A formal consensus-based distributed monitoring approach for mobile IoT networks

Jose Alfredo Alvarez Aldana, Stephane Maag, Fatiha Zaïdi

Journal Pre-proof

A Formal Consensus-based Distributed Monitoring Approach for mobile IoT networks

Jose Alfredo Alvarez Aldana, Stephane Maag, Fatiha Zaidi

PII: S2542-6605(20)30183-9
DOI: https://doi.org/10.1016/j.iot.2020.100352
Reference: IOT 100352

To appear in: *Internet of Things*

Received date: 12 August 2020
Revised date: 27 November 2020
Accepted date: 15 December 2020

Please cite this article as: Jose Alfredo Alvarez Aldana, Stephane Maag, Fatiha Zaidi, A Formal Consensus-based Distributed Monitoring Approach for mobile IoT networks, *Internet of Things* (2020), doi: https://doi.org/10.1016/j.iot.2020.100352

A Formal Consensus-based Distributed Monitoring Approach for mobile IoT networks

Jose Alfredo Alvarez Aldana[a], Stephane Maag[b,*], Fatiha Zaidi[c]

[a]*SONOS, Paris, France*
[b]*Samovar, Télécom SudParis, Institut Polytechnique de Paris, France*
[c]*Université Paris-Sud, CNRS UMR LRI, Univ. Paris-Saclay, France*

Abstract

Internet of Things (IoT) represent a significant area of network research due to the many opportunities derived from the problematics and applications. The most recurring problematics are the mobility, the availability and also the limited resources. A well-known interest in networks and therefore in IoT is to monitor properties of the network and nodes [1, 2]. The problematics can have a significant impact on the monitoring efforts. Mobility and availability can create incomplete results for the monitoring. It can also represent a challenge to monitor distributed properties. The literature states that accuracy is not always reliable and difficult to achieve due to dynamic properties of the IoT in particular with M2M communications and mobile devices. Therefore we propose a distributed monitoring architecture that relies on multiple points of observation. It provides a consensus mechanism that allows it to aggregate and provides a more meaningful and accurate result. We support our proposal with numerous mathematical definitions that model local results for a single node and global results for the network. Finally, we evaluate our architecture with an emulator that relies on AWS, NS3, and Docker with varying number of nodes, network size, network density, speed, mobility algorithms and timeouts. We obtain very promising results, especially regarding accuracy.

Keywords: Consensus Algorithms, Distributed Systems, Internet of Things, Monitoring

1. Introduction

Internet of Things (IoT) represent an important area of network research since they constitute various problematics but at the same time, multiple applications [3], therefore many opportunities. Some of the most recurring problematics of IoT are the mobility, the availability and also the limited resources of each node [4]. These problematics affect multiple aspects of communications inside a network of IoT, e.g., sending messages or

*Corresponding author
Email addresses: josealfredo1515@gmail.com (Jose Alfredo Alvarez Aldana), stephane.maag@telecom-sudparis.eu (Stephane Maag), zaidi@lri.fr (Fatiha Zaidi)
URL: www.elsevier.com (Stephane Maag)

collecting data efficiently. The mobility creates network fragmentation, which affects the connectivity of the nodes. This causes two or more subnetworks and each can have their own set of information. The availability makes uncertainty about which nodes in the network we can rely on to send a message. This uncertainty creates the need for robust mechanisms that can withstand the availability of the nodes. Finally, the limited resources force us to create efficient and optimized solutions which do not sacrifice resources as computing power or communications efforts.

One prominent interests in the IoT literature is monitoring. It is highlighted in the work of Lee et al. [1] as well as in a recent paper of Raposo [2], that monitoring is of high research interest and high market interest. It is known that when a network is deployed, there is always a need to know the state of the network. The network state information allow the managers to improve, maintain, and debug the network at any moment. Therefore, the monitoring of a network is of great importance. To monitor IoT, we need to understand and overcome the challenges in place. We can look at the problems for monitoring from the mobility perspective. If a node triggers the monitoring while the network is fragmented, it will not be able to reach all the nodes. Therefore this can cause incomplete results for the monitoring purposes. Another problem is the analysis and interpretation of the data by the network. Given the mobility and the nodes availability, it can be difficult to compute a function among all nodes. Therefore each node can reply for themselves, but aggregation is needed to provide an overall result. These problems affect the monitoring of the network and the existing solutions.

Most of the known monitoring approaches in the literature state that accuracy is not reliable in a mobile IoT due to the dynamic properties. Network fragmentation has a harsh impact on the accuracy of a decentralized and distributed approach. Given that the information is spread through the network then if a subset of nodes of the network fragments, there is a probability that the information for the monitoring result is fragmented as well. Therefore, there have been many studies that try and have successfully increased and maintained the accuracy of the approaches but most of the time the accuracy decreases with the number of nodes, or density of the network [5, 6]. Besides, it has been shown in recent works [7, 8] that in such distributed systems, there is a need to monitor multiple points in the network in order to reach a consensus among the different information to have useful and processed data. This leds us to believe that a distributed system combined with a robust consensus algorithm could provide a robust IoT monitoring mechanism.

While surveying the published works in distributed monitoring, [9, 10, 6] propose interesting approaches based on hybrid categorization. However, though they enhance the accuracy of classical techniques by creating multiple root nodes and running redundant monitoring processes, there are at a middle point between decentralized and distributed approaches. Moreover, the overall accuracy is increased, but as the number of nodes in the network increases, it affects in a direct proportion the accuracy. In [11], the authors define a distributed trust monitoring technique performing an architecture divided into clusters electing monitor heads. Although they shown a low message overhead, the election methods are inefficient compared to the leader election of a consensus algorithm. Concerning the checking of global properties in a distributed manner, we may cite works based on publish/subscribe principles such as [12, 13]. The concepts are relevant but the middleware in use is too heavy for monitoring IoT networks in a fully distributed way. In [14], the authors prevent such drawback by distributing in a more efficient manner

the events on the network. This approach needs to deploy a PICO-MP network. The numbers of messages can be drastically reduced, but no information about the accuracy of the global properties checking is provided. In our approach, we focus on a lightweight approach based on traces analysis and on a consensus protocol to reduce the number of messages and specifically on the verdict accuracy. We took these works as an inspiration bedrock and started to build a proposal that tackles the problematics for monitoring IoT. This work is part of the unpublished contributions, related to distributed mechanisms to monitor networks, of the PhD thesis [15]. We summarize the main contributions of this paper in the following:

1. The proposal of a mathematical definition that models local results for a single node and global results for a group of nodes or the whole network. These results are the responses to a query definition that is propagated to the network. Finally how these are integrated into a global result definition. These definitions help us to provide a strong foundation for our monitoring architecture.
2. The proposal of a distributed monitoring architecture that relies on multiple points of observations being the nodes. It provides a consensus mechanism that allows the architecture to aggregate and provide a more meaningful result by analyzing the data from a bigger perspective. Along side, we provide an appropriated mechanism to free stored results to avoid resource consumption problems.
3. Finally we evaluate our result in a network emulator built in-house which combines Amazon Web Services, Docker, and NS3. This emulator allowed us to run thousands of emulations varying parameters like the number of nodes, network size, node density, speed, mobility pattern and timeouts. As a result, we were able to generate data and make an in-depth analysis of it by looking at average, maximum, minimum and standard deviation values for accuracy, convergence time, number and size of packets.

The remaining of our paper is as it follows. In Section 2, we discuss distributed monitoring and consensus algorithm, and how that influence our architecture. In Section 3, in the first half, we define our mathematical models for the local and global result. Also, in the second half, we define our architecture based on these models. In Section 4, we present our implementation, we go through many aspects that were involved in the implementation. We discuss the emulator, and finally, we go through the thousands of emulations we performed. Next, in Section 5, we present some interesting related works from which we got inspired. Finally, we conclude and give perspectives in Section 6.

2. Preliminaries

Network monitoring is an extensive field of interest. It can be described as "a number of observers making observations and wish to work together to compute a function of the combination of all their observations" [16]. The goal is that all observers (network nodes) compute a value $f(t)$ in a given instant of time t in a collaborative way. For our purposes, the function can be simple or complex.We define a simple function as a relation where the domain depends only on the local data of a node, e.g. average CPU of all nodes or average memory usage of all nodes. On the other side, we define a complex function as a relation where the domain depends on distributed data among a set of nodes, e.g. the integrity of the communication by checking the checksum of packets in all nodes.

2.1. Distributed monitoring

The classification of the monitoring process has been studied in [17, 18]. For the purposes of this paper, we consider the one major type: distributed. As stated by [18], the problem of distributed monitoring can have various "trivial" solutions, e.g., centralizing the information. The centralized type of monitoring is when all the nodes report their observations to a central entity, named centralizer or coordinator. The distributed approach deals with networks where there is no centralized entity and the data or computing of it is distributed across all nodes. When the property to monitor is simple, it can be easier to achieve a global view. However, when the property to monitor is a complex, which are dependent on more information and non-deterministic, the solutions also tend to be more complex. A solution for approaches that need to be considered by simple and complex properties is defined as the geometric approach by [19]. Sharfman et al. state that a global view can be achieved by breaking down the function into properties or conditions that can be checked locally. The work of Goodloe et al. [20], states that another important aspect of distributed monitoring is the ability to be fault tolerant. According to this work, they state that some systems omit mechanisms to ensure that the network agrees in one result based on the fact that the probability of occurrence of a bad result is small. However, this is a wrong assumption and might incur into an incomplete mechanism to deal with fault tolerant nodes. This derives the need to accommodate a consensus mechanism to provide a Byzantine or asymmetric fault tolerance.

2.2. Consensus

The consensus algorithm is derived from the Byzantine generals problem proposed by Lamport et al. [21]. They state that a reliable computer system must be able to work upon the existence of malfunctioning parts which can be generating conflicting information. There has to be a majority of the parts functioning, which is known as Byzantine fault tolerance. For the purposes of this paper, we assume the original fault tolerance model, where the majority is defined as 2/3 of the parts of the system. From this, we can say that a consensus algorithm is a mechanism to allow different nodes in a network to agree on some information and work as a coherent group despite the failures of some of the nodes [22]. The study of Fischer et al. [23], states that a Byzantine failure can be a faulty process that can send messages when it is not supposed to. This means, sending conflicting information that oppose the general view of the system. One of the most known consensus protocols is Paxos [24]. Although this protocol is considerably complicated and difficult to integrate, it became the foundation for consensus algorithms. Although the complications stated by Paxos, it was still implemented by multiple big tech companies in services like Amazon S3, Amazon DynamoDB, and Apache ZooKeeper. There are other companies like Google that developed their consensus algorithm [25]. In the recent years, a novel approach was proposed based on the limitations of Paxos, this proposal is Raft [26]. Companies like Hashicorp, support this new approach and it has gained enough popularity to have implementations in most of the programming languages like C, Java, GoLang, Python and more. The Raft protocol is considered to be better than Paxos since it decouples the leader election, log replication and safety, thus making it more modular. Moreover, it reduces the degree of non-determinism among servers of Paxos by defining clear and consistent rules for the communication between servers.

2.3. Classical flooding

The transmission of data in a network is not only important but it can also be considered as an energy bottleneck, specially in IoT networks. For our approach, the communication sent by any node is intended to reach every node for availability purposes. Therefore, we selected a classical flooding [27] mechanism to transmit data over the network. Any node that wants to share any information with the network will start this process by broadcasting the message to its nearest neighbours. Then, each node receiving a broadcasted message checks if it has received the message in the past by relying on a message ID. If the message has not been received in the past, it broadcast it to the nearest neighbours, otherwise it discards the message. Although it can be argued that this mechanism has some flaws, the intention of our paper is not to deep dive in the information dissemination algorithm. On the other side, we consider that it aligns with the requirement of low energy consumption of an IoT. Indeed, as it has been shown by Haley et. al. [28], the energy consumed by a node in transmission is much greater than in reception. Although network technologies drastically evolved, energy consumption in IoT has been studied in a very detailed and recent survey [29]. The authors analyse and raise several solutions dedicated to energy consumed by nodes and particularly to power management in IoT. They expose that the solutions to set up depend on several parameters (e.g., OS, applications, etc.). Although in our paper we do not cope with these aspects, this survey as well as the very recent complete works [30, 31] will be very good and strong starting points for future works.

3. A fully distributed monitoring architecture

We propose a fully distributed architecture by relying on a distributed monitoring record and a consensus algorithm. Through this, we intend to provide a mechanism capable of monitoring a network with multiple probes and multiples points of observation. Our approach relies on a protocol we define to support the multiple processes of the network. The protocol depends on a packet that we have defined. Each packet has two parts, a header, and a body. We define four different type of packets: query, transaction, consensus, and block. As a preliminary definition, the query packet is generated by a node or external entity. This packet is sent to the network. Then each node analyzes the query and produces a transaction. The transaction packet is the reply per node of the query, and as the name implies, it contains a transaction. A transaction is defined as the result of the verification of a local property, which is defined in the following sections. The consensus packet is generated to coordinate the election of the validator or consenser based on the consensus algorithm rules. Based on this consensus mechanism, the leader validates a global property based on all local properties generated by the query. This generates a result, which is assembled into a block. The block packet is the final result of a query, which contains a group of transactions and the results based on the query. To adequately define each part of our architecture, we must first define the messages, events, local properties and global properties. For this formalization, we get inspired by works performed in the context of services orchestration [32] and SIP protocol [33].

3.1. Single message

The network is interchanging and routing information through packets. This means that a node A is sending information to node Z. The path between both nodes is changing

due to mobility and availability in an IoT network. Therefore, it means that a packet between these two nodes can take any finite amount of hops. Each node is capable of taking some actions based on the packets which generates a routing event. From these, we define a set of basic actions that can be performed by every node for the routing of information.

Definition 1 (Actions). *An action is the following final set of literals :*

$$\Omega ::= New \mid Income \mid Outcome \mid Received$$

Where New refers to the creation of a new message, Income refers to the reception of an incoming message, Outcome to the transmission of an outgoing message and Received to the acknowledgement of a received message.

It is worth mentioning that any other grammar defining other actions for any other protocol is possible. Keep in mind that how the routing algorithm decides the next hop is out of the scope of this paper. We assume that this is defined using the embedded routing protocol and each node will know what to do up to the reception of a packet.

Given the set of action names (Ω), we can notice that more information is needed to model an actual message. For example the destination of the message, next hop or previous hop, an even more advanced characteristics as status, checksum, protocol, etc. Therefore relying on the grammar for the actions, we can define a message.

Definition 2 (Message). *Given a finite set of action names Ω, of labels $\mathcal{L}$, and of atomic data values $\mathcal{D}$, a message m takes the following form:*

$$m = o(l_1 = v_1, \ldots, l_n = v_n)$$

where $o \in \Omega$ represents the action. The composite data of the message is represented by a set $\{l_1 = v_1, \ldots, l_n = v_n\}$, rewritten as $(\bar{l} = \bar{v})$ for short, in which each field of this data structure is pointed by a label $l_i \in \mathcal{L}$ and its value is $v_i \in \mathcal{D}$.

In this paper, we consider $\mathcal{L}$ as any field name of a message (*e.g.*, a routing packet with the fields destination, checksum, etc.) and $\mathcal{D} = integer \cup string$.

Example 1. *An Income message is formulated as the following:*
m1 = `Income` *(previousHop = "10.20.5.5",*
actualHop = "10.20.5.7", tag = 127,
protocol = "ospf", timestamp = 1520870383,
destination = "10.20.5.50")
where "10.20.5.7" is the IPv4 address of node M, "10.20.5.50" the IPv4 address of node Z and "10.20.5.5" and "10.20.5.9" are the two IPv4 addresses of two intermediary nodes between the package in transit from node A and node Z. The timestamp, protocol and tag properties are for exemplification purposes.

3.2. Messages under observation

From the previous definitions, we say that a message is an instance of an event. Meaning that a message is the single occurrence of an event. In our case, an event is a routing event, *e.g.*, the creation of a packet, the reception or transmission of a packet, and so on. Due to our monitoring purposes, these events are under evaluation. We, therefore, define a candidate event, which is a pair constituted by an event e and a predicate ϕ. This represents a message or set of messages which are an instance of e that are satisfied by ϕ. To define a candidate event, we need to first define a predicate and a term.

Definition 3 (Predicate). *A predicate ϕ is defined wrt. the following:*

$$\phi ::= \neg\phi \mid \phi_1 \wedge \phi_2 \mid t_1 = t_2 \mid t_1 > t_2$$

where t1 and t2 are terms, a term being either a constant or a variable, e.g., $t ::= v \mid x$ where v is an atomic value and x is a variable.

Definition 4 (Predicate Dependency). *We tell that a predicate ϕ_j is dependent of a predicate ϕ_i, noted $\phi_j \sim \phi_i$, if ϕ_j is expressed using variables defined by ϕ_i.*

Definition 5 (Candidate Event). *A candidate event (CE) is a pair $o(l_1 = x_1, \ldots, l_n = x_n)/\phi(x_1, \ldots, x_n)$, denoted by $(\bar{x})/\phi(\bar{x})$, where $o(l_1 = x_1, \ldots, l_n = x_n)$ is a message and $\phi(x_1, \ldots, x_n)$ is a predicate. The predicate can be omitted if it is true.*

Example 2. *To define the event of the start transmission of a message, we write: `New(source=x)/(x="10.20.5.1")` which represents any `New` messages whose source is "`10.20.5.1`".*

Through these definitions we are able to formalize any routing event with their respective properties and more over to represent it using a candidate event for monitoring purposes. It is important to point out that a CE represents a set of messages. From these we can proceed to define local and global properties and further operations.

3.3. Local property

Each node in a network participates in the network operations like routing packets. Not necessarily all nodes and not at all times, this depends on mobility and many other factors. Nonetheless, based on these interactions, each node will generate its own log of information. This information gives each node a unique local perspective of the network. This local perspective is defined through local properties. Therefore, the evaluation of a set of local properties derives a local view of the node. This evaluation can derive an empty set as a result for some nodes, which means that the node cannot satisfy the given local properties. We define a local property $\mathcal{P}$, which is used to represent the behavior to be monitored in a node. In our specific case, these properties are validated against the given logs for the routing information of a node. The idea is that a local property can be evaluated using any data that can be represented as a finite stream of messages. The

local property is expressed as a combination of a conditional statement and a control flow statement. This means that if the context is satisfied therefore a consequence must exist for the given context, *e.g.*, IF context THEN consequence. For example, let us look at Example 1, the logic is if a node in the middle of the routing of a packet destined to another node, it should receive an *Income* message and then it should forward an *Outcome* message. Moreover, for the control flow statement, it means that, given a set or sets of messages, it will iterate performing a predefined operation. The evaluation of the CEs and its result, a set of messages, is explained in detail in the following sections. We proceed to formalize the local property as following:

Definition 6 (Local Property). *A local property $\mathcal{P}$ is a 3-tuple (*cont*, *conseq*, *op*) composed of a Context* cont *AND/OR a Consequence* conseq *AND/OR an Operation* op*. It is denoted as:*

$$\mathcal{P} ::= cont \rightarrow conseq \ \times \ op$$

where:

- *Context is a sequence of CEs, $\langle e_1/\phi_1, e_2/\phi_2, \ldots, e_n/\phi_n \rangle$ where it may exist pairs of indexes $i, j \in \{1, \ldots n\}$, $i < j$ such that $\phi_j \sim \phi_i$.*
- *Consequence is a set of CEs, e.g., $\{CE_1, \ldots, CE_m\}$. Therefore, according to a Context, a Consequence can or can not be validated. Note that a Consequence can be empty. In that case, the property just focuses on the Operation* op*.*
- *Operation is an operator that describes the operation followed by a term or terms, e.g., $(exist|sum|avg|count|*)$.*

  An Operation is a function that considers the terms of cont *if* conseq*$= \emptyset$ or all terms in* cont*$\rightarrow$*conseq *if* op*$\neq \emptyset$.*

  Note that op *is not mandatory, it may be unnecessary for some properties. If* op *is omitted, the result of the validation of $\mathcal{P}$ is a boolean.*

Example 3. *Let us look at the case where a node in the middle of the routing of a packet is destined to another node. If a node receives a message which is destined to another node, it should forward it to the next hop defined by the routing algorithm. Therefore for a node different from the source or the destination, it should be true that if there is an Income message to node Z, there should exists an Outcome message to node Z. This local property is defined as follows:*

$$\begin{aligned}\mathcal{P}_1 \ = \ & \langle \texttt{Income}(destination = x_1) \\ & /(x_1 == \text{“}\texttt{10.20.5.50}'')\rangle \\ & \rightarrow \{\texttt{Outcome}(destination = x_2) \\ & /(x_2 == \text{“}\texttt{10.20.5.50}'')\}\end{aligned}$$

3.4. Global property

Given the distributed nature of our architecture, it is important to note that each node holds its own perspective from its local information. However, some of the data hold locally from some nodes, intersects with other local data of another set of nodes. This intersection of local information gives a more broad view of the network. Therefore we say that there is a network perspective based on the aggregation of the local perspectives. This means that by relying on local perspectives, it can be determined a global view of a property of the network. A global view provides more general information since it aggregates over local views. For example, if we want to model if node A created a message to node B or if we want to model that node B received a message that was originated from node A, both of this cases can be modeled through a local property. However, if we want to ensure that the integrity of the message generated from node A to node B was maintained through the whole routing process, we need more information than just local views. For this specific example, we could verify that the data checksum generated by the source node A is the same as the data checksum by the receiving node Z. We are assuming that the field checksum of a packet is defined as the result of calculating a hash function of the body of the packet. This would give us a degree of certainty that the data generated was the data received but would not ensure that the integrity was maintained through the whole process. Therefore, we would need all the nodes involved in the communication between node A to node Z and validate the checksum of each to verify that the integrity is maintained. This gives us a complete certainty of the integrity for a given packet.For this kind of scenarios, where the verification needs two or more point of observations, we need to define a global property. The global property is also expressed as a conditional statement followed by a control flow statement. This means that if a context is satisfied therefore a consequence must exist for the given context. This property is defined on top of local properties. Therefore we define a global property as follows:

Definition 7 (Global Property). *A global property* $\boldsymbol{G}$ *is a 3-tuple (SET, SET', op) denoted as:*

$$\boldsymbol{G} ::= SET \Rightarrow SET' \ \times \ op$$

where SET *and* SET' *are two sets of local properties and* op *is an Operator as defined for local properties.*

Example 4. *Let us look at a more complex case where an operator is present. Since we work in the context of IoT, we might need to measure the average transmission time of the messages sent. For this, we would need to measure the delta of time between the time when a message was sent and the time when the message was received by the next hop. Therefore, this metric cannot be calculated by single local view, but it can only been measured by the global view. For the purposes of this example, we assume that the packet contains the address of the actual hop with the label "hop" and of the previous hop with the label "prevHop". Therefore, it is defined as:*

$$
\begin{aligned}
\mathcal{P}_1 &= \langle \texttt{Outcome}(id = x_1)/(x_1 == 1234)\rangle \\
\mathcal{P}_2 &= \langle \texttt{Income}(id = x_2)/(x_2 == 1234)\rangle \\
\boldsymbol{G}_1 &= \{\ \mathcal{P}_1,\ \mathcal{P}_2\ \} \Rightarrow \{ \\
&\qquad \langle \texttt{Income}(prevHop = y_1, timestamp = y_2)\rangle \\
&\qquad \rightarrow \{\texttt{Outcome}(hop = y_3, timestamp = y_4) \\
&\qquad /(y_1 == y_3)\}\} \\
&\quad \times \{average\ (y_2 - y_4)\}
\end{aligned}
$$

3.5. Query process

Our approach is reactive to a query generated by a node in the network or by an external request. The query is assumed to request information for monitoring objectives from all the nodes in the network. Therefore the query is generated and then all the network computes a result. When each node receives a query, it evaluates it based on the criteria given to get a result. If the node has a result, it then generates a transaction packet and it is sent to the network. The transaction packet is explained in detail in Section 3.6. The result generated by a node is considered a partial response to the query given that it only takes into account its local view. The query can rely on local and global properties, not necessarily both. The query packet has a body formalized as follows:

Definition 8 (Query). *The body of the query packet is defined by the following tuple:*

$$Query = [G_{PROP},\ T_o,\ T_r]$$

where G_{PROP} is a global property, T_o is a timeout defined in milliseconds by an integer and T_r is an optional number of transactions defined as any number greater than zero as an integer.

The idea is that a query is transmitted to the network. Then evaluated by each node to create local views and later to calculate the global view and final result of the query. The evaluation of the query by a single node, can derive a local view which is encapsulated by our approach as a transaction. We are assuming that each query contains only one global property. If there is need for more global properties, each one of them should be executed using different queries. The query integrates one required and one optional parameter, which are a timeout and a number of transactions. Every query operates over an IoT network, then it has to overcome availability, mobility, fragmentation and other inherent properties of these networks. Therefore, we provide some parameters to ensure a deterministic approach of completing a query. The timeout, as the name implies, is just a time to wait and afterwards all the existing transactions are aggregated. The number of transactions, if present, is a limit on the quantity of existing transactions for the given query, therefore until this threshold is met the query is not complete. If both parameters are present, then the query will be completed when one of them is satisfied. The behavior of a node upon the reception of a query is depicted in Algorithm 1. The first step is to trigger the procedure that waits for the timeout of the query (line 3), which is run asynchronously or in parallel. Then, it iterates over the local properties of a query (line 4). For each one of them, it first evaluates the local property and the set of the node local traces (line 5). The description of the function that evaluates the local property can be found in Section 3.6 in Algorithm 3. If the result is not empty (line 7) then it creates and sends the transaction (line 8).

Algorithm 1 Reception of a Query in a node

```
1: procedure receiveQuery(Q)
2:     flood(Q)
3:     waitTimeout(Q)                          ▷ This is run asynchronously
4:     for each Q.LPs as LP do
5:         result[] ← evalLP(LP, myTraces)
6:     end for
7:     if result is not empty then
8:         sendTransaction(LP, result, myTraces′)
9:     end if
10: end procedure
```

3.6. Transaction process

Upon the reception of the query, the nodes need to parse it and evaluate it. Relying on our previous definitions, if the query is composed by a global property, this means that it contains a set of local properties. As we mentioned before, the global property is only evaluated when there exists a result of multiple local properties. Therefore, a transaction corresponds to the result of evaluating a local property. From this evaluation, a result is derived, which is added to a transaction packet and sent to the network. If the result is different than null, a node can generate only one transaction per query. The contrary case when the result is null, meaning that the node does not contain any information that can suffice the local property, then there is no transaction generated by the node. So before we define the actual transaction packet, we need to define the evaluation of a local property. A global property contains one or more local properties. Each local property contains one or more candidate events. Therefore, to define the evaluation of the query, we need to evaluate all local properties and by transitivity all candidate events of them. However, to evaluate the smallest of the entities, we need a set of data to rely on. For this, we define the data source as a trace segment, which is defined as $\gamma = \langle m_1, \ldots, m_k \rangle$, a list of messages. The traces are instantiated traces coming from the logs of the nodes. This is explained in detail in Section 4.3.2. We first define the evaluation of a candidate event with Algorithm 2.

This algorithm defines two sections, a projection and a selection. The first section (line 2) defines a projection of a candidate event (CE) from the trace segment. This operation projects all traces that contain the corresponding message and field names required by it. We check that the message name is the same and that the list of field names and values corresponds as well. Then, if a message contains the required criteria, it is considered as part of the projection and is added to a set of traces.

The second section defines a selection from the projected traces. To define this selection (line 3), the predicate of the CE is evaluated. By definition of a predicate, we only have boolean operators, the result is boolean as well ($true, false$). This means that the operation selects the messages that comply with the predicate. Finally, the result from both sections is returned. Let us note that the result contains a subset of the original trace segment that satisfies one candidate event. The number of its elements can be zero up to the number of elements of the original trace segment. This will depend on the specification of the candidate event.

Algorithm 2 Evaluation of a CE in a list of traces

Precondition: Π refers to the projection operation and σ refers to the selection operation from relation algebra.

1: **procedure** EVALCE(CE, $Traces$)
2: $projection \leftarrow \Pi_{CE.o,CE.l_i}(Traces)$
3: $selection \leftarrow \sigma_{\phi(v,x)}(projection)$
4: **return** *selection*
5: **end procedure**

From this, we can define the evaluation of local property by considering an algorithm that analyzes the context, the consequence, the operation and then their results in a conditional statement. For that, we define the Algorithm 3. The algorithm has three sections. The first one (line 2) is where a subset of traces are derived from the context. The second part (line 6) is where a subset of traces are derived for the consequence. Finally, we evaluate the operation of the local property (line 10). First, we evaluate that if an operation exists, then we apply the given operation to the list of messages. Otherwise, we fall back to the default behavior (line 13) and return the result of the conjunction that for every context it should exist a consequence.

Algorithm 3 Evaluation of a Local Property in a list of traces

1: **procedure** EVALLP(LP, $Traces$)
2: $evalcontext \leftarrow$ map()
3: **for each** $LP.Context.CEs$ as CE **do**
4: $evalcontext[] \leftarrow$ EVALCE($CE, Traces$)
5: **end for**
6: $evalconsequence \leftarrow$ map()
7: **for each** $LP.Consequence.CEs$ as CE **do**
8: $evalconsequence[] \leftarrow$ EVALCE($CE, Traces$)
9: **end for**
10: **if** $LP.Operation$ is not empty **then**
11: **return** $LP.Operation^{n}_{i=1}(evalcontext[i],\ evalconsequence[i])$
12: **end if**
13: **return** $\bigwedge_{i=1}^{n} evalcontext[i] \exists evalconsequence[i]$
14: **end procedure**

With these two algorithms, we can define the transaction packet as the tuple defined as a property, the evaluation of that property and the traces that satisfy the local property.

Definition 9 (Transaction). *The body of the transaction packet is defined as follows:*

$$Transaction = [\ P, evalLP(P, \Gamma_{local}),\ \Gamma'_{local}\ ]$$

where P is a local property, $evalLP(P, \Gamma_{local})$ refers to the result of the evaluation of P through the set of traces Γ_{local}, and $\Gamma'_{local} \subset \Gamma_{local}$ represents the traces that were effective

for the evaluation of P (i.e., the above-mentioned subset, indeed, not all traces from the list are locally usable or useful for evaluating the formula, this can depends on the node processing the property evaluation). Note that $evalLP(P, \Gamma_{local})$ can be either boolean, integer, double or any other data type defined by the Operation of the local property.

It is important to mention that, if the evaluation of the local property returns false, then no transaction packet is generated. But if a packet is generated, then this packet is sent to the network. The behavior of a node upon the reception of a transaction is depicted in Algorithm 4. This algorithm starts by sending the transaction to ensure the transmission of it (line 2. Then it stores the transaction (line 3. Followed by getting the corresponding query (line 4). Finally it checks if there is enough transactions based on the parameters of the query (line 5 and if this is satisfied, then it triggers the consensus process (line 6).

Algorithm 4 Reception of a Transaction by a node

```
1: procedure receiveTransaction(T)
2:     flood(T)
3:     transactions[] ← T
4:     Q ← get query for T
5:     if Q.T_r exists and is >= than Ts of Q then
6:         triggerConsensus
7:     end if
8: end procedure
```

3.7. Consensus process

Our approach aims to get the responses of the nodes to the query, but taking advantage of the distributed nature of the approach, also to compute a global result based on these responses to provide a more robust conclusion. We propose to utilize a consensus algorithm as a mechanism for grouping the transactions, validating them on a higher level and finally providing the complete result of the query to the network (Fig. 1). Every time a response to a query is generated by a node to the network, transaction, the nodes verify if the parameters (timeout and/or the number of transactions) of the query completion are met. If the conditions are met, then the consensus algorithm is triggered to elect a validator. Therefore this entity groups the transactions of the query and then it validates the global property. Finally, a block is assembled with the group of transactions and the results of the validation, and it is sent to the network. After this, the elected leader goes back to a sleep state, and all the nodes stop any consensus activity until it is again needed. When it is needed again, a new election is held, and a new leader is elected. This is an essential factor of our approach. Given the properties of the IoT, it is crucial that, at each round, any node can act as the validator. This makes the approach more dynamic, suitable for change and scalable. From the data perspective, it also makes it more democratized, fair and distributed, since all nodes can and would participate in the validation of the monitoring information.

Relying on the definitions made by Raft [26], we define our leader election algorithm illustrated as an input-output finite state machine (Figure 1). After this, the leader

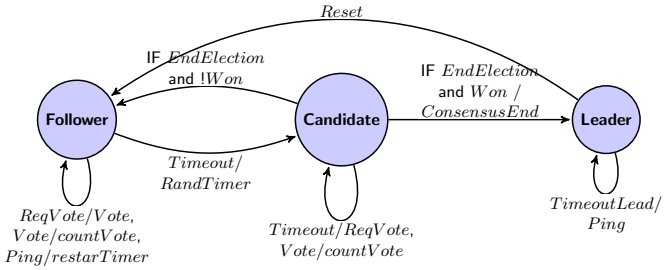


Figure 1: IO FSM definition of the raft-inspired leader election

evaluates the global property based on the transactions containing the local views. This generates a block packet that is sent to the network, which is explained in detail in Section 3.8. Finally, the consensus is completed when the network verifies the result.

Any node participating in the election can be in three different states: Follower, Candidate or Leader. The Follower state is the most common state, and every node starts in this state. Aside from that, if they are requested to vote for a leader, they do so one time, and they vote for the one that they were requested first, therefore based by time and not by any other preference. Otherwise, all nodes vote, and at the same time they participate as watchers in the election. Therefore they keep track of the votes they hear aside from theirs. This gives the ability to the network to be sure that the leader is effectively the elected leader. The Candidate state is for a postulating node to be a leader. Therefore the first thing they do is to do a random timeout and when that timeout is reached they request for votes. If they hear a vote either for them or for another participating node, they count it nonetheless. If they receive a request for vote, they will fall back to the Follower state and vote for that candidate. It is important to note that, if two or more candidates clash with the same random time, and they request votes at the same time, they all return to the Follower state and after reaching a timeout a new election is held. By the end of the election, defined as a particular timeout, the node with the majority of nodes is the winner. Then that node announces itself as the leader, switches into the Leader state, and it closes the election. From this moment, the leader pings periodically the followers so they know of its existence. If the leader goes out of range or goes offline, another election will be held. Once a leader is elected, this node triggers the consensus end, giving place to the creation of the block and the completion of a query. However, once the block is created and transmitted, it resets everything and remains in a sleep state until it is needed again. Therefore, if there is no need for a leader, there is also no need for the exchange of messages of the given election leader protocol.

3.8. Block process

All the nodes in the network contain the queries and the transactions generated by the queries. Based on the query restrictions and the transactions, every node is capable of deciding when to aggregate the data. This is referred in Algorithm 4 and Algorithm 1. Therefore, when these criteria are met, all nodes employ a consensus algorithm to dictate a node to perform this task. The temporal leader node takes all the transactions it has from a given query, aggregates them and analyzes the global property. To determine the evaluation of a global property, we rely on the definition of the evaluation of the local

property. The Algorithm 5 defines the evaluation of a global property. It is a recursion over the same concept on how to evaluate a local property. It has three parts, (i) getting the traces of the context, (ii) the traces of the consequence and (iii) returning the result of the operation. This process yields a result consolidated and sent as a block packet.

Algorithm 5 Evaluation of a Global Property in a list of traces

```
1:  procedure evalGlobalProperty(GP, Traces)
2:      context ← map()
3:      for each GP.Context.LPs as LP do
4:          context[] ← traces of evalLP(LP, Traces)
5:      end for
6:      consequence ← map()
7:      for each GP.Consequence.LPs as LP do
8:          consequence[] ← traces of evalLP(LP, Traces)
9:      end for
10:     if GP.Operation is not empty then
11:         return GP.Operation_{i=1}^{n}(context[i], consequence[i])
12:     end if
13:     return ⋀_{i=1}^{n} context[i] ∃ consequence[i]
14: end procedure
```

Definition 10 (Block). *The body of the block packet is defined as follows:*

$$Block ::= [\ G,\ evalGP(G, \gamma_{local_i}),\ \gamma_{local_i}\ ]$$

where G is a global property, γ_{local_i} is a set of traces from the local properties of G and $evalGP(G, \gamma_{local_i})$ refers to the result of evaluation the global property in all the local trace segments.

As mentioned before, once the consensus algorithm elects a leader, the aggregation process is done. This process consolidates the query and the transactions of the query. Then the global property is evaluated as explained before. Finally, the block is consolidated, then sent to the network and added to the ledger.

The behavior of a node upon the reception of a block is depicted in Algorithm 6. The other nodes, upon the reception of the block, verify the block packet to make sure that the information is congruent. The verification is done by evaluating the query with the data that the node has. This should have the same result as the one transmitted in the block (line 3). If the block passes this second verification (line 4), then it is added to the nodes ledger (line 5) and then sent to the network. Otherwise, the block is rejected by the node and announced to the network (line 8). With these procedures, we can define the block packet, as the tuple with a global property, the evaluation of the global property and the traces that satisfy the global property. It is important to mention that, given the inherent properties of IoT, there can be incongruent scenarios for some nodes. Let us look at the case of network fragmentation, so all nodes have the same information, but one of the nodes in the network went off range for a period of time, and it has one or more missing transactions. This means that when this node tries to verify the block,

it will fail to do so. In this case, each node has to keep track of what the other nodes are sending, and if the majority of the network approves a block that was rejected, then the node should accept it. Another case along the same line is if a node verified a block, but it hears an error for the same block from another node. Once again, the majority of the network is what defines the outcome. It is worth mentioning that, if a node verifies a block, but it receives that the majority of the network has acknowledged the block as an error, then the block should be discarded. This case could happen due to network fragmentation, but at the time that a node sent its transaction to the network, it was off the range. Therefore most nodes had a missing part. Therefore, for simplicity purposes, each node agrees with what the majority of the network agrees.

Algorithm 6 Reception of a Block in a node

```
1: procedure receiveBlock(B)
2:     T ← get Transactions for B.Q
3:     testResult ← evalGlobalProp(B.Q, ⋃T.traces)
4:     if testResult == B.result then
5:         addToChain(B)
6:         flood(B)
7:     else
8:         sendError
9:     end if
10: end procedure
```

3.9. Clean and Restart processes

To avoid resource consumption, specifically storage space for each node, there is a process to free resources from past queries. This is a restart process, and this takes care of cleaning storage of all the blocks from a specific query, specific set of queries or all queries. Therefore, a query or multiple queries can be performed at any given point in time. Once there is enough information, this is defined by the node or external entity generating the query, it can be asked to the network to discard all blocks from these queries. Our approach intends that multiple queries can be performed in a defined and finite amount of time and that all these results are stored for enhancing the overall result. However, in the long term this can become a storage problem. Therefore the idea is to avoid these problems with this process.

4. Experiments

We evaluate our proposal using an emulator[1] built in-house based on Dockemu 2.0 [34, 35]. This tool, based on container virtualization, allow to develop applications on current operating systems which can be tested on larger networks, leading to accurate results and saving time in duplicity of an implementation. Besides, it relies on Docker (v18.03.1-ce) and NS3 (v3.26), well accepted tools that guarantee the accuracy of our

[1] https://github.com/chepeftw/NS3DockerEmulator

results. It sits on top of an orchestrator that relies on Amazon Web Services Elastic Cloud Computing (AWS EC2) instances to launch parallel emulations for fast and efficient generation of results. This means that each arrangement of parameters (number of nodes, network size, node speed, timeout and mobility model) is emulated on its own AWS EC2 instance, which it is running parallel to many other emulations. This enables us to improve the overall throughput for our experiments, giving us more data in less time. Finally, the orchestrator parses the logs from the Docker containers per emulation, extracts the information and centralizes it in a MongoDB server for further analysis. Together they provide a highly scalable, replicable and robust environment to conduct experiments. The testbed consisted in an implementation separated in four repositories of our proposal in the language Go (v1.9). Each repository represents a layer of the complete implementation, it is separated into ledger[2], miner[3], router[4] and raft[5]. The implementation is separated into different layers to isolate the different processes and to allow parallel execution of all the tasks without any problem. The ledger layer refers to the entity in charge of storing the chain of blocks, generating the transactions and validating blocks. The miner layer refers to the entity of starting the election process, and upon the reception of the results of the election, it creates the block for a given query. The router layer refers to the entity that handles the packets from the network and sent it to the corresponding layer, also performs the flooding when it needs to send information to the network. Finally, the raft layer refers to the entity in charge of electing a leader. The idea is to identify primarily the completion time per query, the accuracy of the approach, the traffic it generates and how it compares to other approaches.

The completion time per query is defined as the time it takes from the generation of the query to the official result of the query represented as a block. This includes the time it takes to propagate the query, analyze it per node, generating the transactions, the consensus algorithm, the block creation and finally the verification of the result. The accuracy of our approach is defined by the validity of the blocks generated from the consensus algorithm. This means that every time a node receives a block, it validates the information of it and determines if the block is valid or not. Then this information is used to determine that for each query, the overall process is valid or not, and how many nodes agreed on this. It is essential to measure this since mobile IoT networks are prone to fragmentation due to mobility. Therefore, it exists a need to determine the accuracy of the approach despite fragmentation or other IoT problematics. The traffic generated by our approach tells us how many packets are being generated and transmitted over the network. Ideally, it should transmit few packages for resource efficiency, but problems like fragmentation tend to make this difficult. Finally, we compare our approach to other approaches in the literature to determine the pros and cons of the approach at hand. The following is a summary of the previous definitions for easier lecture.

[2]https://github.com/chepeftw/LedgerMANET
[3]https://github.com/chepeftw/MinerMANET
[4]https://github.com/chepeftw/RouterLedgerMANET
[5]https://github.com/chepeftw/raft

$$
\begin{aligned}
&\Omega ::= New \mid Income \mid Outcome \mid Received \\
&m = o(l_1 = v_1, \ldots, l_n = v_n) \\
&\phi ::= \neg\phi \mid \phi_1 \wedge \phi_2 \mid t_1 = t_2 \mid t_1 > t_2 \\
&o(l_1 = x_1, \ldots, l_n = x_n)/\phi(x_1, \ldots, x_n) \\
&\mathcal{P} \ ::= cont \rightarrow conseq \ \times \ op \\
&\mathbf{G} \ ::= SET \Rightarrow SET' \ \times \ op
\end{aligned}
$$

4.1. Test case and monitored properties

We propose a test case where there is an IoT network in a defined space with a fixed number of nodes. A node from the network, chosen randomly, triggers a query and is sent to the network. The query we defined it to determine the integrity of packet transmission with a $T_o = 60000$ and $T_r = 4$. We chose this test case, since the monitoring needs information for every participating node in the communication. We assume that the field checksum was properly calculated at the time of the transmission. We check that the checksum is maintained through the whole communication. Therefore, we rely on the three below detailed local properties and the global property of the query looks like $\mathbf{G}_{T1}$, with a $T_o = 60000$ and $T_r = 4$, both values were chosen arbitrarily.

$$
\begin{aligned}
\mathcal{P}_{T1} &= \langle \texttt{New}(id = x_1)/(x_1 == 1)\rangle \rightarrow \{\texttt{Outcome}(id = x_2)/(x_2 == 1)\} \\
\mathcal{P}_{T2} &= \langle \texttt{Income}(id = x_1)/(x_1 == 1)\rangle \rightarrow \{\texttt{Outcome}(id = x_2)/(x_2 == 1)\} \\
\mathcal{P}_{T3} &= \langle \texttt{Income}(id = x_1)/(x_1 == 1)\rangle \rightarrow \{\texttt{Received}(id = x_2)/(x_2 == 1)\} \\
\mathbf{G}_{T1} &= \{\ \mathcal{P}_1,\ \mathcal{P}_2,\ \mathcal{P}_3\ \} \Rightarrow \{ \\
&\quad \langle \texttt{New}(checksum = y_1)\rangle \rightarrow \{\texttt{Received}(checksum = y_2)/(y_1 == y_2)\}, \\
&\quad \langle \texttt{Income}(checksum = y_3)\rangle \rightarrow \{\texttt{Outcome}(checksum = y_4)/(y_3 == y_4)\} \\
&\quad \}
\end{aligned}
$$

The nodes that can answer this query generate a transaction with the evaluation of the local property. Once the query and parameters are met, the consensus is triggered. The leader elected validates the global property of the query. Finally, it assembles a block and sends it to the network. There is a wait time of 5 seconds, and new random node triggers a query. This is repeatedly done for four consecutive times. Afterward, all the processes stop and the test case is over. The nodes use the MAC protocol of 802.11a. The transport protocol is UDP with a data rate of 54Mbps and a node range of ≈125m. Each node starts in a random position and starts moving in a random direction at a fixed velocity. After the test case has started, there is an initial configuration time, equal to the number of nodes to randomize the position of the nodes. The idea is for the nodes, to shuffle from their original location. All of this is done to enhance the trustability of the results by using random values to ensure that the algorithm works in any given scenario and that any emulation parameter does not tie it.

4.2. Scenarios

For all the scenarios, we define three different level of network densities, namely low, medium and high. The low density refers to at least 1.5 nodes every 100m×100m. The medium density refers to at least 2 nodes every 100m×100m. Finally, high density refers to at least 3 nodes every 100m×100m. It is worth mentioning that low density or high density could refer to more extreme cases, but we intend to experiment with different densities. Based on these densities, we define the network size for each number

Table 1: Emulation network parameters

Nodes	Low density	Medium density	High density	EC2 type
20	365x365	316x316	258x258	t2.medium
30	447x447	387x387	316x316	t2.large
40	515x515	447x447	365x365	m5.large
50	577x577	500x500	408x408	m5.xlarge

of nodes. The number of nodes determines the type of Amazon EC2 instance where the emulation is running. This is due to the fact of the computation power required by NS3 as the number of nodes and number of packages to simulate increases. The parameter relationship between the number of nodes, network size and Amazon EC2 instance type for the different node densities is summarized in Table 1. We define four scenarios to be emulated. The first scenario analyzes the efficiency of our approach by running the test case with a different number of nodes, different network sizes according to different densities, and different timeout for the raft process with a fixed speed and fixed mobility pattern. The second and third scenario analyze the efficiency as well with the previous parameters for a number of nodes and network size for different densities but having two different speeds and two different mobility patterns. Finally, the fourth scenario compares our results with other approaches. Each scenario runs, the previously defined test case, through 200 cycles parallel to the other scenarios. This number of cycles is approximately equivalent to running each scenario for 24 hours to ensure accurate and consistent results. Therefore, each data point of each graph represents approximately 800 runs of a query, meaning that each graph represents 9,600 runs of a query. It is fair to assume that each graph contains sufficient data points to ensure the quality of the measurements.

4.3. Implementation

All the layers of the implementation depend on a packet definition and the logs that are being monitored.

4.3.1. Packet definition

The packet is defined in a JSON ([36]) format summarized in Figure 2 and contains multiple subparts to work with the overall implementation. The handling of the packet is done by the Type property in the structure. The different types that can exist are: (a) query, (b) transaction, (c) block, (d) election, (e) raft_ReqVote, (f) raft_Vote, (g) raft_Ping, (h) raft_ConsensusEnd and (i) raft_Reset.

For example, if the packet is a query type, then the implementation handles it as a query and looks for the query structure inside the packet. For this specific type, the query structure depends on other structures (global property, local property, and candidate event) hence the arrows in Figure 2. This also means that it omits the transaction, block, and raft structure since they are not relevant to the query. Another example is for any of the raft types, the implementation looks for the raft structure in the packet and omits the query, transaction, and block.

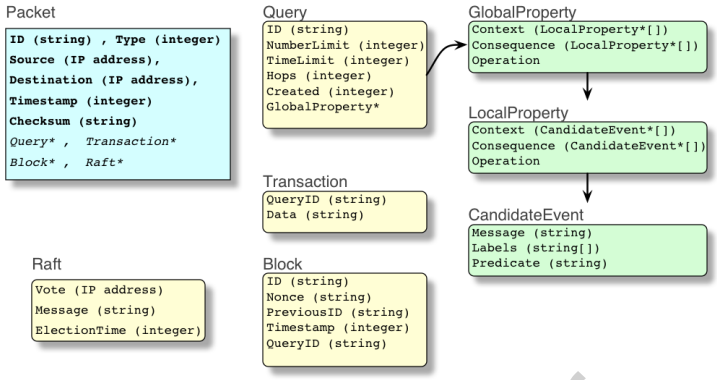


Figure 2: Packet structure

4.3.2. Log parsing

The monitoring of our properties and our approach is based on analyzing a set of traces. The set of traces we relied on is the logs of the node. For this, we assumed that the log contains a predefined format. The format contains default information as timestamp and severity, but it also contains the action and labels.

Listing 1: Log Format

```
Timestamp OtherFields* Action [labelN=valueN]+
```

For our implementation, we developed a small tool that parses the information from a tcpdump file and outputs our log format. The captured information is from an IoT network where a node routes a packet to another node using OSPF. From this, we captured the information for all participating nodes and created the log we use. It can be seen as an extra step to achieve the log, but this modularization allows a more easy adaptation for other tools.

4.4. Results

4.4.1. Results varying the number of nodes, node density and timeout

For this scenario, the intention is to emulate all the combinations defined in Table 1. For each pair of number of nodes and network size, we evaluate two timeouts (200ms and 300ms) for the raft process but use a fixed speed of 2m/s and a fixed mobility pattern of random waypoint model. This scenario is designed to test the time and monitoring efficiency of the overall approach. Also taking into account the resources that are being used, we analyze the number of packets generated by our approach. The results for the first timeout of 200ms are summarized in the following way: the results for the time to complete the query are in Figure 3a, the results for the accuracy of our approach are in Figure 4a, and the results for the number of messages generated by the raft leader election process are in Figure 5a. The results for the second timeout of 300ms are summarized in the following way: the results for the time to complete the query are in Figure 3c, the results for the accuracy of our approach are in Figure 4c, and the results for the number of messages generated by the raft leader election process are in Figure 5c. Looking at the results, the first apparent remark we can make is that the query completion time

(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

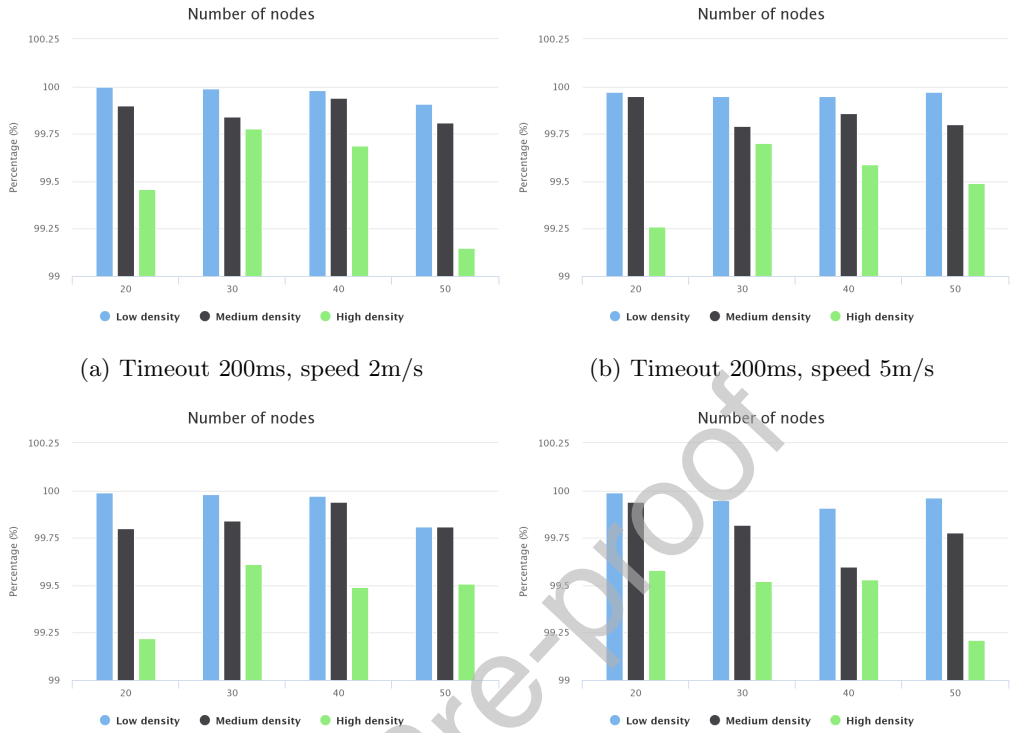
(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 3: Query Complete Time Results for Random Waypoint Mobility Model

increases as the number of nodes increases. Then if we look at the message counts for raft, we notice as the number of nodes increases the higher number of messages in the network. This is an interesting but obvious result since it means that the more nodes there are in the network, the most work (packets) and time it takes for the consensus protocol to elect a leader. Then we can observe some more interesting results for the query completion time, which is faster when the timeout is higher. This is not so obvious, but this is because a timeout of 200ms is too small for the needed interactions to complete an election process compared to the timeout of 300ms. Therefore, there are more problems and re-elections during the full process.

Another interesting comparison is between the results for a timeout of 200ms and a timeout of 300ms. The results for a timeout of 300ms, seem to follow a more clear trend due to the increase in time for the interactions to complete. Meanwhile, the timeout of 200ms, shows some possible irregularities between 40 and 50 nodes. This could be due to the geometry of the space in medium density which combined with a lower timeout causes more problems to complete the interactions than expected. This behavior can be observed in all of our experiments being explained by the same reasons. In a more profound analysis for the completion time, including results for both timeouts, we can refer to Table 2. Another highly promising result is the accuracy of the approach. As



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 4: Accuracy Results for Random Waypoint Mobility Model

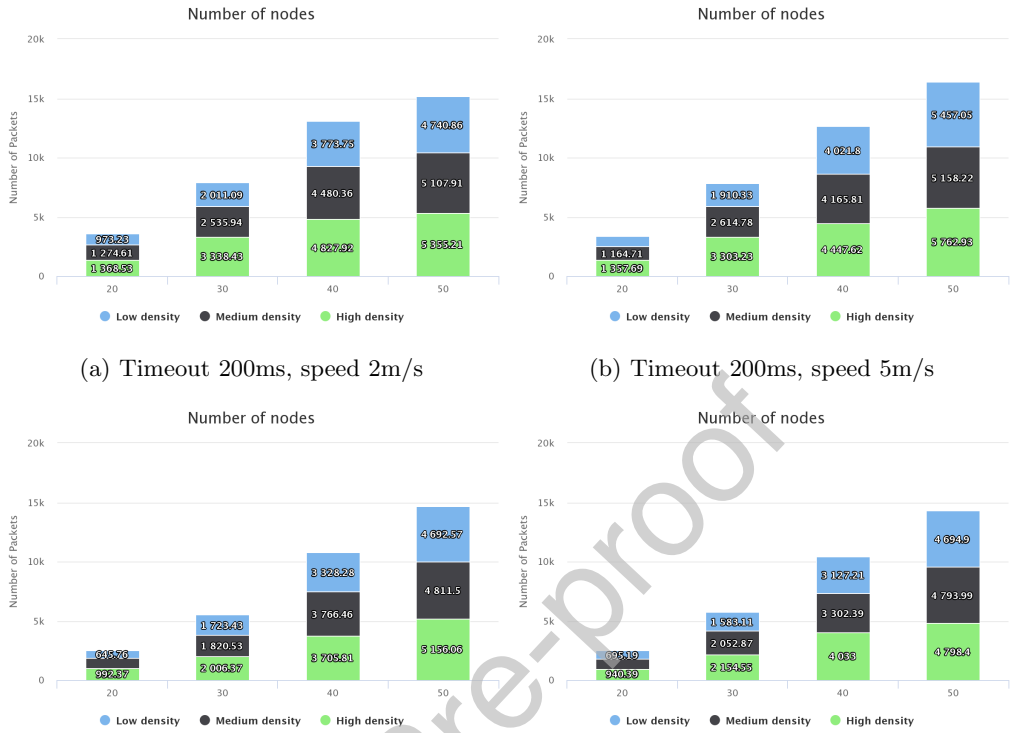
we can see in Figure 4a and Figure 4c, the accuracy of the approach ranges from 99% to 100%. This result is significant since it means that although several IoT inherent problems may occur, such as fragmentation, our approach converges to an accurate result.

Another interesting result is to notice the difference in accuracy for high density. This might be due to the fact that in high density scenarios, there might be more probabilities of messages clashing in the spectrum. This can lead to significant variances in the accuracy. Nonetheless, these variances (confidence interval in Table 2 and 3) are between 98.5% and 99.5%, which seem like acceptable. However, it would be relevant for future works to expand in high density scenarios.

It is worth mentioning that each figure contains a total of 7,500 emulations. The standard deviation for the timeout of 200ms is 1.90 and for the timeout of 300ms is 2.31. These results make the approach highly effective despite the number of nodes, the density, size of the network, and the selected timeout.

4.4.2. Results varying the number of nodes, node density and timeout for different node speed

For this scenario, the intention is to emulate all the combinations defined in Table 1. For each pair of number of nodes and network size, we evaluate two timeouts (200ms and



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 5: Raft Message Count Results for Random Waypoint Mobility Model

300ms) for the raft process but use two speeds of 2m/s and 5m/s and a fixed mobility pattern of random waypoint model. This scenario is designed to test the time and monitoring efficiency when the speed is different. Also taking into account the resources that are being used by analyzing the number of packets generated by our approach. The results for both timeouts at both speeds are summarized in the following way: the results for the time to complete the query are in Figure 3, the results for the accuracy of our approach are in Figure 4, and the results for the number of messages generated by the raft leader election process are in Figure 5. From these results, we can confirm the similar behavior in comparison to our previous results. For instance, the time for the query to complete is proportional to the number of nodes. There is no clear better or worst time when comparing the two different speeds. In a more in-depth analysis of the completion time, including results for both timeouts for the speed of 5m/s, we can refer to Table 3. Comparing Table 2 and Table 3, we can observe in more detail the similarities. In both cases, we have extreme cases as the minimum and maximum suggested. Based on the standard deviation, we can conclude that for both speeds the average query complete time can vary more as the number of nodes increases. Once again, this makes sense since trying to get to a consensus becomes more difficult the more participants there are. Therefore, it is logical that as the number of nodes increases, the variability of

Table 2: Statistical analysis for the query completion time at a speed of 2m/s

Nodes	Minimum	Maximum	Average	σ	CI 95%
20	1,495	58,743	4,129.33	4,936.19	2,310.18
30	497	58,998	5,341.81	7,250.71	2,707.42
40	737	59,753	8,759.20	10,558.74	3,376.86
50	430	59,997	9,415.15	10,278.96	2,921.29

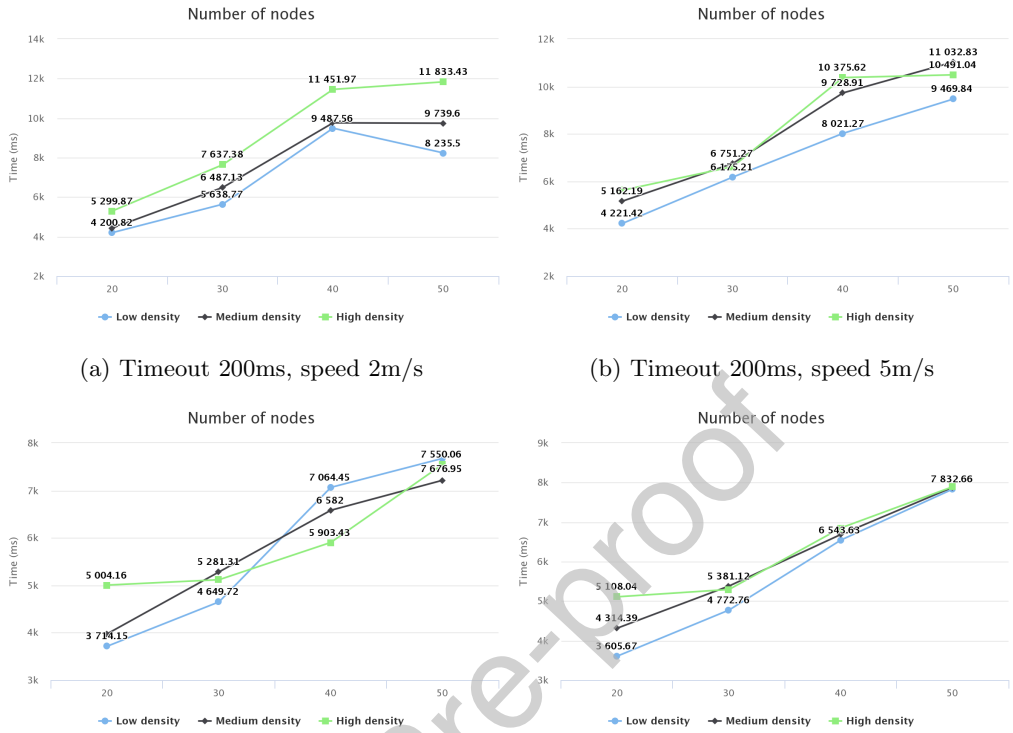
Table 3: Statistical analysis for the query completion time at a speed of 5m/s

Nodes	Minimum	Maximum	Average	σ	CI 95%
20	1,039	56,136	3,931.39	4,683.64	2,191.99
30	1,547	59,505	5,291.06	7,011.55	2,618.12
40	631	59,784	8,398.48	10,124.50	3,237.99
50	796	59,520	9,003.42	10,089.37	2,867.40

the completion time increases as well. Another impressive result is the accuracy of the result. We can observe that independently from the speed of the nodes, the approach can converge with high accuracy. The standard deviation for all of the accuracy results of a speed of 5m/s, for the timeout of 200ms is 1.78 and for the timeout of 300ms is 2.43. Then is fair to assume that the variability of the accuracy is extremely low, therefore making it very accurate and promising for future experiments.

4.4.3. Results varying the number of nodes, node density and timeout for different mobility models

For this scenario, the intention is to emulate all the combinations defined in Table 1. For each pair of number of nodes and network size, we evaluate two timeouts (200ms and 300ms) for the raft process but use two speeds of 2m/s and 5m/s and two mobility patterns of random waypoint model and random walk model. This scenario is designed to test the time and monitoring efficiency of the overall approach. Also taking into account the resources that are being used by analyzing the number of packets generated by our approach. The results for the random waypoint mobility model are summarized in the following way: the results for the time to complete the query are in Figure 3, and the results for the accuracy of our approach are in Figure 4. The results for the random walk mobility model are summarized in the following way: the results for the time to complete the query are in Figure 6, the results for the accuracy of our approach are in Figure 7. The results for the number of messages generated by the raft leader election process and by the router layer are omitted since are extremely similar to the results presented in the previous sections. Looking at the results from the query complete time for both mobility models, we can notice that the random walk mobility model results are slightly slower. If we look at it closer, we can analyze the average per number of nodes for all densities and speed for both mobility models. This can be seen in Table 4, where it is more evident that indeed is slightly slower. Changing to the accuracy results, we can observe, once again, that the approach is highly effective despite the mobility pattern. Nonetheless, it is important to point out that the accuracy results for high



(a) Timeout 200ms, speed 2m/s

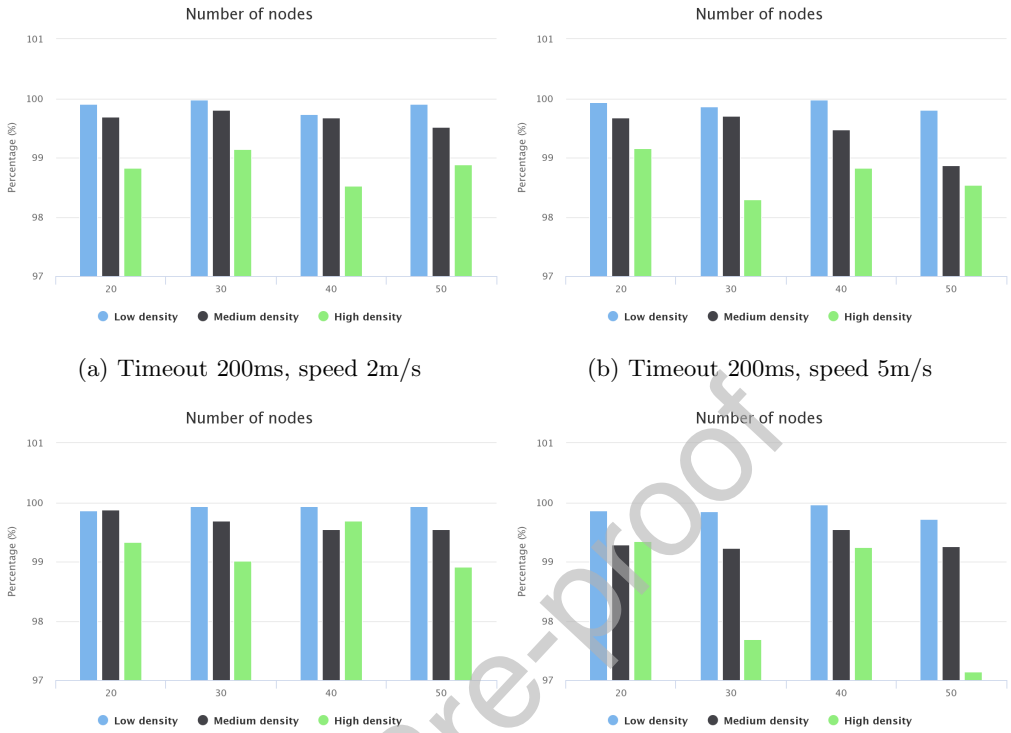
(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

Figure 6: Query Complete Time Results for Random Walk Mobility Model

density are more volatile, based on the observation of the standard deviation than the other densities. This is interesting since it shows a scenario where our approach decreases its accuracy but not by much which shows the efficiency of the approach. In a closer look to the accuracy results, the standard deviation for all of the results of a speed of 2m/s, for the timeout of 200ms is 2.57 and for the timeout of 300ms is 2.33. Also, the standard deviation for the results of a speed of 5m/s, for the timeout of 200ms is 3.21 and for the timeout of 300ms is 3.90. These values, compared to the results from the previous sections, show a higher variability due to the mobility pattern. In fact, the lower accuracy value is 97.14% (which can be seen in Figure 7d), and if you consider that is the scenario of 50 nodes, high density and speed of 5m/s, it is still quite accurate and efficient from our perspective. On the other hand, the raft message count results show an increase as well, it is a small increase, but it tells that due to the mobility, the consensus algorithm is having more difficulties trying to converge on a leader. This is expected since this is correlated to the query complete time, and for the random walk mobility model these values were higher. Setting aside the small increases in the results, we can confirm that our approach works efficiently, independently of the timeout, speed, network size, number of nodes and mobility model.



(a) Timeout 200ms, speed 2m/s

(b) Timeout 200ms, speed 5m/s

(c) Timeout 300ms, speed 2m/s

(d) Timeout 300ms, speed 5m/s

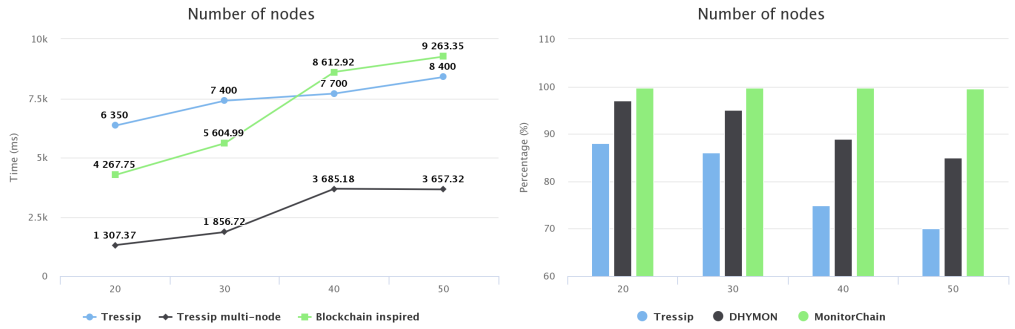
Figure 7: Accuracy Results for Random Walk Mobility Model

4.4.4. Results from comparing the proposed approach against Treesip and DHYMON

For this scenario, the intention is to compare the previous results with the approaches of Treesip [6] and DHYMON [10]. It is worth mentioning that although both results have shown promising results, neither of them is capable of monitoring a complex function like a global property. Both of them can provide a global view of local properties by performing an aggregation or average among the results, but they are not capable of computing a result that relies on more complex definitions. We can find the results of the convergence time and accuracy in Figure 8. The convergence time is defined as the time it takes to the approach from the start of the query to the end of the result. In our case, it is the query completion time, but we use the term convergence time for comparison purposes. The accuracy is defined as the validity of the final result. In our

Table 4: Query Complete Time for both mobility models

	20 nodes	**30 nodes**	**40 nodes**	**50 nodes**
Random Waypoint	4,301.26	5,800.91	9,998.94	10,489.95
Random Walk	4,847.68	6,738.90	10,256.85	10,768.09



(a) Convergence Time

(b) Accuracy

Figure 8: Convergence time and Accuracy for multiple approaches

approach, this is calculated by each node and then sent to the network. This means that the network by itself can determine the accuracy per query and therefore per block at runtime. In the approaches we are comparing, an outside analysis of the logs did this. Both approaches logged how many results were they able to aggregate, and the later analysis would consider this value and the number of nodes to calculate the accuracy ratio. The results regarding the convergence time are shown in Figure 8a. For the other approaches, although there is not a statistical analysis of the results, they do affirm that they emulated their test cases multiple times to ensure the veracity of the results. The DHYMON approach seems to be the most efficient in term of time, but it has not been tested with different densities and mobility models. The Treesip approach seems to be the overall slowest, and our approach is somehow in the middle. The results regarding the accuracy are shown in Figure 8b. This is where it gets interesting, although our approach is not the fastest, it is by far the most accurate. We can see that the Treesip has the worse accuracy ranging from 88% to 70% and the DHYMON ranges from 97% to 85%. The DHYMON has a more steady accuracy but is not as consistent as the 99% of our approach. It is also worth mentioning that the other approaches have not conducted tests to varying the densities of the networks, which could diverge the results. It seems that the other approaches have tried to increase the accuracy, but it always decreases while the number of nodes increases. In our case, the accuracy is outstanding but the time could be improved.

5. Related works

In the literature, Mobile IoT has been widely studied. Indeed, the interconnected devices, or objects, can communicate, actuate, exchange their information over Internet to the end users by using the communication technologies and networks [29, 37]. For such systems, the problematic of monitoring IoT is of high interest and has been studied in the literature [1] as well as in a recent paper of Raposo [2]. Although sensors-based/IoT networks are commonly utilized to monitor a system (environmental, healthcare, etc.) [38, 39], their nodes themselves are also needed to be monitored for several reasons such as testing, security, reliability, maintainability, etc. Indeed, the monitoring of IoT allows

to have some information of the state of the network and to check the correctness or security of the system [40, 41]. In a mobile environment, distributed monitoring is needed as there is no centralized and fixed infrastructure. In Battat et al. [17], it can be said that monitoring can be classified into centralized and decentralized or distributed monitoring. In the decentralized monitoring context, we can mention three prominent subcategories: (a) gossip, (b) hierarchical or (c) hybrid approaches.

In the gossip-based categorization, we can discuss Mobi-G [5] which is inspired by Gossipico [42] for static networks. This is an algorithm to calculate the average, the sum or the count of node values in a large dynamic network. It can provide accurate results even for fluctuating attributes. It also can reduce the communication cost. It does not suffer from extended range connectivity, but the accuracy decreases for an increasing spatial network size. In the hierarchical categorization, BlockTree [9] proposes a fully decentralized location-aware monitoring mechanism. The idea is to divide the network into proximity-based clusters, which are arranged hierarchically. Even though the excellent performance, the average power consumption increases directly proportional to the spatial network size or node density. For the hybrid categorization, we can look at DHYMON [10]. The proposal of DHYMON, which is an extension of Treesip [6], enhances the accuracy of it by creating multiple root nodes and running redundant monitoring processes to improve the overall result. This approach is a middle point between a decentralized and distributed approach. The overall accuracy is increased, but as the number of nodes in the network increases, it affects in a direct proportion the accuracy.

Other related approaches are the work from Belfkih et al. [43], this approach is applied over wireless sensor networks (WSN), but they share the same resource limitations for IoT Manet-based systems. They propose an improvement over classical SQL-Like query language processors for WSN to query values of the network. They evaluate their proposal relying on some relay nodes that use some scheduling algorithms to process the responses. It is important to mention that their monitoring results had a convergence time between 10s to 20s. An interesting approach is the work of Battat et al. [11]. They define a trust computation method for a node to determine if neighboring nodes are showing malicious or selfish behaviors. This is done by proposing an architecture that is logically divided into clusters. Then each cluster elects a monitor head. In their simulations, they provide a low message overhead, but the election methods could be weak compared to the leader election of a consensus algorithm. In the recent work of Li et al. [8], the authors present a distributed consensus algorithm employed for events detection in IoT and CPS. Considering dynamic mobility changes, the methodology is based on compressed sensing for process monitoring. Although their objective is different from ours, they show that events detection and a fast convergence of the results may be obtain with a distributed consensus algorithm. In [44], they propose a reference model to enable the verification and certification of IoT WSN-based applications. Such as the work addressed in this paper, the authors propose a monitoring platform to improve the correctness of the behavior of IoT applications. The accent in this paper is put on the design on a standardized monitoring platform.

By surveying the literature, we can find also several works that have been done to address the verification of global properties in a distributed manner. Formal tools or formal languages can be used for that purpose. In [45], they use the GOANNA model checker [46] to prevent threats of IoT implementations, the model checker is based on

static program analysis. The security aspects is out of the scope of our paper and it is not clear how this approach can be used to verify global properties in a distributed manner. Other approaches more close to our work are based on publish/subscribe principles such as [12, 13] which are topic-based pub/sub middleware, that are limited to topic-based routing. The middleware [47] is also too heavy for IoT sytems. Moreover, they allow to warranty for instance QoS which is not the target of our framework. All these systems are middleware which appear to be either too heavy for IoT-based systems or limited to QoS evaluation. In [14], which has been implemented for TinyOS [48] operating system, they prevent such drawback by distributing in a more efficient manner the events on the network. Indeed, the subscribers are notified about the evaluation to true only for the properties they request, the properties are first order logic formulas. This approach needs to deploy a PICO-MP network and it is a tree-based approach. Even, if the numbers of messages can be reduced, we have no information about the accuracy of the evaluation of the global properties. In our case, we put the accent on a lightweight approach based on traces analysis and on a consensus protocol to reduce the number of messages and on the verdict accuracy. Besides, we define formally how the properties can be expressed and how accurate the results can be. For that purpose, we formalise the non functional and functional properties relying on previous work such as [49, 50, 51, 52]. Indeed, these works have been an inspiration for the herein proposed approach in terms of models or formal definitions. But they are mostly limited to local validation made by local nodes and do not allow a distribution of the verification process.

6. Conclusions

We have presented in this paper a fully distributed architecture for monitoring Internet of Things networks that consists of the combination of a distributed record, consensus algorithm, and a network protocol. Providing a robust and scalable solution, taking advantage of the best qualities for the used technologies focused on the IoT context. The architecture starts by propagating a query that contains a global property and a set of local property to satisfy. Each node calculates the local properties, and if it can satisfy it, it replies a transaction to the network. Then based on the query criteria, all the nodes can switch to aggregate and complete the query. A consensus algorithm does this and this allows the network to choose a leader for that query in a distributed, deterministic and democratized way. This node aggregates the local views and computes the global property. The process derives a block as a result, which is sent to the network and all the nodes agree on the result. It is important to highlight that each node can validate, audit and agree on a result. Which not only allows the network to consider security concerns but also to ensure the trustability of the data, and all of this is done by design. Based on our study, we concluded that the approach has an incredibly high accuracy compared to other approaches. We also defined the following:

1. strong mathematical background describing candidate events, local properties, global properties and all the following structures for our architecture. This provides a strong mathematical test ground for the theory.
2. The fully distributed monitoring architecture that is designed to withstand multiple of the problematics of the mobile IoT networks. This includes mobility, availability and resource consumption.

3. The importance and the accuracy of a novel distributed monitoring architecture through the combination of a distributed ledger and a consensus algorithm. It is worth highlight that this was proven by the emulation of multiple scenarios, combining different parameters, and ran multiple times.

As future works, based on our experiments it is somehow clear to spot that the consensus algorithm provides a time bottleneck. Therefore we think that the study of stronger consensus algorithms in the presence of mobility is needed. Aside from this, another area of interest could be the utilization of a different query propagation mechanism. For the moment we relied on an epidemic algorithm, but maybe a more refined gossip algorithm could provide enhancements. There is a need to analyze and run emulations of the architecture with online scenarios. For the ease of testing this architecture, we relied on offline data. Despite this, the transition to online data would require attention in the generation of our predefined log and an improved way to handle the query completion. We also think that although this architecture was heavily tested, it would be interesting to compare different types of global properties in order to compare the impact of these computations in the overall result. Finally, we aim at studying the energy consumption of our approach and to compare the several existing solutions proposed in the literature.

References

[1] I. Lee, K. Lee, The internet of things (iot): Applications, investments, and challenges for enterprises, Business Horizons 58 (4) (2015) 431–440.

[2] D. Raposo, A. Rodrigues, S. Sinche, J. Sá Silva, F. Boavida, Industrial iot monitoring: Technologies and architecture proposal, Sensors 18 (10) (2018) 3568.

[3] W. Kiess, M. Mauve, A survey on real-world implementations of mobile ad-hoc networks, Ad Hoc Networks 5 (3) (2007) 324–339.

[4] N. Raza, M. U. Aftab, M. Q. Akbar, O. Ashraf, M. Irfan, Mobile ad-hoc networks applications and its challenges, Communications and Network 8 (2016) 131–136.

[5] D. Stingl, R. Retz, B. Richerzhagen, C. Gross, R. Steinmetz, Mobi-g: Gossip-based monitoring in manets, in: IEEE Network Operations and Management Symposium, 2014.

[6] J. Alvarez, S. Maag, F. Zaïdi, Monitoring dynamic mobile ad-hoc networks: A fully distributed hybrid architecture, in: 31st IEEE International Conference on Advanced Information Networking and Applications, AINA 2017, Taipei, Taiwan, March 27-29, 2017, 2017, pp. 407–414.

[7] T. M. D. Tran, A. Y. Kibangou, Collaborative network monitoring by means of laplacian spectrum estimation and average consensus, International Journal of Control, Automation and Systems 17 (7) (2019) 1826–1837.

[8] S. Li, S. Zhao, P. Yang, P. Andriotis, L. Xu, Q. Sun, Distributed consensus algorithm for events detection in cyber-physical systems, IEEE Internet of Things Journal 6 (2) (2019) 2299–2308.

[9] D. Stingl, C. Gross, L. Nobach, R. Steinmetz, D. Hausheer, Blocktree: Location-aware decentralized monitoring in mobile ad hoc networks, in: IEEE 38th Conf. Local Computer Networks, 2013, pp. 373–381.

[10] J. Alvarez, S. Maag, F. Zaidi, Dhymon: a continuous decentralized hybrid monitoring architecture for manets, arXiv preprint arXiv:1712.01676 (2017).

[11] N. Battat, A. Makhoul, H. Kheddouci, S. Medjahed, N. Aitouazzoug, Trust based monitoring approach for mobile ad hoc networks, in: International Conference on Ad-Hoc Networks and Wireless, Springer, 2017, pp. 55–62.

[12] U. Hunkeler, H. L. Truong, A. J. Stanford-Clark, MQTT-S - A publish/subscribe protocol for wireless sensor networks, in: S. Choi, J. Kurose, K. Ramamritham (Eds.), Proceedings of the Third International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2008), January 5-10, 2008, Bangalore, India, IEEE, 2008, pp. 791–798.

[13] J. Chen, M. Díaz, B. Rubio, J. M. Troya, PS-QUASAR: A publish/subscribe qos aware middleware for wireless sensor and actor networks, J. Syst. Softw. 86 (6) (2013) 1650–1662.

[14] N. Dulay, M. Micheletti, L. Mostarda, A. Piermarteri, PICO-MP: de-centralised macro-programming for wireless sensor and actuator networks, in: L. Barolli, M. Takizawa, T. Enokido, M. R. Ogiela, L. Ogiela, N. Javaid (Eds.), 32nd IEEE International Conference on Advanced Information Networking and Applications, AINA 2018, Krakow, Poland, May 16-18, 2018, IEEE Computer Society, 2018, pp. 289–296.

[15] J. A. Alvarez Aldana, Une méthode de test fonctionnel en-ligne basée sur une approche de monitorage distribuée continue appliquée aux systèmes communicants, Ph.D. thesis, thèse de doctorat dirigée par Maag, Stéphane Informatique Université Paris-Saclay (ComUE) 2018 (2018).
URL http://www.theses.fr/2018SACLL005

[16] G. Cormode, The continuous distributed monitoring model, ACM SIGMOD Record 42 (1) (2013) 5–14.

[17] N. Battat, H. Seba, H. Kheddouci, Monitoring in mobile ad hoc networks: A survey, Computer Networks 69 (2014) 82–100.

[18] G. Cormode, Continuous distributed monitoring: a short survey, in: Proceedings of the First International Workshop on Algorithms and Models for Distributed Event Processing, ACM, 2011, pp. 1–10.

[19] I. Sharfman, A. Schuster, D. Keren, A geometric approach to monitoring threshold functions over distributed data streams, ACM Transactions on Database Systems (TODS) 32 (4) (2007) 23.

[20] A. E. Goodloe, L. Pike, Monitoring distributed real-time systems: A survey and future directions (2010).

[21] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Transactions on Programming Languages and Systems (TOPLAS) 4 (3) (1982) 382–401.

[22] J. Qin, Q. Ma, Y. Shi, L. Wang, Recent advances in consensus of multi-agent systems: A brief survey, IEEE Transactions on Industrial Electronics 64 (6) (2017) 4972–4983.

[23] M. J. Fischer, The consensus problem in unreliable distributed systems (a brief survey), in: International Conference on Fundamentals of Computation Theory, Springer, 1983, pp. 127–140.

[24] L. Lamport, et al., Paxos made simple, ACM Sigact News 32 (4) (2001) 18–25.

[25] M. Burrows, The chubby lock service for loosely-coupled distributed systems, in: Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association, 2006, pp. 335–350.

[26] D. Ongaro, J. K. Ousterhout, In search of an understandable consensus algorithm., in: USENIX Annual Technical Conference, 2014, pp. 305–319.

[27] S. R. D. C. E. Perkins, E. M. Belding-Royer, Ip flooding in ad hoc mobile networks, Tech. rep., IETF (2001).

[28] M. Healy, T. Newe, E. Lewis, Power management in operating systems for wireless sensor nodes, in: 2007 IEEE sensors applications symposium, IEEE, 2007, pp. 1–6.

[29] F. Javed, M. K. Afzal, M. Sharif, B.-S. Kim, Internet of things (iot) operating systems support, networking technologies, applications, and challenges: A comparative review, IEEE Communications Surveys & Tutorials 20 (3) (2018) 2062–2100.

[30] M. Zimmerling, L. Mottola, S. Santini, Synchronous transmissions in low-power wireless: A survey of communication protocols and network services, arXiv preprint arXiv:2001.08557 (2020).

[31] M. S. Aslanpour, S. S. Gill, A. N. Toosi, Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research, Internet of Things (2020) 100273doi:https://doi.org/10.1016/j.iot.2020.100273.
URL http://www.sciencedirect.com/science/article/pii/S2542660520301062

[32] H. N. Nguyen, P. Poizat, F. Zaïdi, Online verification of value-passing choreographies through property-oriented passive testing, in: 14th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2012, Omaha, NE, USA, October 25-27, 2012, 2012, pp. 106–113.

[33] X. Che, S. Maag, H. N. Nguyen, F. Zaïdi, Guiding testers' hands in monitoring tools: Application of testing approaches on SIP, in: Testing Software and Systems - 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25, 2015, Proceedings, 2015, pp. 105–123.

[34] M. A. To, M. Cano, P. Biba, Dockemu–a network emulation tool, in: 2015 IEEE 29th international conference on advanced information networking and applications workshops, IEEE, 2015, pp. 593–598.

[35] E. Petersen, G. Cotto, M. A. To, Dockemu 2.0: Evolution of a network emulation tool, in: 2019 IEEE 39th Central America and Panama Convention (CONCAPAN XXXIX), IEEE, 2019, pp. 1–6.

[36] T. Bray, The javascript object notation (json) data interchange format, Tech. rep., RFC Editor (2014).

[37] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, N. B. Anuar, The rise of traffic classification in iot networks: A survey, Journal of Network and Computer Applications (2020) 102538.
[38] B. Thilagavathi, S. Parthasarathy, A. R. Lakshminarayanan, Survey on utilization of internet of things in health monitoring systems, in: Proceeding of the International Conference on Computer Networks, Big Data and IoT, Vol. 49, Springer Nature, 2020, p. 340.
[39] H. Mokrani, R. Lounas, M. T. Bennai, D. E. Salhi, R. Djerbi, Air quality monitoring using iot: A survey, in: 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), IEEE, 2019, pp. 127–134.
[40] M. T. Lazarescu, Design of a WSN Platform for Long-Term Environmental Monitoring for IoT -applications, IEEE J. Emerg. Sel. Top. Circuits Syst. 3 (2013) 45–54.
[41] V. Casola, A. De Benedictis, A. Riccio, D. Rivera, W. Mallouli, E. M. de Oca, A security monitoring system for internet of things, Internet of Things 7 (2019).
[42] R. Van De Bovenkamp, F. Kuipers, P. Van Mieghem, Gossip-based counting in dynamic networks, in: International Conference on Research in Networking, Springer, 2012.
[43] A. Belfkih, C. Duvallet, B. Sadeg, L. Amanton, A real-time query processing system for wsn, in: International Conference on Ad-Hoc Networks and Wireless, Springer, 2017, pp. 307–313.
[44] J. V. Capella, J. C. Campelo, A. Bonastre, R. Ors, A reference model for monitoring iot wsn-based applications, Sensors 16 (11) (2016) 1816.
[45] R. Huuck, Iot: The internet of threats and static program analysis defense, Tech. rep. (2015).
[46] R. Huuck, A. Fehnker, S. Seefried, J. Brauer, Goanna: Syntactic software model checking, in: S. D. Cha, J. Choi, M. Kim, I. Lee, M. Viswanathan (Eds.), Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings, Vol. 5311 of Lecture Notes in Computer Science, Springer, 2008, pp. 216–221.
[47] G. Li, H. Jacobsen, Composite subscriptions in content-based publish/subscribe systems, in: G. Alonso (Ed.), Middleware 2005, ACM/IFIP/USENIX, 6th International Middleware Conference, Grenoble, France, November 28 - December 2, 2005, Proceedings, Vol. 3790 of Lecture Notes in Computer Science, Springer, 2005, pp. 249–269.
[48] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. L. Hill, M. Welsh, E. A. Brewer, D. E. Culler, Tinyos: An operating system for sensor networks, in: W. Weber, J. M. Rabaey, E. H. L. Aarts (Eds.), Ambient Intelligence, Springer, 2005, pp. 115–148.
[49] P. Mouttappa, S. Maag, A. R. Cavalli, Using passive testing based on symbolic execution and slicing techniques: Application to the validation of communication protocols, Computer Networks 57 (15) (2013) 2992–3008.
[50] F. Lalanne, S. Maag, A formal data-centric approach for passive testing of communication protocols, IEEE/ACM Trans. Netw. 21 (3) (2013) 788–801.
[51] X. Che, S. Maag, H. Tan, H. Tan, Z. Zhou, A passive testing approach for protocols in wireless sensor networks, Sensors 15 (11) (2015) 29250–29272.
[52] S. A. Abid, M. Othman, N. Shah, 3d p2p overlay over manets, Computer Networks 64 (2014) 89–111.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.