SN Computer Science (2024) 5:111
https://doi.org/10.1007/s42979-023-02412-w

ORIGINAL RESEARCH

Dynamic Topics Management in Publish/Subscribe Systems over Mobile Ad Hoc Networks

Martin Xavier Tchembé[1] · **Maurice Tchoupé Tchendji**[1]

Received: 17 July 2022 / Accepted: 11 October 2023

Abstract
Information dissemination by publish/subscribe in Mobile Ad hoc NETworks (MANETs) has been the subject of numerous studies over the last few decades. There are generally two main classes in the taxonomy of publish/subscribe systems: the content-based publish/subscribe and the topic-based publish/subscribe. The latter generally offers a predefined set of (possibly hierarchical) topics to publishers and subscribers. This set of topics in the context of applications deployed on MANETs is duplicated within each station of the network. It can be difficult when designing such applications to know in advance all the topics that will be dealt with. The question of how to update this set of topics once the application is deployed is a major concern. This paper proposes a distributed protocol which extends the SocialMANET protocol for information dissemination in topic-based publish/subscribe systems for MANETs, to make it capable of dynamically updating the set of topics in such a system. The proposed protocol guarantees the uniqueness of the topic identifiers as well as the coherence of the different replicas (of the hierarchy) of the topics on the different stations in the network. Experimental studies carried out on this protocol have shown that it effectively disseminates topic updates as long as the network stations communicate frequently.

Keywords Distributed updates · Dynamic topics · MANETs · Publish/subscribe · Logical clock

Introduction

Thanks to its asynchronous nature and the high level of decoupling (spatial decoupling, temporal decoupling and synchronisation decoupling) it offers [1], the publish/subscribe communication paradigm appears to be a very attractive candidate for the dissemination of information in MANETs [2]. The main actors in a publish/subscribe system are the publishers who supply information to the system, and the subscribers who consume the information present in the system. The different ways in which subscribers can mark their interest in the information in the system by means of subscriptions make it possible to distinguish between two main variants of publish/subscribe: the content-based variant where information is only delivered to a subscriber if the attributes or content of the information match the constraints defined by its subscription; and the topic-based variant where the information is published in "topics" or named logical channels. Subscribers to a topic-based publish/subscribe system will receive all messages published in the topics to which they subscribe.

In topic-based publish/subscribe systems, the interacting parties exchange information through a set of predefined topics that represent many distinct (and fixed) logical channels [3]. In the context of MANETs, the set of topics is defined beforehand and then, a copy of it is made available at the level of each station during the deployment: each station in the network, therefore, has its own copy of the set of topics. In case of an error in the preparation of the set of topics or changing needs afterwards, it would be difficult to update if the system is already deployed. If we take the example of a publish/subscribe system for the dissemination of information during a scientific conference where there is a "*cryptography*" topic representing a discussion group around this topic, we can imagine that new topics are created (e.g.: *symmetrical cryptography*, *asymmetrical cryptography*) to further refine the discussions in small groups. To our knowledge, this issue has never been addressed in the case of publish/subscribe systems for MANETs. Existing solutions

✉ Martin Xavier Tchembé
martinxaviertchembe@gmail.com

Maurice Tchoupé Tchendji
ttchoupe@yahoo.fr

[1] Department of Mathematics and Computer Science, University of Dschang, Dschang, Cameroon

Published online: 18 December 2023

[4–12] are designed to operate in infrastructure-based networks; they need to be revisited or even reinvented to meet the requirements of MANETs: lack of centralised control, mobility, dynamic topology, etc.

In this paper, we propose a distributed protocol for the dynamic management of the set of (hierarchical) topics of a publish/subscribe system deployed in a MANET. The proposed protocol is an extension of the protocol for information dissemination in MANETs named SocialMANET [13]. This extension can be observed at several levels:

Messages

The *QUERY* announcement message of the SocialMANET protocol has been modified to add information on operations that have occurred on certain topics (addition, modification, deletion). New messages have been introduced, such as the *REFRESH_REQUEST*, *UPDATE_REPLY* and *FORCE_DELETE* messages detailed in "A dynamic topics management protocol for publish/subscribe systems in MANETs".

Data structures

Two new data structures have been introduced to represent the topics and their hierarchical organisation.

Primitives

Primitives have also been introduced, mainly for creating a new topic, modifying or deleting an existing topic, and updating the local set of topics when an *UPDATE_REPLY* message has been received.

Parameters

New parameters have been introduced such as *UPDATE_PROPAGATION_TIMEOUT* and *AWARENESS_THRESHOLD*.

Topic identification

The identifiers of newly created topics are generated using a different technique which consists of inserting the identifier of the creating station into the topic identifier in order to avoid possible identifier conflicts.

Apart from the dynamic topic management capability this extension gives the SocialMANET protocol, it makes it possible to ensure that the topics created in a distributed manner have unique identifiers; it also ensures that the different replicas of the hierarchy of topics hosted by each station are consistent. Indeed, once a station has carried out a local update of the hierarchy of topics it hosts, the proposed protocol disseminates it to all the other stations in the system. Simulations carried out to test the performance of this protocol show that it ensures that all the replicas of the topic hierarchy are updated within a reasonable time. However, it should be noted that the additional load on the network in terms of the number of messages exchanged as a result of these updating operations is not negligible compared to the number of messages exchanged using the core SocialMANET protocol; nevertheless, we have made few recommendations to minimise it.

The rest of this paper is organised as follows: "Related Works" presents a state of the art on dynamic topics management in publish/subscribe systems, section "Overview of the SocialMANET Protocol" presents a brief overview of the SocialMANET protocol. Section "A dynamic topics management protocol for publish/subscribe systems in MANETs" is devoted to the presentation of the proposed approach. A practical example of its use is presented in "Sample Case: Illustration of the Protocol". The results of the simulations carried out and their discussions are presented in "Performances". "Conclusion" concludes this paper.

Related Works

Several works in the literature have dealt with the problem of information dissemination following the topic-based publish/subscribe communication model in MANETs [2, 13–15]. Very few, however, have dealt with the changes that can take place on the topics at stake (dynamic topics). Some authors have rather focused on the issue of dynamic subscription allowing subscribers to dynamically modify their subscriptions, or to automatically subscribe/unsubscribe to certain topics under certain conditions (low battery level, location, etc.) [16–18]. Among the work carried out on the issue of dynamic topics management, are scientific papers [10–12] and software systems [4–9].

Software Systems Offering Publish/Subscribe with Dynamic Topics Management

Amazon SQS [4] is a reliable and highly scalable hosted queue for storing messages as they flow between applications or micro services. It moves data between distributed application components and helps decouple these components. It can be viewed as a topic-based publish/subscribe system where the topics are the queues. Amazon SQS provides various APIs for performing various operations on queues (create, modify, delete, set permissions, etc.).

Google Pub/Sub [5] is a messaging and ingest service for event-driven systems and streaming analytic. It allows to send and receive messages among various applications. It is a centralised, topic-based publish/subscribe system that offers a set of APIs via various platforms to dynamically manage topics (create, delete, view).

IBM MQ [6] is a system similar to Google Pub/Sub, providing an universal messaging infrastructure with robust connectivity for flexible and reliable application messaging. It also offers APIs via various platforms to create topics and view the list of topics to which subscribers can subscribe.

There are many other similar tools on the market nowadays. These include Apache Kafka [7], Rabit MQ [8], Red Hat AMQ [9], etc.

Scientific Papers Dealing with Publish/Subscribe with Dynamic Topics Management

A distinction is made between work on distributed systems [10, 11] and work on centralised systems [12].

Antony Rowstron et al. in [10] present Scribe, a large-scale distributed event-based notification platform for topic-based publish/subscribe applications. Scribe is built on top of Pastry [19], a generic peer-to-peer object search and routing system on the Internet. Scribe nodes have the ability to create new topics which they can then subscribe to as needed. The creation of a new topic uses an identifier to be assigned to the topic and is done via an API offered by Scribe, which then delegates this task to Pastry.

Similarly, it is presented in [11], a distributed framework for creating and discovering topics in distributed publish/subscribe systems. It allows a node wishing to create a new topic to issue a topic creation request by providing the necessary data (topic creator, lifetime of the topic, etc.), addressed to special nodes called TDNs (Topic Discovery Nodes), which are responsible for tracking the topics contained in the system. These TDNs also allow the nodes to find the topics in the system via topic discovery requests.

Early implementations of the SBML server [20] for publish/subscribe statically predefined the topics to which subscribers could subscribe. The work carried out in [12] has enabled authors to extend this server to include dynamic topics management. It offers two services to clients: *getMsgSelectors()* to retrieve the list of topics and *addMsgSelector(String search)* to define new topics dynamically (while the server is running), without having to go through the manual steps of modifying certain configuration files (*topicDefinitions.xml*, *SBMLTopics-services.xml*) and then restarting the server as was the case in the past.

It should be remembered that as far as our knowledge is concerned, not much work has been done on the topic of dynamic topics management in publish/subscribe systems. The few that we have found and presented above have designed solutions operating only on infrastructure-based networks, where the stations are perfectly addressable and relatively stable (immobile and almost always connected to the network). These solutions are unfortunately unsuitable for MANETs where the network is built spontaneously and has unstable nodes that can leave and enter the network as they constantly move.

It should also be noted that in the above-mentioned work, only certain aspects of the dynamic nature of the topics are addressed. In fact, although all of them dealt with the issue of adding new topics, the issue of deleting topics (resp. modifying existing topics) was only dealt with by software products such as Apache Kafka [7], Amazon SQS [4] and Google Pub/Sub [5]. It is easy to see that these concerns need to be addressed differently for distributed applications deployed in MANETs. In fact, as the set of topics can no longer be centralised on server(s), each station in the network must have its own copy (replica) of this set and the consistency of these copies must be ensured.

Overview of the SocialMANET Protocol

The protocol we propose in this paper is an extension of the information dissemination protocol in MANETs called SocialMANET [13]. This extension aims to give the SocialMANET protocol the ability to provide dynamic topic management, allowing network stations to add, delete and modify topics while the application implementing the protocol is deployed. In order to set the basis for the presentation of the protocol for dynamic updates of the topics dealt with in this paper, we give below a brief presentation of the SocialMANET protocol. The interested reader can consult [13] for a more complete presentation.

SocialMANET is a topic-based publish/subscribe information dissemination protocol for MANETs. Since SocialMANET runs on a network that can be discontinuous,[1] each station running SocialMANET has a local copy of all topics in the system to ensure their availability in real time during subscription and publication operations. In fact, given the discontinuity of the network, it is not possible to have a central element in charge of hosting and managing the topics, because it may be off the network at given times or out of reach of certain stations.

SocialMANET implements the hierarchical topic subscription model [1] which, in addition to allowing the formulation of refined subscription requests close to the content-based subscription model [1], also minimises the number of subscriptions required to receive publications in several topics; this gives a tree-like structure to the set of topics (we will simply call it later the "*topics tree*"). SocialMANET consists of three sub-protocols: the subscription sub-protocol, the publication sub-protocol and the dissemination sub-protocol. The latter is carried out in two phases, one of detection of needs and another of transfer of publications. During the phase of detection of the needs, and in a periodic way, a station broadcasts in the network, by a *QUERY* message whose format is illustrated by Fig. 1, an announcement revealing the list of the identifiers of the topics to which it is subscribed as well as those of the publications stored locally on these topics. In this message, *ID* represents the identifier of the transmitting station, *topic-i* the identifier of the i-th topic to which it subscribes and *key-i-j* the identifier of the j-th publication it has on the i-th topic.

[1] Network discontinuity can keep stations off the network for long periods of time.

```
ID [
    TOPIC-1 (KEY-1-1, KEY-1-2, ..., KEY-1-N1)
    TOPIC-2 (KEY-2-1, KEY-2-2, ..., KEY-2-N2)
     ...
    TOPIC-M (KEY-M-1, KEY-M-2, ..., KEY-M-NM)
]
```

Fig. 1 Format of the QUERY announcement message

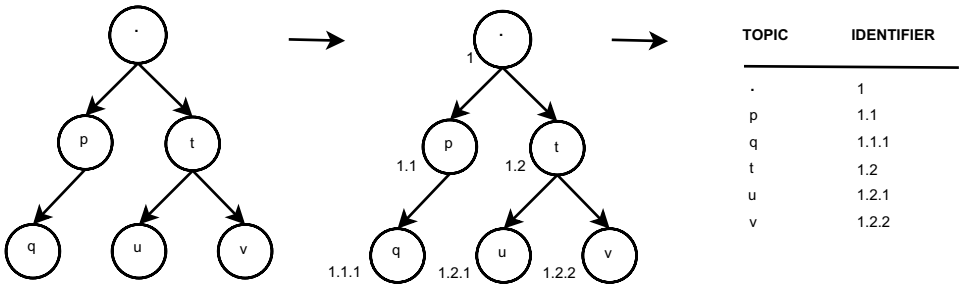

Fig. 2 Topic identifiers generation

These topic identifiers are generated using the Dewey identification technique [21] from the topic tree; each topic in the tree is represented by a path from the root. Figure 2 illustrates this mechanism of generating topic identifiers with a tree consisting of the topics p, q, t, u and v. It is thus easy to recognise, given two topics, whether one is an ancestor of the other. Indeed, a topic with identifier *x* is ancestor of a topic with identifier *y* if *x* is a prefix of *y*.

In the remainder of this document, the use of the Dewey identification technique to generate topic identifiers does not imply that there are no holes[2] in the set of topic identifiers if they have been created by other stations. We made this choice because the semantic of a topic is very much linked to that of its hierarchical ancestors (as it is only a refinement of them). It is, therefore, logical that if we consider a topic to be no longer relevant, its hierarchical descendants should also no longer be relevant.

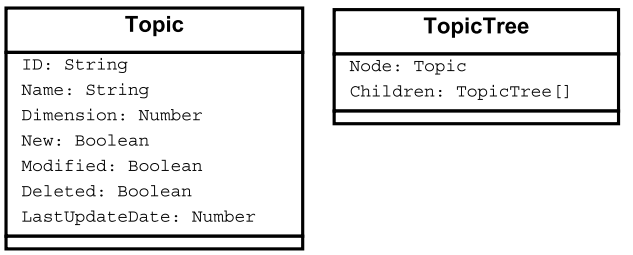

Fig. 3 A representation of the *Topic* and *TopicTree* data structures used in the protocol

A Dynamic Topics Management Protocol for Publish/Subscribe Systems in MANETs

Assumptions

The proposed protocol is based on the following assumptions:

A1. The network we consider is a network with discontinuous connectivity, defined in [15] as a network made up of a set of clusters,[3]

A2. Only the station that created a topic can edit or delete it. However, during this operation, if the topic to be deleted has child topics, they will also be deleted even

The proposed protocol addresses all cases of updating that may occur on the hierarchy of topics, namely (1) the addition of a new topic, (2) the modification of an existing topic and (3) the deletion of an existing topic. Each of these operations is carried out in two stages: firstly, updating the hierarchy of topics stored locally at the station initiating the operation (stage 1), then disseminating this update to the other stations in the network (stage 2).

Data Structures and Notations

Two main data structures are used in the protocol: The *Topic* data structure (modelling one topic from the set of topics in the system) and the *TopicTree* data structure (modelling the tree—the hierarchy—of topics). An illustration of these data structures is shown in Fig. 3.

Each data structure has a set of properties. Thus, a topic is characterised by an identifier (ID), a name (string type), a dimension[4], three flags (New/Modified/Deleted) which

[2] We say that there is a hole in the set of topic identifiers if there are two topics with identifiers *x.i-1* and *x.i+1*. respectively, while there is no topic with identifier *x.i*.

[3] The clusters are made up of stations whose radio proximity allows them to communicate directly with each other or by multiple hops; however, the stations on two different clusters are sufficiently far apart so that they cannot communicate with each other.

[4] *Dimension* property designates a king of counter which is initialized to 0, and is incremented each time a direct sub-topic of the cur-

Algorithm 1 CREATE_TOPIC Creates and adds a new topic to the topic tree

Input: ***ID_Station*** *(the identifier of the station that is creating the new topic),* ***S_Name*** *(the name of the new topic),* ***ID_Topic_P*** *(the identifier of the parent topic of the new topic),* ***A*** *(the topic tree)*
Output: *The topic tree after adding the new topic*

$Node\_P \leftarrow FIND(ID\_Topic\_P, A)$ ▷ retrieve the parent node from the parent topic identifier
if $IS\_BEING\_DELETED(Node\_P.Node)$ **then**
  Throw Error ▷ cannot extend a deleted topic
end if
$S\_ID \leftarrow GENERATE\_NEW\_TOPIC\_ID(Node\_P.Node, ID\_Station)$ ▷ generate the identifier of the new topic
$Node \leftarrow NEW\_NODE(S\_ID, S\_Name, 0, True, False, False, 0)$ ▷ build the new node
$A' \leftarrow ADD\_NODE(Node, Node\_P, A)$ ▷ add the new node to the tree and update the parent node
$Node\_P.Node.Dimension \leftarrow Node\_P.Node.Dimension + 1$
$Node\_P.Node.Updated \leftarrow True$
$Node\_P.Node.New \leftarrow False$
$Node\_P.Node.LastUpdateDate \leftarrow Node\_P.Node.LastUpdateDate + 1$
return A'

indicate whether the topic has been newly created/modified/deleted and finally, a last update date in case the topic has been modified; this date is a logical date. It is set to 0 when the topic is created, and is updated following a principle similar to the Lamport Clock [22] principle: in case of a local modification of a topic *t*, the *LastUpdateDate* of *t* is incremented; when a station receives a modified topic from an other station (lets call *Dr* the *LastUpdateDate* of the received topic and *Dl* the *LastUpdateDate* of the local replica of that topic), then the *LastUpdateDate* of the local replica of the considered topic is calculated using formula 1.

$$Dl = max(Dr, Dl) \tag{1}$$

Similarly, a *TopicTree* is characterised by a property *node*, representing a root topic, and a property *children* which is a possibly empty list of sub-trees representing the direct descendants of the current topic.

Access to the property P of an instance I of one of these data structures is done via the pointed notation I.P. Thus, for a topic T, T.ID designates its identifier.

Updating of the Initiating Station's Topic Tree

As retained in the assumptions, only a station that has created a topic can proceed to its modification and/or deletion.

Adding a New Topic

Adding a new topic involves inserting a new node in the topic tree. To do this, the name of the topic is provided and all other properties are filled in automatically. In order to avoid possible conflicts between topic identifiers created in parallel on different stations, the identifier of the station wishing to create a new topic is used to generate the topic identifier. More details on how to generate new Topic IDs are given in Sect. New Topic Identifiers Generation. The algorithm 1 is executed by the creating station to add a topic to its topic tree. The *FIND*, *CONCAT*, *ADD_NODE* and *NEW_NODE* primitives, respectively, allow to retrieve a node of the tree from its identifier, concatenate a set of values taken in parameters into a resulting string, add a node to a tree as a child of another node all taken in parameters, build a new node from its respective characteristics (ID, Name, Dimension, New, Modified, Deleted, LastUpdateDate). In addition, the *GENERATE_NEW_TOPIC_ID* primitive corresponds to the algorithm for generating the identifier of a new topic from its parent topic and the identifier of the creating station described in Sect. New Topic Identifiers Generation. If a topic is being deleted, i.e., it is still present in the tree with its flag "*Deleted*" or that of one of its hierarchical ancestors set to "*True*", then the primitive *IS_BEING_DELETED* which takes as input the node of the tree corresponding to this topic returns "*True*".

Deleting a Topic

When a topic is deleted, the "*Deleted*" flag for that topic is set to "*True*" and the "*New*" and "*Modified*" flags are set to "*False*". However, all its hierarchical descendant topics remain unchanged although they too are intended to be deleted. Each station maintains a *DeletedTopics* list containing the identifiers of the last *k* permanently deleted topics. This list is nothing more than a buffer useful to recognize these topics (see Sect. Continuous Update of the Stations). The permanent deletion of a topic consists in removing it from the topic tree with its hierarchical descendants and inserting its identifier in the *DeletedTopics* list. It takes place after a latency time set by a parameter named *UPDATE_PROPAGATION_TIMEOUT*. The identifiers of the hierarchical descendants are not added to this list because to recognize a topic which has been permanently deleted, it is enough (thanks to the particular form of the identifiers) to

Footnote 4 (continued)
rent topic is created. It is useful when creating direct descendants of a topic (see Sect. New Topic Identifiers Generation).

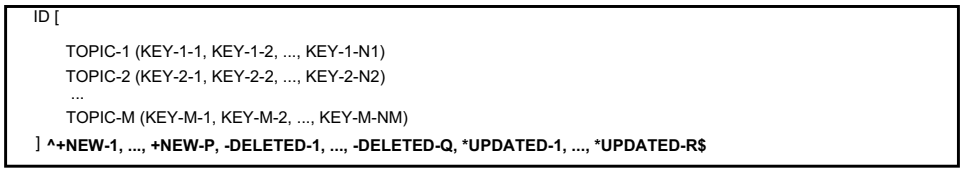

Fig. 4 Format of the modified QUERY message

check if its identifier is present in the *DeletedTopics* list or if it is a descendant of a topic whose identifier is there.

Modifying a Topic

Modifying a topic consists of changing its name. When a topic is modified, the new name replaces the old one and the "*Modified*" flag for that topic is set to "*True*", while the "*New*" flag is set to "*False*". In addition, the *LastUpdateDate* property of the modified topic is simply incremented. Topics that are currently being deleted cannot be modified.

Disseminating Updates to Other Stations in the Network

The ordinary *QUERY* message structure defined in [13] is extended (see Fig. 4) by adding a set of identifiers contained between the '^' and '$' delimiters and separated by commas. Between these delimiters, the identifiers of new topics are preceded by the '+' symbol, those of topics being deleted that have their "*Deleted*" flags set to "*True*" are preceded by the '-' symbol, and finally, the identifiers of modified topics are preceded by the '*' symbol. The identifiers of topics being deleted that do not have their "*Deleted*" flags set to "*True*" are not added to the *QUERY* message, although they are intended to be deleted. This helps to limit the size of this message by avoiding adding redundant information. In fact, since the deletion of a topic leads to the deletion of all its hierarchical descendants, only the knowledge of the ancestor topic with its flag "*Deleted*" set to "*True*" is necessary.

After a network station updates its topic tree (adds, modifies or deletes a topic), all its future announcement messages contain, for a certain time given by the value of the parameter *UPDATE_PROPAGATION_TIMEOUT*, the information on the updated topics. When the *UPDATE_PROPAGATION_TIMEOUT* runs out, all the '*Modified*' and '*New*' flags of all topics in the tree are reset to '*False*'; the identifiers of topics with the '*Deleted*' flag set to '*True*' are added to the *DeletedTopics* list of the local station, and these and all their hierarchical descendants are removed from the topic tree.

When a station in the network receives the *QUERY* message, it updates its topic tree by setting the "*Deleted*" flags of all topics in the tree that are not being deleted and whose identifiers are included among the identifiers framed by the '^', and '$' symbols and preceded by the '-' symbol, to "*True*". Then it sends in unicast, an *UPDATE_REQUEST* request to the station sending the *QUERY* message previously received, with the identifiers framed in this *QUERY* message by the symbols '^', and '$' and preceded by the symbols '+' and '*'.

If we note N (respectively M) the set of identifiers contained in the *QUERY* message and preceded by the symbol '+' (respectively '*'); if we note S the set of topics in the topic tree of the station receiving the *QUERY* message and Hs the set of hierarchical ancestor topics of a topic s ($s \in Hs \subset S$) then, the identifiers of the topics sent in the *UPDATE_REQUEST* are those given by formula (2); they are in fact the identifiers preceded by the symbol '+' in the announcement message and which are not identifiers of topics in the topic tree of the station receiving the *QUERY* message.

$$N - \{s.ID, s \in S\} \tag{2}$$

Similarly, the identifiers preceded by the symbol '*' are sent in the *UPDATE_REQUEST* if they are not those of the topics being deleted in the topic tree of the station receiving the *QUERY* message; they, therefore, belong to the set given by formula (3).

$$M - \{s.ID, s \in S / \exists a \in Hs, a.Deleted = True\} \tag{3}$$

Topic identifiers that fall between the '*[*' and '*]*' symbols and are not present in the topic tree are also added to the *UPDATE_REQUEST* message.

When the station that sent the *QUERY* message receives the *UPDATE_REQUEST* request, it constructs an *UPDATE_REPLY* message (see Fig. 5) with a complete description of each topic whose identifier is contained in the *UPDATE_REQUEST* request. This message is then broadcast in the network after an *UPDATE_REPLY_BACK_OFF* delay. By analogy with the back-off delay preceding the broadcast of publications in SocialMANET, this allows to satisfy several additional *UPDATE_REQUEST* requests at once and thus to save resources.

The algorithm 2 allows a station to update its topic tree after receiving an *UPDATE_REPLY* message. Arranging the topics in lexicographical order on the identifiers makes it possible to walk through the tree level by level (Breadth First), from the root to the leaves, thus ensuring that the creation of a new topic implies that its parent topic exists. To do this, only the *ID-INFO* part (see Sect. New Topic

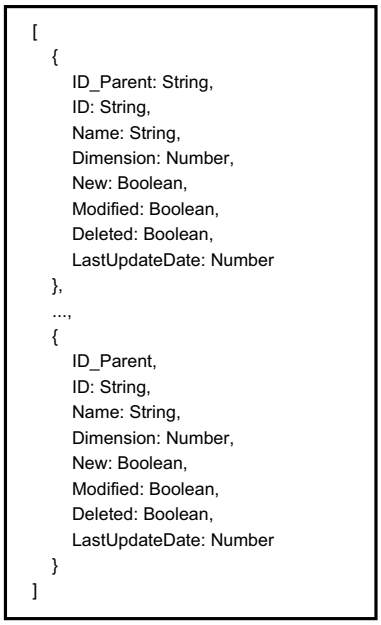

Fig. 5 Format of the UPDATE_REPLY message

Identifiers Generation) of the topic identifier is taken into account to perform the ordering. The *FIND*, *ADD_NODE* and *NEW_NODE* primitives are as defined in Sect. "Updating of the Initiating Station's Topic Tree".

If a topic is deleted, its identifier or that of one of its hierarchical ancestors is in the *DeletedTopics* list; then the *IS_DELETED* primitive that takes a topic as input returns "*True*" and the corresponding iteration for that topic is skipped (Algorithm 2, lines 4–6).

It may happen that a station receives a topic that has undergone a modification on a different date than the modification it knows of the same topic; the algorithm 2 allows to keep the most recent modification by using the logical date (see Algorithm 2, lines 10-17).

If the *UPDATE_REPLY* message is a reply to a previous *REFRESH_REQUEST* message received (see Sect. "Continuous Update of the Stations"), it may contain a topic that is being deleted (*Deleted* = *True*) according to the message *UPDATE_REPLY*, but not from the point of view of the current station's topic tree. The algorithm 2 (lines 18-22)

Input: ***TLIST*** *(the list of topics contained in the UPDATE_REPLY message),* ***A*** *(the topic tree)*
Output: *The updated topic tree*

1: $A' \leftarrow A$
2: Arrange $TLIST$ topics in the lexicographical order on the identifiers
3: **for all** $s \in TLIST$ **do**
4: **if** $IS\_DELETED(s)$ **then**
5: Continue
6: **end if**
7: $NodeEx \leftarrow FIND(s.ID, A')$
8: **if** $NodeEx \neq Null$ **then**
9: **if** $s.Modified = True$ **then** ▷ modified topic
10: **if** $NodeEx.Node.LastUpdateDate < s.LastUpdateDate$ **then**
11: $NodeEx.Node.Name \leftarrow s.Name$
12: $NodeEx.Node.Dimension \leftarrow s.Dimension$
13: $NodeEx.Node.New \leftarrow False$
14: $NodeEx.Node.Modified \leftarrow True$
15: $NodeEx.Node.Deleted \leftarrow False$
16: $NodeEx.Node.LastUpdateDate \leftarrow s.LastUpdateDate$
17: **end if**
18: **else**
19: $NodeEx.Node.New \leftarrow False$
20: $NodeEx.Node.Modified \leftarrow False$
21: $NodeEx.Node.Deleted \leftarrow True$
22: **end if**
23: **else** ▷ new topic for the current station
24: $Node\_P \leftarrow FIND(s.ID\_Parent, A')$
25: $NNode \leftarrow NEW\_NODE(s.ID, s.Name, s.Dimension, s.New,$
26: $s.Modified, s.Deleted, s.LastUpdateDate)$
27: $A' \leftarrow ADD\_NODE(NNode, Node\_P, A')$
28: **end if**
29: **end for**
30: **return** A'

Algorithm 2 **HANDLE_UPDATE_REPLY**

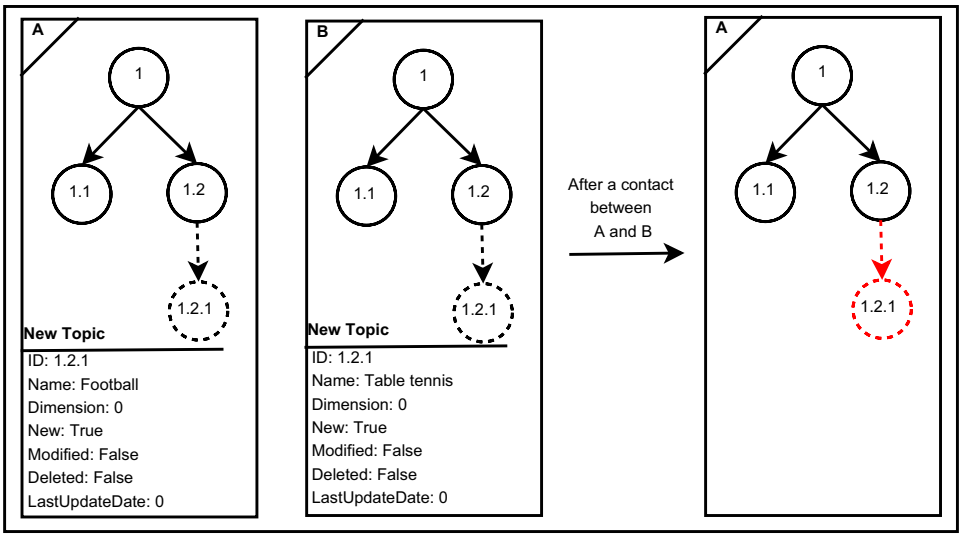

Fig. 6 Conflict of identifiers when creating new topics

Algorithm 3 GENERATE_NEW_TOPIC_ID Generates an ID for a new topic

Input: ***P_TOPIC*** *(the parent topic of the new topic),* ***ID_STATION*** *(the identifier of the station that is creating the new topic)*
Output: *The identifier of the new topic*
1: $CREATORS \leftarrow Null$
2: $IDS \leftarrow Null$
3: **if** P_TOPIC.ID contains "::" **then**
4: $\quad CREATORS \leftarrow CONCAT(ID\_STATION, " \sim ", CREATOR\_INFO(P\_TOPIC.ID))$
5: $\quad IDS \leftarrow CONCAT(ID\_INFO(P\_TOPIC.ID), '.', P\_TOPIC.Dimension + 1)$
6: **else**
7: $\quad CREATORS \leftarrow ID\_STATION$
8: $\quad IDS \leftarrow CONCAT(P\_TOPIC.ID, '.', P\_TOPIC.Dimension + 1)$
9: **end if**
10: $ID \leftarrow CONCAT(CREATORS, " :: ", IDS)$
11: **return** ID

handles this situation by allowing the current station to update its topic tree properly.

New Topic Identifiers Generation

As mentioned in Sect. "Overview of the SocialMANET Protocol", topic identifiers are generated using the Dewey identification technique. However, this technique shows its limitations when network stations can create new topics concurrently, as there is a risk that the topic identifiers created concurrently clash with each other. If this were to happen, it would be impossible to distinguish between each of the conflicting topics.

For example, consider a publish/subscribe system for disseminating information during university games, and two stations A and B each adding a topic to their respective trees, which were previously identical. Let us assume that each of the new topics (e.g., *Football* and *Table tennis*) is inserted in the respective trees of stations A and B as a child of the topic of identifier 1.2 (e.g., *Sport*); these new topics will therefore, following Dewey's identification technique, have identifier 1.2.1 in their respective topic trees (see Fig. 6). During a contact between stations A and B, station A will receive identifier 1.2.1 from B as a new topic, but it will do nothing because it will assume that it already has this topic in its local tree (see Fig. 6).

To avoid errors of this type when updating the topic trees of the network stations after a topic has been added, we have proposed a new way of assigning identifiers to the new topics.

We have introduced a new data in the topics identifiers; it contains information about the creator of the topic. The resulting format for a topic identifier is then as follows: *<CREATOR-INFO>*::*<ID-INFO>*. The part *CREATOR-INFO* is a sequence of identifiers containing, from left to right, the identifier of the station that created the current topic, as well as those of the stations that created the ancestor topics of the current topic from the closest to the furthest hierarchical distance, respectively, separated by the symbol '∼'. The *ID-INFO* part represents the topic identifier obtained by applying the Dewey identification technique. The algorithm 3 shows how the identifier of a new topic is generated from the identifier of the parent topic and the one

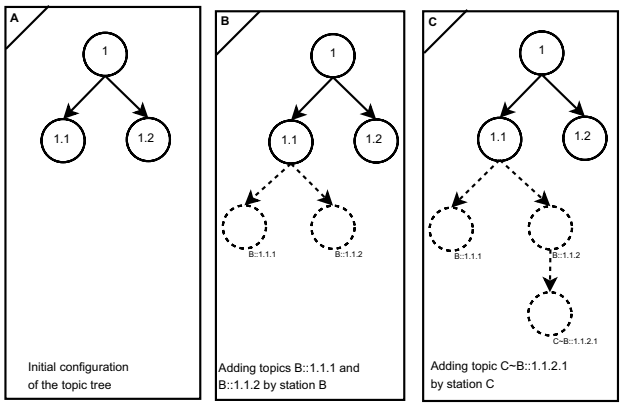

Fig. 7 Illustration of the algorithm for generating identifiers of new topics

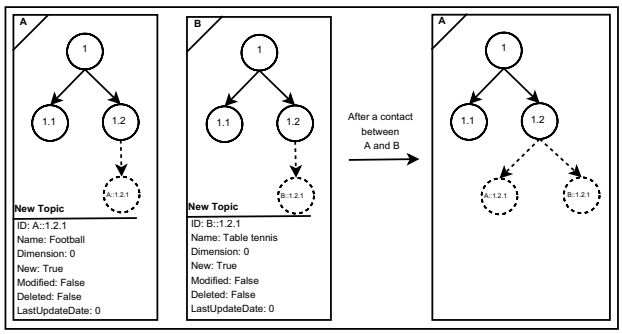

Fig. 8 Resolution of the conflict of topics identifiers illustrated in Fig. 6

of the creating station. Examples of identifiers are given in Fig. 7.

The *CREATOR_INFO* and *ID_INFO* primitives are used to extract the *CREATOR-INFO* and *ID-INFO* parts of a topic identifier, respectively. Figure 7 illustrates this technique.

Figure 8 is an illustration of the use of algorithm 3 to solve the conflict of identifiers problem illustrated in Fig. 6. Since each of the stations A and B has added its identifier to the topic it has created, station A will recognise the topic of identifier *B::1.2.1* created by station B as a new topic.

Continuous Update of the Stations

When the *UPDATE_PROPAGATION_TIMEOUT* delay of a station expires, the propagation of updates stops for all relevant topics. This can happen when some stations out of range have not been updated yet. This sub-section aims to propose a solution to allow such stations to be informed of changes in the topic tree during their "*unavailability*" (while out of range).

The proposed solution once again exploits the periodic *QUERY* announcement messages that the network stations broadcast to signal their presence and the topics to which they have subscribed (see Fig. 1). In fact, from a *QUERY* message sent by a station, the receiving station can deduce certain topics of which it is not aware and request them later via an *UPDATE_REQUEST*.

Indeed, as mentioned in Sect. "Disseminating Updates to Other Stations in the Network", if a station receives a *QUERY* message and realizes that some topic identifiers between the '*[*' and '*]*' symbols are not present in its topic tree, it adds these identifiers to the contents of its *UPDATE_REQUEST* message. This allows stations that have not been updated until the *UPDATE_PROPAGATION_TIMEOUT* delay from other stations has expired to receive updates on topics that were created while they were out of range.

The question could be asked what happens then to modified and deleted topics; how can stations be updated against these cases after the propagation time of the updates has expired? For this purpose, we introduce a new parameter in the protocol called *AWARENESS_THRESHOLD* which represents at a given moment the maximum acceptable value of the time elapsed between the last update of a station and the current moment. When this threshold is exceeded, at the next contact with the other stations in the network, the stations concerned broadcast a *REFRESH_REQUEST* message in the network, containing all the topics they have in their topic trees. In this case, the stations will have to store the date of last update of their topic trees. This allows all stations receiving this query to broadcast, using the *UPDATE_REPLY* message, a full description of all the topics contained in their topic trees, which are not included in the *REFRESH_REQUEST* message. In addition, the *UPDATE_REPLY* message also includes all topics in the *REFRESH_REQUEST* message for which at least one of the following conditions is met: (1) the topic's "*Deleted*" flag in the local topic tree is set to '*True*', but not in the *REFRESH_REQUEST* message; (2) the date the topic was last modified in the local topic tree is greater than the date the topic was last modified in the *REFRESH_REQUEST* message. This will result in an update of the stations that have broadcast such a *REFRESH_REQUEST* message. For topics contained in the *REFRESH_REQUEST* message whose identifiers are present in the local *DeletedTopics* list, a *FORCE_DELETE* broadcast message is sent with these identifiers. Any station receiving the *FORCE_DELETE* message copies the identifiers it contains in its *DeletedTopics* list and systematically removes from their topic tree the topics whose identifiers are contained within this message, as well as their hierarchical descendants.

The *REFRESH_REQUEST* message can also inform the receiving station that some topics of which it is not aware

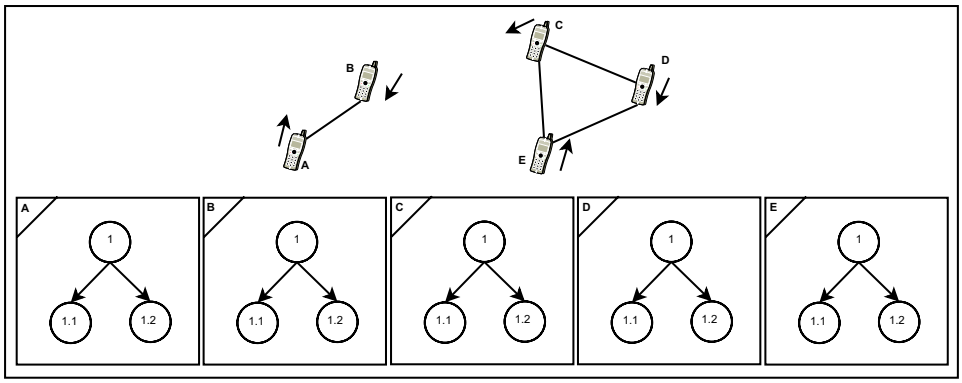

Fig. 9 Initial state of the stations: identical topic tree and two clusters

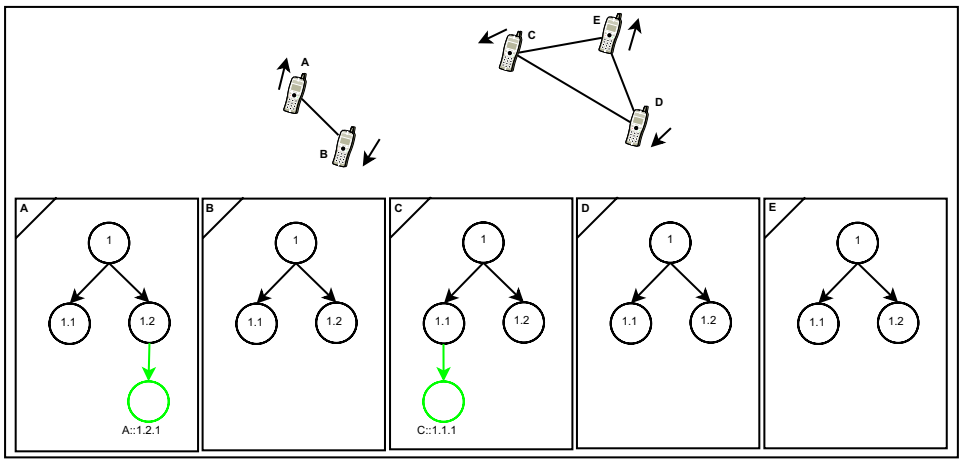

Fig. 10 Addition of two new topics by stations A and C

exist. These are the topics absent in its topic tree and present in the *REFRESH_REQUEST* message. In this case, the receiving station will systematically update its topic tree, if their identifiers are not present in its local *DeletedTopics* list. The receiving station can also exploit this message for modified topics by updating topics in its topic tree whose last modification date in the *REFRESH_REQUEST* message is greater than the last modification date in the local topic tree.

Sample Case: Illustration of the Protocol

We present in this section an illustration of the proposed protocol. One can imagine being in a natural disaster area (e.g. a earthquake) with teams of rescue workers going here and there and having a mobile application to disseminate information by publish/subscribe among them. The topics in such an application could be *Fire* (so that the interested parties are fire-fighters), *Injured* (so that the interested parties are emergency workers), etc. We will consider five stations A, B, C, D and E respectively to run our example. These stations will represent rescue workers, or more precisely instances of the mobile application (installed in their smartphones for example) that they use to communicate. We will show the evolution of the network topology over time as the stations move, as well as the evolution of the topic trees structure for each node in the network.

Table 1 QUERY messages broadcast by network stations (1)

Source	Content	Targets
A	^+A::1.2.1$	B
C	^+C::1.1.1$	D, E

Figure 9 shows the initial status (after the deployment) of the stations in the network. The topic tree at all stations is the initial tree (no modifications, deletions or additions have been made). For the sake of simplicity, only the IDs of topics have been displayed. A solid line connecting two stations means that they can communicate with each other. The arrows symbolize the movements of the stations.

Continuing the illustration, it is assumed that stations A and C add topics *A::1.2.1* and *C::1.1.1* respectively (Fig. 10). For the sake of simplicity, the QUERY messages shown in Table 1 following these additions are only partial representations of the actual *QUERY* messages. It can be noticed that station B learned from A the creation of topic *A::1.2.1*; similarly stations D and E learned from C the creation of topic *C::1.1*. After

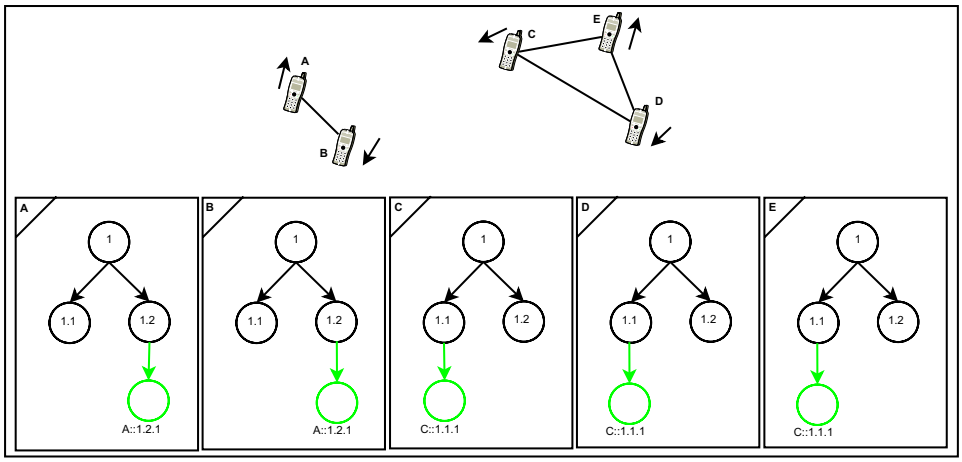

Fig. 11 Updating stations B, D and E

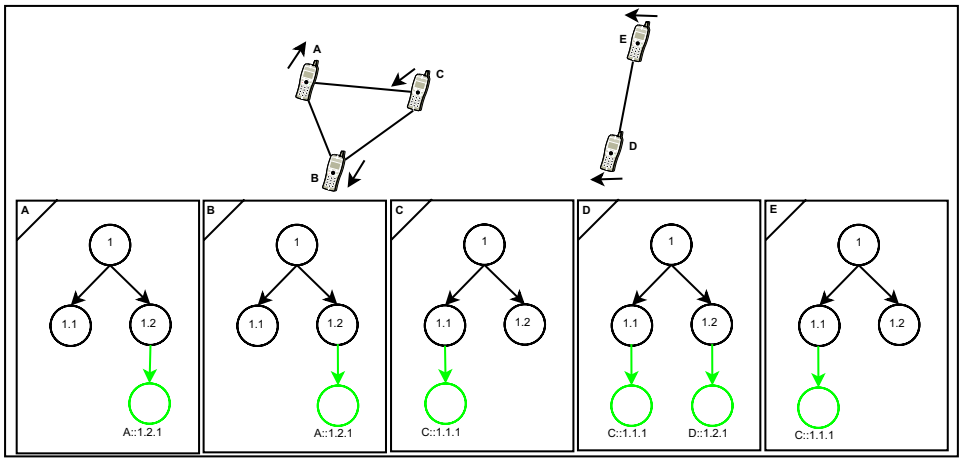

Fig. 12 Creating a new topic with identifier D::1.2.1

Table 2 QUERY messages broadcast by network stations (2)

Source	Content	Targets
A	^+A::1.2.1$	B, C
B	^+A::1.2.1$	A, C
C	^+C::1.1.1$	A, B
D	^+C::1.1.1, +D::1.2.1$	E
E	^+C::1.1.1$	D

a few exchanges between these stations to obtain the details of the created topics (for instance, exchanges of *UPDATE_REQUEST* and *UPDATE_REPLY* messages), stations B, D and E will update their respective topic trees (see Fig. 11).

Continuing the illustration, it is assumed that station D has created a new topic with identifier *D::1.2.1* and that station C has joined the cluster containing A and B (Fig. 12).

Table 2 gives an overview of the contents of the *QUERY* messages sent by the different stations. After exchanges of *UPDATE_REQUEST* and *UPDATE_REPLY* messages between stations in the network, the updates of the respective topic trees of different stations are shown in Fig. 13. Let us observe in that figure that, stations A, B and C now have the new topics *A::1.2.1* and *C::1.1.1* in their topic trees; similarly, stations D and E have the new topics *C::1.1.1* and *D::1.2.1* in their topic trees.

In order to continue the illustration, let us consider the new network topology shown in Fig. 13, in which new connections are created; this give rise to new exchanges (see Table 3) between the stations of the network. These exchanges allow them to update their topic trees, as can be seen in Fig. 14.

In the same figure (Fig. 14), it can once again be seen that the moving of the stations has once again resulted in a new network configuration. The modification of topic *A::1.2.1* by station A and the deletion of topic *C::1.1.1* by station C can also be noted. All these changes also lead to a change in the contents of the *QUERY* messages broadcast by the stations in the network (Table 4). These different contents inform that station E will be aware of

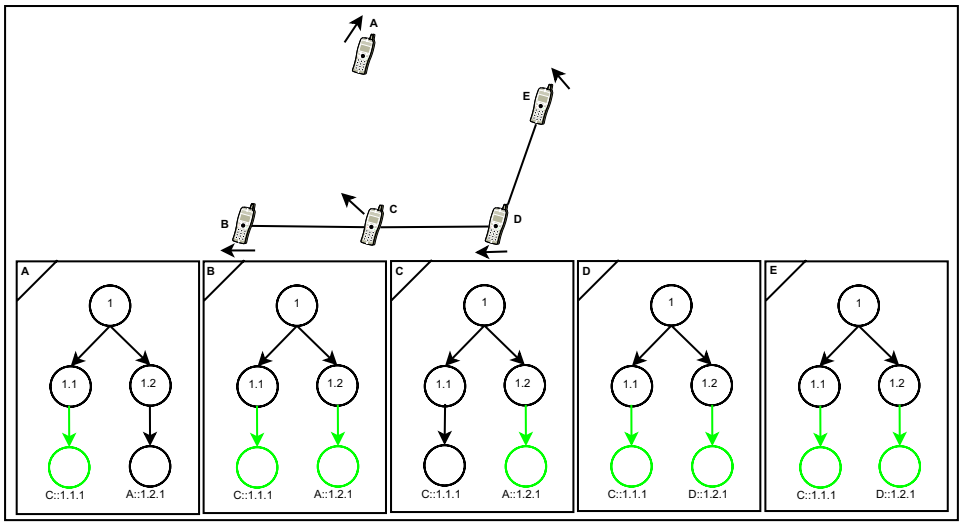

Fig. 13 New update of the network stations

Table 3 QUERY messages broadcast by network stations (3)

Source	Content	Targets
A	^+C::1.1.1$	-
B	^+A::1.2.1, +C::1.1.1$	C
C	^+A::1.2.1$	B, D
D	^+C::1.1.1, +D::1.2.1$	C, E
E	^+C::1.1.1, +D::1.2.1$	D

the modification of the topic *A::1.2.1* that it does not have in its topic tree, which will allow it to update itself. Similarly, stations A and B will be aware of the creation of topic *D::1.2.1*; station B will also be aware of the deletion of topic *C::1.1.1*. They will then be able to update their topic trees and continue the propagation of the different information collected on the changes in the topic tree.

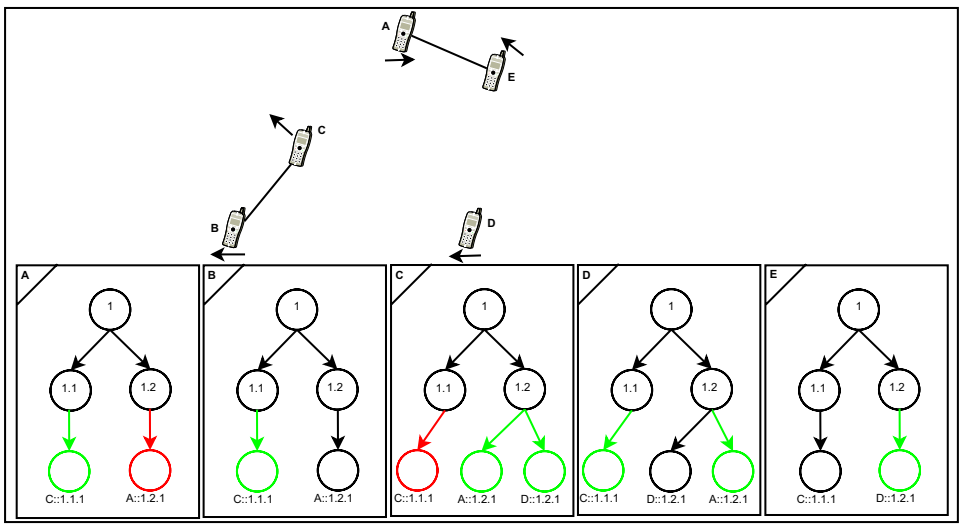

Fig. 14 New status of topic trees, modification of topic A::1.1.1 and deletion of topic C::1.1.1.1

Table 4 QUERY messages broadcast by network stations (4)

Source	Content	Targets
A	^+C::1.1.1, *A::1.2.1$	E
B	^+C::1.1.1$	C
C	^+A::1.2.1, +D::1.2.1, -C::1.1.1$	B
D	^+A::1.2.1$	–
E	^+D::1.2.1$	A

Table 5 Simulation parameters

Parameter	Value
MAC layer	802_11b
Routing protocol	None
Bandwidth	1Mb/s
Detection range	50 m
Transmission range	50 m
Number of stations	100
Simulation area	950 m * 800 m
Mobility model	RWM
Maximum speed	4 m/s
Pause time	2 s
Duration of simulation	900 s

Performances

We performed simulations to evaluate the performances of the proposed protocol in terms of the propagation speed of the topic tree updates, and the effective update rate of the different stations in the network. The simulations were carried out using the NS2 simulator ("NS-2 Network Simulator" http://www.isi.edu/nsnam/ns/).

Simulation Parameters

Simulations were made with a set of 100 nodes deployed over an area of 950 m x 800 m, moving according to the Random Waypoint Mobility model (RWM). The overall parameters of the simulation are summarised in Table 5.

The RWM mobility model makes it possible to make the movement of nodes in the network random. Throughout the simulation, each node randomly selects a destination, moves there at a random speed below a preconfigured value, remains stationary for a so-called *pause time*, then selects a new destination and the cycle starts again. We have set the maximum speed of the nodes at 4 m/s and the pause time at 2 s.

We realized an implementation of the SocialMANET protocol within the NS2 simulator and we grafted to it an implementation of the dynamic topics management protocol presented in this paper. Unless otherwise stated, all simulations were performed under normal operating conditions of SocialMANET; i.e., stations subscribed and published information during the simulations and the different publications were disseminated among stations in the network.

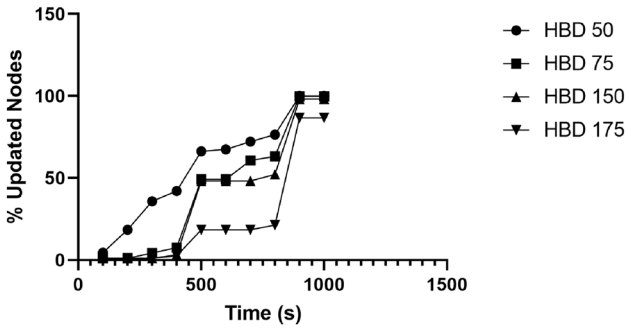

Fig. 15 Update propagation speed

Results

Update Propagation Speed

During this simulation, which lasted 1000 s, we set the value of the seed[5] of the random number generator to 0. Each station is activated at the beginning of the simulation; the activation here consists of launching the first iteration of the SocialMANET protocol's *Need Detection* phase (the other iterations follow). We have also set the value of the *UPDATE_PROPAGATION_TIMEOUT* parameter to 50 s, that of the *AWARENESS_THRESHOLD* parameter to 200 s and the number of altruistic stations[6] to 0.

Initially, the topic trees for all stations in the network contain five topics. At the very beginning of the simulation, a new topic is created by one station in the network. We have identified the number of stations that have updated their topic trees with the new topic created for different values of the *RESEARCH_DELAY*[7] parameter (50, 75, 150 and 175) over time. The results are shown in Fig. 15.

It can be noticed that for small values of the *RESEARCH_DELAY (HBD)* parameter, the propagation speed of the updates is high: more than half of the stations in the network had been updated in the middle of the simulation. It can also be

[5] This is a number which is used by the simulator to generate the random numbers generally obtained using the *rand()* function.

[6] Altruistic stations participate in the dissemination of publications belonging to topic they have not subscribed to [13].

[7] The *RESEARCH_DELAY* parameter represents for the SocialMANET [13] protocol the time out between two consecutive broadcasts of the *QUERY* announcement message.

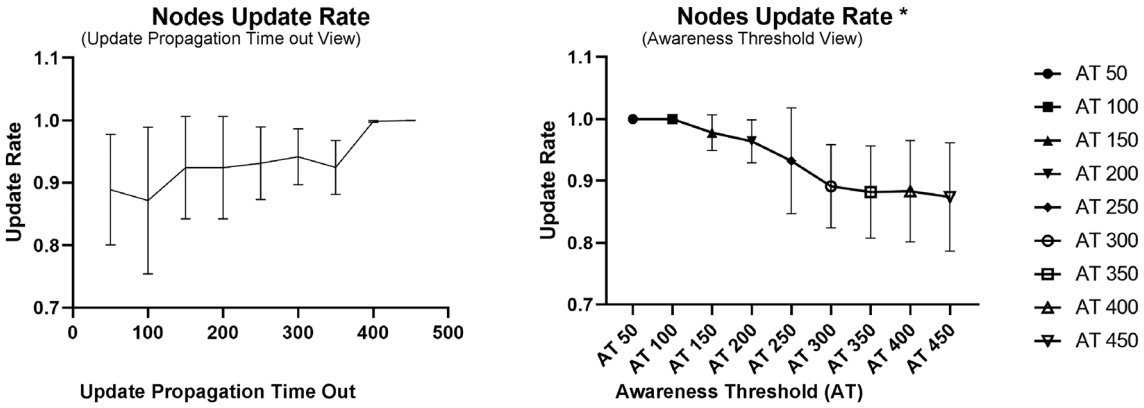

Fig. 16 Network stations update rate

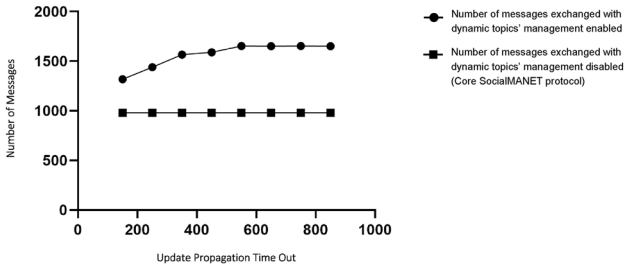

Fig. 17 Network load in terms of number of exchanged messages

seen that, whatever the case, all or almost all the stations in the network are actually updated before the end of the simulation.

Network Stations Update Rate

For this second 1000-s simulation, we have also set the value of the seed of the random number generator to 0, with each station being activated at the beginning of the simulation. We have set the value of the *RESEARCH_DELAY* parameter to 50 s and the number of altruistic nodes to 0.

Initially, the topic trees for all stations in the network contain five topics. Six more topics are created at the 50th, 150th, 175th, 225th, 250th and 400th second of the simulation, respectively. We have played this simulation several times by varying the parameters *UPDATE_PROPAGATION_TIMEOUT* and *AWARENESS_THRESHOLD* from 50 to 450 in steps of 50. Figure 16(a) shows for each value of the parameter *UPDATE_PROPAGATION_TIMEOUT*, the mean value and the error of the network stations update rate for the different values of the *AWARENESS_THRESHOLD* parameter.

It can be noticed that the update rate of the stations in the network is quite high (mostly above 0.9 (90%)) and tends towards a rate of 1.0 (100%) when the value of the *UPDATE_PROPAGATION_TIMEOUT* parameter reaches 400 s (Fig. 16(a)). Note the impact of the *AWARENESS_THRESHOLD* (AT) parameter which, as its value increases (Fig. 16(b)), has a degrading effect on this rate (although not away from 90%).

Number of Exchanged Messages

We carried out a simulation to evaluate the number of messages exchanged in the network related to the updating of the stations' topic trees. We compared the results obtained to the ones of the core SocialMANET protocol (when the dynamic topics' management is disabled). We ran this simulation three times for different values of the seed of the random number generator, namely -1, 0 and 1. At each run, the duration of the simulation was 1000 s. We then used the average of the results obtained to get the representation in Fig. 17.

Each station is activated at a random time during the first 450 s of the simulation. We have also set the value of the *RESEARCH_DELAY* parameter to 150 s and the number of altruistic nodes to 0. Initially, the topic trees of all stations in the network contain five topics. Six more topics are created at the 50th, 150th, 175th, 225th, 250th and 400th second of the simulation, respectively.

Table 6 Ratio of the number messages exchanged

Update propagation time-out	150	250	350	450	550	650	750	850
Ratio	1.32	1.44	1.57	1.59	1.66	1.65	1.66	1.65

Table 7 Comparison of the solutions

Features	Our solution	Kafka [7]	SQS [4]	Pub/Sub [5]	MQ [6]	Rowstron et al. [10]	Shrideep et al. [11]	Nicklas et al. [12]
Distributed?	Yes	No	No	No	No	Yes	Yes	No
Suitable for MANETs?	Yes	No	No	No	No	No	No	No
Supported update operations	Add, Modify, Delete	Add, Modify, Delete	Add, Modify, Delete	Add, Delete	Add, Delete	Add	Add	Add
Where are the topics stored?	All the stations	The server(s)	The server(s)	The server(s)	The server(s)	The Station whose Id is close to that of the topic	TDNs track and provide data related to topics	The server

By varying the value of the parameter called *UPDATE_PROPAGATION_TIMEOUT*, we measured the number of messages exchanged in the network. The results are shown in Fig. 17. On the *x*-axis, we have the variation of the *UPDATE_PROPAGATION_TIMEOUT* parameter and on the *y*-axis the number of messages exchanged in the network. When the dynamic topics' management is not active (Core SocialMANET protocol), the number of messages exchanged is constant. This number increases when dynamic topic management is enabled, as the value of the *UPDATE_PROPAGATION_TIMEOUT* parameter increases. This increase is due to exchanges to propagate the updates of station topic trees.

Table 6 presents the ratio of the number of messages exchanged with the dynamic topics' management enabled (*#MSG_DTM*) over the number of messages exchanged in the case of the core SocialMANET protocol (*#MSG_CSM*). This ratio is obtained by the formula 4.

$$\text{Ratio} = \frac{\#MSG\_DTM}{\#MSG\_CSM} \tag{4}$$

Viewing the results from this angle, we can see that the introduction of dynamic topic management within the SocialMANET protocol increases the number of messages exchanged in the network from 1.32 to 1.66 times, depending on the value of the *UPDATE_PROPAGATION_TIMEOUT* parameter.

Discussion

When network stations communicate with a high frequency (low values of the *RESEARCH_DELAY* parameter), the propagation speed of topic updates is also high. This is due to the relatively high frequency of *QUERY* announcement messages due to the low value of the *RESEARCH_DELAY* parameter.

We noticed a degrading effect on the station update rate due to the *AWARENESS_THRESHOLD* parameter. The higher the value of the *AWARENESS_THRESHOLD* parameter, the more the stations tolerate not receiving any updates, resulting in a lower update rate for the stations in the network. Nevertheless, this rate is high overall (mostly above 90%). This is in line with what we expected; it proves that our strategy is effective not only in terms of local changes, but also in terms of global changes in MANETs. Indeed, the techniques we use ensure that almost all stations are updated, as even stations that have been out of range for a long time can receive updates even though the propagations have been completed at the other stations in the network. Also, the target and source stations of a refresh request can both be updated using the same refresh request (*REFRESH_REQUEST message*). Finally, changes to the set of topics can be detected even during normal SocialMANET protocol communications using the classic *QUERY* announcement message in which a station can detect an identifier of a topic it is not aware of.

Updating the set of system's topics is costly in terms of the number of messages exchanged in the network, as shown by the results of the simulations presented in Sect. "Results". This could consequently lead to over-consumption of energy by the stations. Therefore, when designing a topic-based publish/subscribe system for MANETs, the set of topics should be well defined in order to minimise the possible changes that could occur to it. In addition, the values of the parameters *RESEARCH_DELAY*, *UPDATE_PROPAGATION_TIMEOUT* and *AWARENESS_THRESHOLD* must be chosen in such a way as to minimise the number of messages exchanged in the network without significantly altering the propagation speed and the effective update rate of the stations in the network.

The problem of dynamic topics management in topic-based publish/subscribe systems is an interesting one. However, not much work addresses it in the literature. To our knowledge, it has no solution capable of running in a

MANET. Our solution is designed to cope with the constraints imposed by this type of network. Table 7 presents a summary of the characteristics of our solution and those found in the literature [4–7, 10–12].

The update operations that can be performed on the set of topics in a topic-based publish/subscribe system are the *addition*, *modification* and *deletion* of topics. All of the solutions presented in Table 7 addressed the addition of new topics, but it should be noted that only the solutions presented in [4–7] addressed the deletion of topics in addition, and those presented in [4, 7] addressed the modification of topics. Our solution manages all possible updating operations. In addition, our protocol suggests that each station on the network has a copy of all topics, which is not the case for the other distributed solutions presented in Table 7. This has the advantage that no topic search operation (which can be costly in distributed systems) is necessary as in [11] to subscribe to a topic or perform a publication on a topic. Indeed, since each station stores all the topics in the system, a simple reading of the local set of topics is necessary. However, for this to be possible, stations need to communicate regularly to ensure consistency of local copies. But this is a worthwhile effort for systems operating in MANETs where it is impossible to envisage a resource being stored only by a subset of the stations that can access it, thus running the risk of making the resource inaccessible if the stations storing it are kept out of reach for long time.

Conclusion

In this paper, by extending the SocialMANET protocol for information dissemination using publish/subscribe in MANETs, we have proposed an approach to dynamically manage the topics in a topic-based publish/subscribe system for MANETs. Our approach allows to track any local changes that may occur across the topics in such a system and to disseminate these changes to all the other stations in the network, including those kept out of reach for long periods of time due to network discontinuity. Thanks to the use of an ingenious topic identification technique based on the Dewey identification technique, our approach makes it possible to avoid possible conflicts among the identifiers of topics created by different stations concurrently. The results of the simulations we have conducted show that our approach allows fast propagation of the changes occurring on the set of topics if there is frequent communication among stations, with a high update rate of the stations in the network.

Although we have obtained satisfactory results from the simulations we have conducted, we plan to proceed in a near future with real life experiments on our protocol. To this end, we have initiated the implementation of our protocol within a publish/subscribe system for information dissemination during a scientific conference, over Android smartphones. This experimentation will allow us to further validate our approach.

The number of messages exchanged in the network for the update operations of the topics is not negligible and consequently leads to an over consumption of energy. However, we have given recommendations to limit this effect. Future work could consist of carrying out simulations in order to find the right values for the parameters *RESEARCH_DELAY*, *UPDATE_PROPAGATION_TIMEOUT* and *AWARENESS_THRESHOLD* to be chosen to limit the number of messages exchanged in the network.

Funding No funds, grants, or other support was received.

Code Availability Not applicable.

Availability of Data and Materials The datasets generated during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of Interest The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Patrick TE, Pascal F, Rachid G, Anne-Marie K. The many faces of publish/subscribe. ACM Comput Surv. 2003;35(2):114–31. https://doi.org/10.1145/857076.857078.
2. Baehni S, Chhabra CS, Guerraoui R. Frugal event dissemination in a mobile environment. In: Alonso G, editor. Middleware 2005. Berlin: Springer; 2005. p. 205–24. https://doi.org/10.1007/11587552_11.
3. Baldoni R, Contenti M, Virgillito A. In: Schiper, A., Shvartsman, A.A., Weatherspoon, H., Zhao, B.Y. (eds.) The Evolution of Publish/Subscribe Communication Systems, pp. 137–141. Springer, Berlin (2003). https://doi.org/10.1007/3-540-37795-6_25
4. SQS A. Amazon Simple Queue Service. https://aws.amazon.com/sqs. Accessed 15 Mar 2022.(2022)
5. Pub/Sub: Cloud Pub/Sub (Google Cloud) (2022). https://cloud.google.com/pubsub. Accessed 21 Feb 2022.
6. MQ: MQ - Overview (IBM) (2022). https://www.ibm.com/products/mq. Accessed 21 Feb 2022 (2022)
7. Foundation A. Apache Kafka (2023). https://kafka.apache.org/. Accessed 09 Feb 2023.
8. MQ R. RabbitMQ: easy to use, flexible messaging and streaming—RabbitMQ (2023). https://www.rabbitmq.com/. Accessed 09 Feb 2023.
9. Hat R. Red Hat AMQ (2023). https://www.redhat.com/en/technologies/jboss-middleware/amq. Accessed 09 Feb 2023.
10. Rowstron A, Kermarrec A-M, Castro M, Druschel P. Scribe: The design of a large-scale event notification infrastructure. In: Crowcroft J, Hofmann M, editors. Networked Group Communication. Berlin: Springer; 2001. p. 30–43. https://doi.org/10.1007/3-540-45546-9_3.
11. Shrideep P, Geoffrey F, Harshawardhan G. On the secure creation, organisation and discovery of topics in distributed

publish/subscribe systems. Int J High Perform Comput Netw. 2008;5(3):156–67. https://doi.org/10.1504/IJHPCN.2008.020860.
12. Lisa N, Pullen JM, Corner D. Dynamic publish / subscribe topics in the scripted bml server. In: Spring Simulation Interoperability Workshop 2011, Boston, Massachusetts, USA, April 4–8, 2011, Proceedings, pp. 268–273 (2011)
13. Tchendji MT, Tchembé MX, Necheu IT. Socialmanet: a publish/subscribe events dissemination protocol for mobile ad-hoc networks. J Comput Sci. 2019;15(9):137–41. https://doi.org/10.3844/jcssp.2019.1237.1255.
14. Baldoni R, Beraldi R, Quema V, Querzoni L, Tucci-Piergiovanni S. TERA: topic-based event routing for peer-to-peer architectures. In: Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems. DEBS '07, pp. 2–13. ACM, New York, NY, USA (2007). https://doi.org/10.1145/1266894.1266898
15. Julien H, Frédéric G. A protocol for content-based communication in disconnected mobile ad hoc networks. Mob Inf Syst. 2010;6(2):123–54. https://doi.org/10.3233/MIS-2010-0096.
16. Zhao Y, Kim K, Venkatasubramanian N. Dynatops: A dynamic topic-based publish/subscribe architecture. In: Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems. DEBS '13, pp. 75–86. Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2488222.2489273
17. Bainomugisha E, Paridel K, Vallejos J, Berbers Y, De Meuter W. Flexub: dynamic subscriptions for publish/subscribe systems in magnets. In: Göschka KM, Haridi S, editors. Distributed applications and interoperable systems. Berlin: Springer; 2012. p. 132–9. https://doi.org/10.1007/978-3-642-30823-9_11.
18. Canas C, Zhang K, Kemme B, Kienzle J, Jacobsen H-A. Evolving pub/sub subscriptions for multiplayer online games: Demo. In: Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems. DEBS '16, pp. 344–347. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2933267.2933297
19. Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui R, editor. Middleware 2001. Lecture Notes in Computer Science. Berlin: Springer; 2001. p. 329–50. https://doi.org/10.1007/3-540-45518-3_18.
20. Pullen M, Corner D, Singapogu S. Scripted battle management language web service version 1.0: Operation and mapping description language. In: Proceedings of a Meeting Held 23-27 March 2009, San Diego, California, pp. 327–336 (2009)
21. Mitra P. Dewey decimal system. In: Liu L, Özsu MT, editors. Encyclopedia of database systems. Boston: Springer; 2009. p. 808–9. https://doi.org/10.1007/978-0-387-39940-9_877.
22. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun ACM. 1978;21(7):558–65. https://doi.org/10.1145/359545.359563.