rightarrow \mathbb{N}^0 \end{array}$$

Here the SLA considers an issue of interest to the subscriber, *Iss*, which will be in our case the QoS level value fixed to one of 0, 1 or 2, expected from a particular agent *Ag* in relation to a specific topic *Top*. The outcome of the utility function is a real number r representing the satisfaction level of the subscriber in terms of both the SLA and the real values reported by events.

For example, consider the following SLA instance

$$\mathit{sla} = ((\mathit{temp\_server}, \mathit{temperature}, \mathit{QoS}), 2)$$

then given the event ev_{ex2}, the utility function could return the following value:

$$\mathit{utility}(t, \mathit{temp\_server}, \mathit{Publish}, 1234, 1, ((\mathit{temp\_server}, \mathit{temperature}, \mathit{QoS}), 2)) = 1$$

This indicates that the subscriber's requirements have been fulfilled, as indicated by their SLA ($r = 1$), with the results reported by the event ev_{ex2}. On the other hand, considering the same SLA, the utility function might return:

$$\mathit{utility}(t, \mathit{temp\_server}, \mathit{Publish}, 1234, 0, ((\mathit{temp\_server}, \mathit{temperature}, \mathit{QoS}), 2)) = 0$$

to show that the subscriber has a satisfaction value of 0 since the number of times the message was delivered to the subscriber is lower (i.e. 0) than what its QoS level is defined in the SLA (i.e. 2), therefore breaching the exactly-once delivery semantics to the subscriber principle in MQTT.

Since the number of times a message is delivered will either confirm or not to the level of QoS expected by the subscriber, in all of the above cases, the score given will reflect either total satisfaction (i.e. 1) or total dissatisfaction (i.e. 0).

3.2 Reputation Models

After introducing the main notions of an event and a utility function, we can now define models of reputation for the clients (e.g. sensor devices) and the MQTT server (broker) that aggregates the messages from the clients before publishing them to the subscribers. The subscribers are assumed to be the business applications or data consumers, and we do not include them in the reputation model. The MQTT standard does not prohibit a client from acting as both a device (i.e. source of data) and a subscriber (i.e. consumer of data). However, in our case, we only measure the reputation of the "source of data" clients.

The Server Reputation Model The first reputation model reflects the behaviour of MQTT servers. Given a set of events, *Event*, captured by the monitor system and relevant to the server for whom the reputation is being calculated, then we can define the server's reputation function computed at a particular point in time and parameterised by a specific SLA as follows:

$$[Srv, SLA, TimeStamp]$$

$$s\_rep\_sla : Srv \times SLA \times TimeStamp \rightarrow [0, 1]$$

$$\forall\, eset_s : \wp(Event) \bullet$$

$$s\_rep\_sla(s, sla, t) = \frac{\sum_{ev\, \in eset_s . snd(ev)\, = fst(sla) = s \,\wedge\, id\_top(ev, sla)} \varphi(t, te) utility(ev, sla)}{\# eset_s}$$

where $\# s$ denotes the cardinality of a set s and $\varphi(t, te)$ is a time discount function that puts more importance (emphasis) on events registered closer in time to the moment of computing the reputation. One definition of $\varphi(t, te)$ could be the time discount function defined by [13], which we redefine here as $\varphi(t, te) = e^{-\frac{t-te}{\lambda}}$, where t is the current time at which the reputation is calculated, te is the timestamp of the event being considered and λ is *recency scaling factor* used to adjust the value of the function to a scale required by the application. After this, the server reputation function, *s_rep_sla*, is defined as the weighted average of the utilities obtained from all the generated events with respect to some SLA.

The above definition aggregates the set of all relevant events, i.e. the events that first have the same server name as that appearing in the SLA and second that are on an instance of the protocol related to the topic of the SLA. The first condition is checked using the two operators *fst* and *snd*, which will return the first and second elements of a tuple, whereas the second condition is checked using the predicate $id\_top(ev, sla)$, which returns a True outcome if and only if the identity number of an instance of a protocol captured by *ev* corresponds to the topic value mentioned in the SLA *sla*. Considering the example events of the previous section, we would have the following calculation of $id\_top(ev, sla)$:

$id\_top(12{:}09{:}52, temp\_server, Publish, 1234, 1, ((temp\_server, temperature, QoS), 2)) = True$

The above definition calculates the sum of the time-discounted utility function values, with respect to the given SLA and the events gathered, and average these over the total number of events gathered ($\# eset_s$) in any one instance when this reputation value is calculated.

Based on the definition of *s_rep_sla*, we next aggregate the reputation of a server across every SLA that binds that server to its subscribers:

$$[\mathit{Srv}, \mathit{TimeStamp}]$$

$$\mathit{s\_rep} : \mathit{Srv} \times \mathit{TimeStamp} \to [0, 1]$$

$$\forall\, \mathit{slaset}_s : \wp(\mathit{SLA}) \bullet \mathit{s\_rep}(s, t) = \frac{\sum_{\mathit{sla} \,\in\, \mathit{slaset}_s . \mathit{fst}(\mathit{sla}) = s} \mathit{s\_rep\_sla}(s, \mathit{sla}, t)}{\#\mathit{slaset}_s}$$

Which provides a more general indication of how well a server s behaves in relegation to a number of subscribers. This reputation is again calculated in a particular point in time, t, however it is straightforward to further generalise this reputation function over some time range, between t and t'.

The Client Device Reputation Model After introducing the reputation model of the server, we define here the client's reputation model. Like the server, a client might also be implementing the QoS correctly, but it requires multiple reconnections, duplicate messages etc., while the server does not. For instance, if the devices are not sending PINGs or responding to them, or this is delayed, it might indicate a problem is more likely to occur in the future. Similarly, if the device needs to send multiple duplicate messages or needs to be sent duplicate messages, it also might indicate possible failure in the future. Thus, the reputation model for a client may be based on either the "Keep Alive/PING" case or the "Client's Retransmission Procedure" case. However, we start with the definition of an overall reputation model that generalises these two cases.

Given a set of events, *Event*, captured by the monitor system relevant to some client, then we define the client's reputation function computed at a particular point in time in a specific process (Keep Alive/PING procedure or retransmission procedure) and parameterised by a specific SLA as follows:

$$[\mathit{Client}, \mathit{SLA}, \mathit{TimeStamp}, \mathit{Procedure}]$$

$$\mathit{c\_rep\_sla\_p} : \mathit{Client} \times \mathit{SLA} \times \mathit{TimeStamp} \times \mathit{Procedure} \to [0, 1]$$

$$\forall\, \mathit{pset}_s : \wp(\mathit{Event}) \bullet \mathit{c\_rep\_sla\_p}(c, \mathit{sla}, t, p) = \frac{\sum_{\mathit{ev} \in \mathit{pset}_s . \mathit{snd}(\mathit{ev}) = \mathit{fst}(\mathit{sla}) = c \,\wedge\, \mathit{id\_top}(\mathit{ev}, \mathit{sla})} \varphi(t, \mathit{te})\, \mathit{utility}(\mathit{ev}, \mathit{sla})}{\#\mathit{pset}_s}$$

This definition gathers the set of all related events, i.e. the events that first have the same client name as that appearing in the SLA and second that are on an instance of the protocol related to the topic of the SLA. The definition is parameterised by the client, an SLA, a timestamp and the specific procedure (e.g. Keep Alive/PING or retransmission). The SLA represents what the expectation is, from the server's point of view, of the client's behaviour in the context of the specific procedure. Similar to the case of *s_rep_sla*, a utility function is applied to measure the satisfaction of the server, in a time-discounted manner, in relation to the client's behaviour and this is then averaged over the total number of events captured in a specific instance of time.

For example, consider the case of the Keep Alive/PING procedure, then *c_rep_sla_ka* is defined as the time-discounted average of the utilities obtained from all generated events with respect to the Keep Alive/PING procedure.

$$[\mathit{Client}, \mathit{SLA}, \mathit{TimeStamp}, \mathit{KeepAlive}]$$

$$\mathit{c\_rep\_sla\_ka} : \mathit{Client} \times \mathit{SLA} \times \mathit{TimeStamp} \times \mathit{KeepAlive} \to [0, 1]$$

$$\forall\, \mathit{pset}_s : \wp(\mathit{Event}) \bullet \mathit{c\_rep\_sla\_ka}(c, \mathit{sla}, t, \mathit{ka}) = \frac{\sum_{ev \in \mathit{kapingset}_s . \mathit{snd}(ev) = \mathit{fst}(\mathit{sla}) = c \,\wedge\, \mathit{id\_top}(ev, \mathit{sla})} \varphi(t, te)\, \mathit{utility}(ev, \mathit{sla})}{\#\mathit{kapingset}_s}$$

In this procedure, the client sends a Pingreq message within each KeepAlive time period, then the receiver answer with a Pingresp message when it receives a Pingreq message from the gateway to which it is connected. Clients should use KeepAlive timer to observe the liveliness of the gateway to check whether they are connected to broker. If a client does not receive a Pingresp from the gateway even after multiple retransmissions of the Pingresq message, it fails to connect with gateway during the Keep Alive period.

Hence, for the above example, using *id_top* to show a set of related events *ev* corresponds to the topic value mentioned in the SLA, *sla*, we would have that:

$\mathit{id\_top}(12 : 09 : 52, \mathit{client}, \mathit{Pingreq}, \mathit{False}, \mathit{False}, 1234, 1, ((\mathit{client}, \mathit{temperature}, \mathit{QoS}), 0)) = \mathit{True}$

The event $ev_{kaping} = (12 : 09 : 52, \mathit{client}, \mathit{Pingreq}, \mathit{False}, \mathit{False}, 1234, 1)$ generated by the monitor, could reflect a client device that has sent once the Pingreq message to connect to the gateway within the instance number 1234 of the protocol during the Keep Alive period. Then, given the SLA instance $\mathit{sla} = ((\mathit{client}, \mathit{temperature}, \mathit{QoS}), 0)$, the client should deliver this Pingreq message in relation to a specific topic (in this case *temperature*) at most once within each KeepAlive time period, but there is no guarantee the message will arrive.

From the definition of *c_rep_sla_p*, we generate a more general reputation for some client in a particular point in time t within a period, *Period*, as follows:

$$[\mathit{Client}, \mathit{SLA}, \mathit{TimeStamp}]$$

$$\mathit{c\_rep\_sla} : \mathit{Client} \times \mathit{SLA} \times \mathit{TimeStamp} \to [0, 1]$$

$$\forall\, \mathit{periodset}_s : \wp(\mathit{Period}) \bullet \mathit{c\_rep}_s \mathit{la}(c, \mathit{sla}, t) = \frac{\sum_{ev \in \mathit{periodset}_s . \mathit{snd}(ev) = \mathit{fst}(\mathit{sla}) = c \,\wedge\, \mathit{id\_top}(ev, \mathit{sla})} \mathit{c\_rep\_sla\_p}(c, \mathit{sla}, t, p)}{\#\mathit{periodset}_s}$$

Giving an example based on the Keep Alive/PING procedure, assume the KeepAliveTimer is set to 60, then calculating $\mathit{c\_rep\_sla}(c, \mathit{sla}, t)$ will give us the reputation of the client device during the whole Keep Alive period of 60 seconds. In another example, based on the retransmission procedure, we assume that Nretry is set to 10. Aggregating over the $\mathit{c\_rep\_sla}(c, \mathit{sla}, t)$ values yields reputation in relation to the client's retransmissions within a 10 time-unit limit.

Finally, based on the definition of *c_rep_sla*, we can further generalise the reputation value over all relevant SLAs for a specific client, c, and in a particular point in time, t, as follows:

$$
\begin{array}{|l}
[\mathit{Client}, \mathit{SLA}] \\
\mathit{c\_rep} : \mathit{Client} \times \mathit{SLA} \rightarrow [0,1] \\
\hline
\forall\, \mathit{slaset}_s : \wp(\mathit{SLA}) \bullet \mathit{c\_rep}(c,t) = \dfrac{\sum_{c_{rep}(c,\,t) \,=\, sla \,\in\, slaset_s.fst(sla) \,=\, c} \mathit{c\_rep\_sla}(s, sla, t)}{\#\mathit{slaset}_s}
\end{array}
$$

This definition gives a more general indication of how well the client device generally behaves in relation to the SLAs it holds with the server (possibly on behalf of the subscribers dealing with the server). These could include scenarios where the clients might use the Keep Alive/PING procedure to observe the liveliness of the gateway to check whether they are connected to a broker. Moreover, in the case of messages that expect a response, if the reply is not received within a certain time period, the client will be expected to retransmit this message.

The reputation model of a client in different procedures might cause different failures. Thus, as we demonstrated above the first reputation model, *c_rep_sla_p*, will lead to new models with slight variations capturing this variety of failures. For example, for the case of a client's retransmission procedure, all messages that are "unicast" to the gateway and for which a gateway's response is expected are supervised by a retry timer *Tretry* and a retry counter *Nretry*. The retry timer *Tretry* is started by the client when the message is sent and stopped when the expected gateway's reply is received. If the client does not receive the expected gateway's reply during the *Tretry* period, it will retransmit the message. In addition, the client should terminate this procedure after *Nretry* number of retransmissions and should assume that it is disconnected from the gateway. The client should then try to connect to another gateway only if it fails to re-connect again to the previous gateway.

One such client reputation, is defined based on a specific *TRetry* timer:

$$
\begin{array}{|l}
[\mathit{Client}, \mathit{SLA}, \mathit{TimeStamp}, \mathit{TRetry}] \\
\mathit{c\_reptr\_sla\_tr} : \mathit{Client} \times \mathit{SLA} \times \mathit{TimeStamp} \times \mathit{TRetry} \rightarrow [0,1] \\
\hline
\forall\, \mathit{tretryset}_s : \wp(\mathit{Event}) \bullet \mathit{c\_reptr\_sla\_tr}(c, sla, t, tr) = \\
\dfrac{\sum_{ev \in\, kapingset_s.snd(ev) \,=\, fst(sla) \,=\, c \,\wedge\, id\_top(ev,\, sla)} \varphi(t, te)\, utility(ev, sla)}{\#\mathit{tretryset}_s}
\end{array}
$$

For example, consider the following $sla = ((\mathit{client}, \mathit{temperature}, \mathit{QoS}), 1)$, then given the event $ev_{tretry} = (12:09:52, \mathit{client}, \mathit{Publish}, \mathit{False}, \mathit{True}, 1234, 2)$, it could reflect an event in the retransmission procedure. If the client does not receive a Puback message with QoS level 1 within a time period defined by the *TRetry* value, the client may resend the Publish message with the DUP flag set. When the server receives a duplicate message from the client, it re-publishes the message to the subscribers, and sends another Puback message.

Similarly, another variation of the client's reputation function may be based on the NRetry counter instead:

$$[\mathit{Client}, \mathit{SLA}, \mathit{TimeStamp}, \mathit{NRetry}]$$

$$\mathit{c\_repnr\_sla\_nr} : \mathit{Client} \times \mathit{SLA} \times \mathit{TimeStamp} \times \mathit{NRetry} \to [0,1]$$

$$\forall\, \mathit{nretryset}_s : \wp(\mathit{Event}) \bullet \mathit{c\_reptr\_sla\_tr}(c, \mathit{sla}, t, \mathit{nr}) = \frac{\sum_{ev \in \mathit{kapingset}_s . \mathit{snd}(ev) = \mathit{fst}(\mathit{sla}) = c \,\wedge\, \mathit{id\_top}(ev, \mathit{sla})} \varphi(t, te)\mathit{utility}(ev, \mathit{sla})}{\#\mathit{nretryset}_s}$$

Again, for the above definition, for $sla = ((client, temperature, QoS), 2)$, and given the event $ev_{nretry} = (12 : 09 : 52, client, Publish, False, False, 1234, 1)$, it could indicate that the client should not retransmit again in the retransmission period due to the fact that QoS is set to 2 (meaning the message is guaranteed to be delivered exactly-once to the subscribers). In this case, the DUP flag must be set to False, in order to prevent a retransmission.

4 A Reputation System Architecture for MQTT

Our architecture for a reputation system for an MQTT network is composed of a *reputation monitor* and a *reputation engine*, as shown in Figure 1.

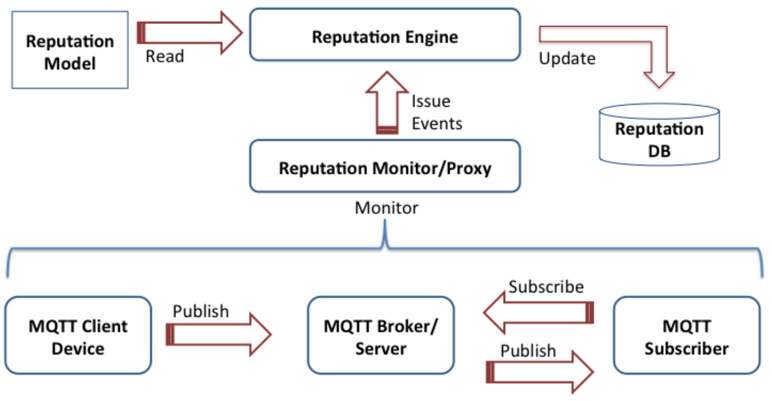

Fig. 1. The Reputation System Architecture.

The architecture defines the capabilities of the various components in an MQTT network. The reputation monitor (also sometimes referred to as the proxy) will *monitor* the MQTT interactions that take place among the MQTT network components, namely the client devices, server and subscribers. Monitoring implies that the reputation monitor will issue events to the reputation engine whenever these are required after each time it has captured an MQTT communication relevant to the utility functions predefined by the consumers (possibly the subscribers). These events could represent aggregations/abstractions of data collected from such communications, in order to minimise the additional network traffic created by this process.

Once an event has arrived at the reputation engine, it is either stored for applying further aggregations/abstractions or it is used immediately to compute new updates for the various reputation values for the clients and the server. The calculations are based on the reputation models defined in the previous sections, and the updates to these reputation values are then stored in a local reputation database. In our architecture, we only consider the monitoring problem, however, it is easy to extend this architecture in the future to include a control step, where the reputation values for different participants are then used to impact/feed back into the MQTT network communications.

5 Simulation of the Model

We implemented the architecture, described in the previous section, by running a number of off-the-shelf open source tools. First, we used the Mosquitto tool [1] as the broker (server). Mosquitto is an open source MQTT broker written in the C language that implements the MQTT protocol version 3.1. The Mosquitto project is highly portable and runs on a number of systems, which made it an effective choice for our experiments.

In order to simulate the client, we used the source code for the Eclipse Paho MQTT Python client library [2], which comes with Java MQTT client API and implements versions 3.1 and 3.1.1 of the MQTT protocol. This code provides a client class, which enables applications to connect to an MQTT broker to publish messages, receive published messages and to subscribe to topics. It also provides some helper functions to make publishing one off messages to an MQTT server very straightforward. Using this library we created a set of programs that would publish and subscribe to the Mosquitto broker. Finally, to implement the monitoring function we needed to capture all the traffic between the client and the server. For this we extended the Paho MQTT *test proxy* [2], which acts as a "reverse proxy", impersonating an MQTT server and sending all the traffic on to a real broker after capturing the messages. The proxy represents a mediator between clients and the broker. By extending this proxy we were able to trace all the packets being sent and received and send monitoring information to our reputation engine in order to calculate the reputation of the client and broker.

5.1 Results

To begin with, we assume that the network will misbehave with regards to the messages that are exchanged among the various entities in the system. This misbehaviour is modelled as the network dropping some messages according to a predefined rate (e.g. 0–100%). There could be other sources of network misbehaviour, such as the insertion of new messages and the repetition or modification of transmitted messages, however, for simplicity, we consider only the suppression of messages as our example of how the network could misbehave and how such misbehaviour would affect the reputation of MQTT clients and servers.

In our case, we chose the rate of successful message delivery to be in the range of 50% to 100%, where 50% means that one message in every two is dropped by the network, and 100% means that every message is delivered successfully to its destination. This latter case is equivalent to the normal behaviour discussed above. There are a number of tools that can drop network packets selectively. However, we created a new tool based on the above-mentioned proxy that specifically targets disrupting MQTT flows by dropping MQTT packets. The tool allowed us to target a percentage of dropped packets and therefore calculate the reputation under a given percentage of packet loss.

Since our aim is to demonstrate, in general terms, how reputation-based trust can be obtained in an IoT system such as an MQTT network, and for simplicity, we opted to consider only one source of misbehaviour, namely message suppression, without considering the other sources. Despite the fact that such sources are also interesting, they do not affect the generality of our approach.

5.2 Reputation Results

To compute the reputation value of clients and servers, we collected events related to the QoS level agreed between the client and the server throughout the 5-minute measurement window, and used the server and client reputation model proposed in Section 3.2 to calculate their reputation values. The QoS level monitoring is important as it is directly related to the issue of message suppression when messages are communicated over the unreliable network. In the presence of such abnormal behaviour, the reputation values of the clients and the server are shown in Figure 2 versus the rate of successful message delivery (0.5 to 1).

From this figure, we note that despite starting at low reputation levels in line with the low delivery rate of messages, these reputation values will increase reaching the optimal value of 1 when the rate of delivery of messages is 1. This optimal case represents the case of normal behaviour when every message is delivered successfully to its destination.

6 Related Work

Reputation is a general concept widely used in all aspects of knowledge ranging from humanities, arts and social sciences to digital sciences. In computing systems, reputation is considered as a *measure* of how trustworthy a system is. There are two approaches to trust in computer networks: the first involves a “black and white” approach based on security certificates, policies, etc. For example, SPINS [17], develops a trusted network. The second approach is probabilistic in nature, where trust is based on reputation, which is defined as a

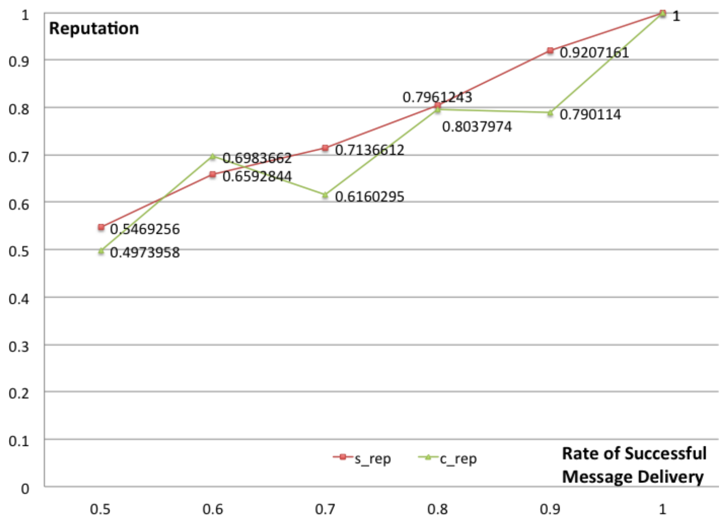

Fig. 2. Reputation Values for the Clients (c_rep) and Servers (s_rep) vs. the rate of Successful Message Delivery.

probability that an agent is trustworthy. In fact, reputation is often seen as one measure by which trust or distrust can be built based on good or bad past experiences and observations (direct trust) [14] or based on collected referral information (indirect trust) [5].

In recent years, the concept of reputation has shown itself to be useful in many areas of research in computer science, particularly in the context of distributed and collaborative systems, where interesting issues of trust and security manifest themselves. Therefore, one encounters several definitions, models and systems of reputation in distributed computing research (e.g. [12, 14, 20]).

There is considerable work into reputation and trust for wireless sensor networks, much of which is directly relevant to IoT trust and reputation. The Hermes [22] and E-Hermes [23] systems utilise Bayesian statistical methods to calculate reputation based on how effectively nodes in a mesh network propagate messages including the reputation messages. Similarly TRM-IoT [11] evaluates reputation based on the packet-forwarding trustworthiness of nodes, in this case using fuzzy logic to provide the evaluation framework. Another similar work is CORE [16] which again looks at the packet forwarding reputation of nodes.

Our approach differs from the existing research in two regards: firstly, the existing reputation models for IoT utilise the ability of nodes to operate in consort as the basis of reputation. While this is important in wireless sensor networks, there are many IoT applications that do not utilise mesh network topologies and therefore there is a need for a reputation model that supports client-server IoT protocols such as MQTT. Secondly, the work we have done evaluates the reputation of a reliable messaging system based on the number of retries needed to successfully transmit a message. Although many reputation models have been based on rates of packet forwarding, the analysis of a reliable messaging system (like MQTT with QoS > 1) is different as messages are always delivered except in catastrophic circumstances. Therefore we looked at the effort and retries required to ensure reliable delivery instead. We have not seen any similar approach to this and consider this the major contribution of the paper.

7 Conclusion

To conclude, we defined in this paper a model of reputation for IoT systems, in particular, for MQTT networks, which is based on the notion of utility functions. The model can express the reputation of client and server entities in an MQTT system at various levels, and in relation to a specific issue of interest, in our case the QoS level of the delivery of messages in the presence of a lossy network. We demonstrated that it is possible, using off-the-shelf open source MQTT tools, to implement an architecture of the reputation system that monitors the MQTT components, and we showed that the experimental results obtained from running such a system validate the theoretical model.

Future work will focus on adapting the reputation model and its architecture and implementation to other IoT standards, e.g. the Advanced Message Queuing Protocol (AMQP) [21], the Extensible Messaging and Presence Protocol (XMPP) [4], the Constrained Application Protocol (CoAP) [18] and the Simple/Streaming Text Oriented Messaging Protocol (STOMP) [3]. We also plan to consider other issues of interest when calculating reputation where satisfaction is not necessarily a binary decision, for example, the quality of data generated by client devices and the quality of any filtering, aggregation or analysis functions the server may apply to such data in order to generate new information to be delivered to the consumers. Further, we intend to apply Bayesian statistics to the results to improve the probabilistic calculation of the reputation values.

Some other interesting, though more advanced areas of research, include the strengthening of the model to be able to cope with malicious forms of client and server behaviour, for example, collusion across such entities in order to produce fake reputation values for a targeted victim, and a study on the welfare of IoT ecosystems based on the different rates of the presence of ill-behaved and well-behaved entities in the ecosystem, and how variations in the presence ratio of such entities would lead to a notion of reputation reflecting the wider ecosystem.

References

1. Mosquitto: An open source mqtt v3.1/v3.1.1 broker. `http://mosquitto.org/`, accessed: 2016-03-11
2. Paho. `http://www.eclipse.org/paho/`, accessed: 2016-03-11
3. STOMP: The Simple Text Oriented Messaging Protocol. `https://stomp.github.io`, accessed: 2016-03-11
4. XMPP Standards Foundation. `http://xmpp.org`, accessed: 2016-03-11

5. Abdul-Rahman, A., Hailes, S.: Supporting trust in virtual communities. In: HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6. IEEE Computer Society, Washington, DC, USA (2000)
6. Arenas, A.E., Aziz, B., Silaghi, G.C.: Reputation management in collaborative computing systems. Security and Communication Networks 3(6), 546–564 (2010)
7. Aziz, B., Hamilton, G.: Reputation-controlled business process workflows. In: Proceedings of the 8th International Conference on Availability, Reliability and Security. pp. 42–51. IEEE CPS (2013)
8. Aziz, B., Hamilton, G.: Enforcing reputation constraints on business process workflows. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) 5(1), 101–121 (Mar 2014)
9. Banks, A., Gupta, R.: MQTT Version 3.1.1 (2015)
10. Birman, K., Joseph, T.: Exploiting Virtual Synchrony in Distributed Systems. SIGOPS Oper. Syst. Rev. 21(5), 123–138 (Nov 1987)
11. Chen, D., Chang, G., Sun, D., Li, J., Jia, J., Wang, X.: Trm-iot: A trust management model based on fuzzy reputation for internet of things. Computer Science and Information Systems 8(4), 1207–1228 (2011)
12. Fullam, K., Barber, K.: Learning trust strategies in reputation exchange networks. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. pp. 1241–1248. ACM Press (2006)
13. Huynh, T.D., Jennings, N.R., Shadbolt, N.R.: An integrated trust and reputation model for open multi-agent systems. Autonomous Agents and Multi-Agent Systems 13(2), 119–154 (Sep 2006)
14. Jøsang, A., Ismail, R., Boyd, C.: A Survey of Trust and Reputation Systems for Online Service Provision. Decision Support Systems 43(2), 618–644 (March 2007)
15. Locke, D.: MQ Telemetry Transport (MQTT) V3.1 Protocol Specification (2010)
16. Michiardi, P., Molva, R.: Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In: Advanced Communications and Multimedia Security, pp. 107–121. Springer (2002)
17. Perrig, A., Szewczyk, R., Tygar, J., Wen, V., Culler, D.E.: Spins: Security protocols for sensor networks. Wireless networks 8(5), 521–534 (2002)
18. Shelby, Z., Hartke, K., Bormann, C.: Constrained Application Protocol (CoAP) draft-ietf-core-coap-18. `https://tools.ietf.org/html/draft-ietf-core-coap-18`, accessed: 2016-03-11
19. Silaghi, G.C., Arenas, A., Silva, L.: Reputation-based trust management systems and their applicability to grids. Tech. Rep. TR-0064, Institutes on Knowledge and Data Management & System Architecture, CoreGRID - Network of Excellence (February 2007), `http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0064.pdf`
20. Silaghi, G.C., Arenas, A., Silva, L.M.: Reputation-based trust management systems and their applicability to grids. Tech. Rep. TR-0064, Institutes on Knowledge and Data Management and System Architecture, CoreGRID - Network of Excellence (February 2007)
21. Vinoski, S.: Advanced Message Queuing Protocol. IEEE Internet Computing 10(6), 87–89 (Nov 2006)
22. Zouridaki, C., Mark, B.L., Hejmo, M., Thomas, R.K.: Hermes: A quantitative trust establishment framework for reliable data packet delivery in manets. Journal of Computer Security 15(1), 3–38 (2007)
23. Zouridaki, C., Mark, B.L., Hejmo, M., Thomas, R.K.: E-hermes: A robust cooperative trust establishment scheme for mobile ad hoc networks. Ad Hoc Networks 7(6), 1156–1168 (2009)