Designing and prototyping an event-based communication system on mobile ad hoc networks

Conception et prototypage d'un système d'événements pour réseaux mobiles ad hoc

Amir R. Khakpour
Isabelle Demeure

2008D009

Juillet 2008

Département Informatique et Réseaux
Groupe S3 : Systèmes, Logiciels, Services

Designing and Prototyping an Event-based Communication System on Mobile Ad Hoc Networks - Conception et Prototypage d'un Système d'Evénements pour Réseaux Mobiles Ad Hoc

Amir R. Khakpour, Isabelle Demeure

Technical Report
July 2008

Institut TELECOM - TELECOM ParisTech - Départment Informatique et Réseaux
46, rue Barrault, 75634 Paris Cedex 13, France.

Abstract

In this report we present "Chapar"[1], a publish-subscribe event system for mobile ad hoc networks. Chapar is designed to support mobility and to be robust to disconnections and networks partitions that often occur in mobile ad hoc networks. Chapar is part of the Transhumance middleware project [31, 32] and is designed as a cross-layer overlay architecture.

Dans ce rapport, nous présentons "Chapar"[2], un système d'événements de type publication-souscription pour réseaux mobiles ad hoc. Chapar est conçu pour opérer dans un environnement mobile, pour résister aux déconnexions et aux partitions réseau fréquentes dans les réseaux mobiles ad hoc. Chapar fait partie de l'intergiciel conçu dans le cadre du projet Transhumance [31, 32]. Il est développé comme une couche de recouvrement qui traverse plusieurs couches (cross-layer).

[1]Chapar is the first message delivery service known in the history. It was used in the Persian Empire in hundred years B.C. [Reference: Allyn Huntzinger,"Persians in the Bible", Global Commission Inc, 2004.]

[2]Chapar est le
premier service de messagerie connu dans l'histoire. Il était utilisé dans l'empire perse quelques centaines d'années avant J.C. [Reference: Allyn Huntzinger,"Persians in the Bible", Global Commission Inc, 2004.]

Contents

Introduction

In recent years the diversity of the applications developed for the mobile nodes is noticeably increasing. These applications mainly employed in many domains such as military, police and fire fighting, gaming, and voting, should be adaptable to the harsh and dynamic mobile environment in which nodes experience undesirable frequent disconnections and suffer from their consequences. Besides, the exclusive characteristics of mobile ad hoc networks and more specifically pervasive and ubiquitous networks including the nodes homogeneity, lack of fixed infrastructure, nodes anonymity, and unreliable wireless shared media induces considerable complications for applications using traditional communication paradigms such as synchronous "client-server" model. Hence, the middleware which supports asynchronous communication paradigms such as publish/subscribe systems and provides a favorable degree of reliability and consistency receives a great deal of attention from a wide range of applications developed for distributed systems.

The publish/subscribe system supporting store and forward mechanism to propagate messages (called events in this paradigm) provides desirable decouplings namely space, time and synchronization decouplings to address the above issues [1]. These decouplings reduce the nodes interdependencies and provide them with many features beneficial in exclusive ubiquitous environments. However, the classical centralized pub/sub system does not suit the dynamicity we encounter in such networks. In general, employment of the centralized approaches in infrastructureless networks is not recommanded. Alternatively, in the destributed algorithms a specific task is assigned to a set of nodes collaborating to provide a reliable service. So in our case, a destributed event system is required to handle the desired asynchronous communication. This distributed event system should be light enough (in terms of communations and computations) to be implemented on mobile nodes which are generally considered as low-power and tiny nodes with limited resources.

Another big challenge that event systems are required to cope with, is node mobility. Nodes' random mobility does not allow the event system to keep track of publishers and subscribers properly. For event notification, the event system is compeled to relocate the subscribers for event forwarding and since the different multicast techniques are usually utilized for event dessimination, for each published event the event system needs to recalculate its multicast tree. Moreover, the mobility induces the transient network partitioning in sparse networks. In network partioning at least two subset of nodes disjoint and then merge over time. This phenomenon causes unreliability in many services uses sysnchronous communication and utilize flooding for message propagation. Therefore, the proposed event system should provide solutions to tackle these problems. And indeed, the performance analysis of the suggested algorithm to handle network partitioning, as well as node failure, high speed node mobility along with the their overload on the resource-restricted network should be investigated.

In this report, we propose a novel event system "Chapar", with distributed architecture fitting the mobile environment charactristics. Chapar is an overlay network utilizing the node routing table (cross-layer approach) to disseminate the events. It is overlay network, because each event has its own destination field by which it is forwarded to the target subscribers through event system. For the routing perposes it uses the underlying routing layer (OLSR [3]) to forward the event to the appropriate next hop to be subsequently delivered to the destination subscriber(s). This event system supports publish/subscribe system, point-to-point and pount-to-multipoint communications. This event system is designed and implemented as a Event Management module in the communication layer of the Transhumance Middleware Project [31, 32].

This report is organized as follows. The Chapter 1 contains a broad overview on publish/subscribe systems, the concept, different implementations and its potential applications in mobile environments. Chapter 2 encompasses the Transhumance middleware project overview and the Event Management module specifications which should be satisfied by the proposed method. We then present the Chapar method description in Chapter 3. The complete performance analysis of this method is also provided in this chapter.

Chapter 1

Publish/Subscribe System

1.1 Introduction

The computer and communication network technology evolution push many entities to develop applications on distributed systems in many domains including financial, governmental, industrial, scientific and etc. Such a vast spectrum of requests from different industries together with their interest to interconnect their large-scale applications require a flexible communication paradigm which is more dynamic and respects the decoupled nature of applications. The tightly-coupled traditional request/reply communication model in "server-client" architecture is not promising enough for new generation of applications on distributed systems. In fact, Point-to-point and synchronized type of communication only supports static type of applications and restricts the developers to develop software with larger variety of services with scalable features.

In this work we focus on the *publish/subscribe* system [1], a loosely-coupled paradigm for communication between entities. This interaction scheme focuses on fully decoupling the entities in different scopes to provide more scalable and dynamic environment for applications to interact. In such an architecture, instead of client and server, two new roles are defined:*Subscriber* (or *Consumer*) on one side and *Publisher* (or *Producer*) on the other side. Subscribers are nodes who express their interest in an *event* and or group of events with specific pattern and submit it to event service (or *Broker* node). On the other hand, the publishers are nodes who generate the events and dispatch them to the broker node. Then, the broker node notifies the subscribers of the published events in an

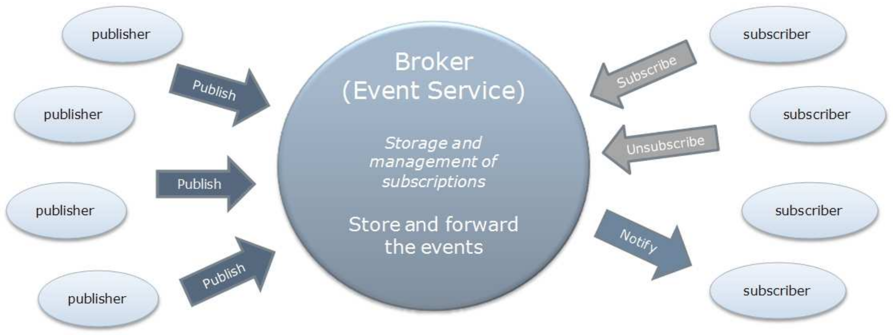


Figure 1.1: The Publish/Subscribe system components and basic diagram

asynchronous communication model (Figure 1.1).

In the first part of the remainder of this chapter we continue the introduction of publish/subscribe systems and point out the principle features of publish/subscribe systems, moreover some publish/subscribe system which are already developed will be surveyed. In the second part we concentrate on adaptation of publish/subscribes system to mobile environments and mobile ad hoc networks.

1.1.1 Publish/Subscribe System: Different Decouplings

As it was described, the publish/subscribe systems introduce a paradigm by which subscribers register their interest to group of events generated by publishers. In this scenario, the event system manages the subscription messages, processes the published events and forwards them to their corresponding subscribers. This type of communication paradigm regardless of how the event system works, supports space, time ad synchronization decoupling (Figure 1.2) between the event publishers and event subscribers:

- **Space decoupling**: the main two parties (subscriber(s) and publisher(s)) work quite independently, and they do not need to know each other and are not aware of each other's network address. This makes the communication completely "anonymous" with respect to where event comes from and where it goes to. Besides, this feature provides an indirect communication through event system and appoints the event system as a broker entity for event delivery between publisher and subscriber. Whereas, in the traditional client-server model, we have direct communication between two parties and the delivered messages has a specific address which indicates the message sender and receiver.

- **Time decoupling**: both entities are not required to be present at communication time. This means that the publisher may publish when the event subscriber is

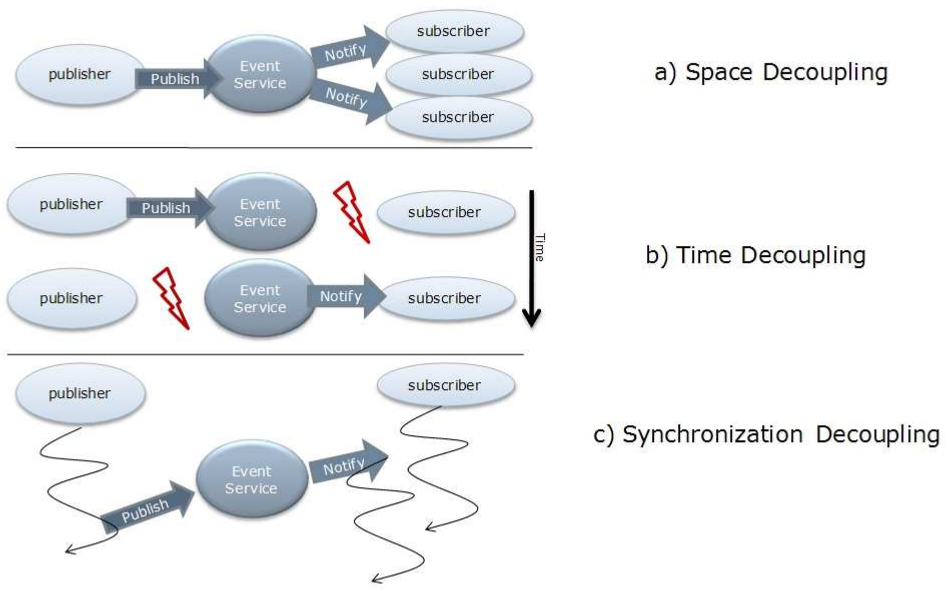


Figure 1.2: Different decouplings in publish/subscribe system

disconnected, or on the other side, the subscriber is able to subscribe to and be notified of an event whose publisher is detached from the network at that moment. This decoupling requires the event system to have a memory and queuing system to store the published events and forward them not only to those who are subscribed and disconnected, but also to newcomer nodes that subscribe to an event which was published before.

- **Synchronization decoupling**: The communication is completely asynchronous. In other words, the subscribers do not need to block and wait to be notified, and similarly the publishers need not block the process while producing an event. Therefore, neither subscriber and nor publisher are required to wait to be acknowledged by their recipient before moving on. Since the production and consumption of the events do not happen at the same time, the publish/subscribe exchange is considered as an asynchronous communication paradigm.

These three dimensions of decoupling provide many capabilities in order to develop applications for large-scale networks in which guaranteeing quality of service with respect to node availability monitoring, network connectivity supervision is not feasible. Moreover, publishers and subscribers can be added or eliminated at run-time, which allows the system to grow or shrink in complexity over time. In addition, the publish/subscribe system is helpful to increase the reliability on loosely-connected networks like wireless networks, since a node would not miss any packets while it was temporarily “disconnected” from the network. It also helps the unification of the networks with similar capabilities to a bigger network with the same functionality along with some interaction features. Accordingly, it widens the diversity and raises the number of clients using the network, since they just need to subscribe for what they are interested in and wait to be notified.

For example in real world Business-to-Business (B2B) applications wholesalers need to distribute and announce their products with their corresponding prices and volume to retailers. In pub/sub model, the publishers are the wholesalers which publish the information to the subscribers (retailers) who are subscribed to this kind of information by some product orders. Different techniques in subscription that will be explained in the following subsections enable the retailers to define subscriptions accurately which empowers them to take advantage of hot deals and best prices for their business. Other B2B applications like stock broker, broadcasting the stock prices to investors or manufactures and providing bid request for multiple suppliers follows more or less a same model.

There are also many asynchronous software interested in pub/sub systems because of the mentioned decouplings. Example of which, we can name software and antivirus update daemons, consumer alerts (like security alerts and etc.), multiplayer online games, and etc.

Publish/subscribe systems help security of the two parties as well, since the broker network acts as a proxy between both entities and forwards the events indirectly. This leads to nodes interaction without the necessity of having knowledge about each other (space decoupling). Besides, the delay of publishing and notifying may shelter some services suffering from real-time attacks (synchronization decoupling).

1.1.2 Publish/Subscribe System: Types

There are different ways and techniques utilized by subscriber to express their interest in an event pattern. These techniques are categorized into three basic types of pub/sub systems, namely *topic-based*, *content-based* and *type-based* pub/sub systems [1]. In pub-/sub systems, this interest expression is declared in the form of custom *filters* generated by subscribers over different event attributes. Therefore, different types of pub/sub system use to different filter types and apply these filters on incoming event, using different techniques.

Topic-Based Pub/Sub System: the first and most fundamental pub/sub system is *topic-*, *subject-*, or *channel-based* pub/sub, in which the subscription is based on the classification of the attributes of events. In other words, the subscribers are able to express their interest in some individual topics through some *keywords*. The topic based pub/sub system was formerly called *group communication* [1, 4, 5]. In group communication the nodes assigned as members of a group will receive the messages in the group. However, the group communication uses flat addressing, whereas the topic-based pub/sub system may organize the topics into hierarchy and take advantage of the hierarchical addressing scheme and provide containment subscription over topics [1]. This containment subscription may support the wildcards, multiple topic and etc. Figure 1.3 shows a topic based pub/sub system as well as the full decoupling of publisher and subscriber.

Content-Based Pub/Sub System: despite of the fact that topic-based pub/sub systems provide hierarchical addressing and facilities including wildcards and multiple attribute filtering, it is static and has confined expressiveness, which is not really desirable in many applications. *Content-* or *Property-based* pub/sub system allows the event system to discriminate between incoming events by applying powerful and flexible filters on the content of the event. So in this case, we have no classification of events based on their attributes, yet they are distinguished based on the event characteristics. For instance, the subscriber is able to define its interest by operators ($>, <, \geq, \leq, =, \neq$) and, based on the type of the attribute, it could use ($\in$, $\subset$, and etc). Indeed these operators boost the expressiveness of the filters for subscription.

Type-Based Pub/Sub System: in Type-based system [2] the event classification is based on the structure of the event rather than its content. If we consider an event as an object, the filter is basically defined based on different object that would

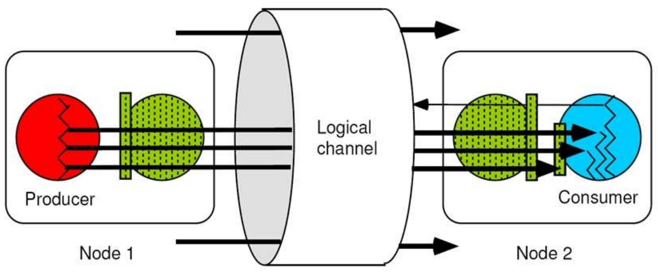


Figure 1.3: The channel-based publish/subscribe schematic

exist in the event system. For example, in the last example of wholesaler/retailer, the wholesaler may have different type of messages to generate such as quotes, the deal information, announcements, discovery and etc, which would be dispatched in different structures and may be filtered based on their types.

To have a brief overview on the pros and cons of different pub/sub variations, the topic based system is faster, and simpler with low run-time overhead. Using efficient hierarchical addressing as well as smart design of the event in the sense of defining the customized attributes is recommended by high speed and large-scale applications to avoid sophistications. On the other hand, the content-based system provides more features and accuracy in filters interpretation and expressiveness; however, it is costly in terms of complexity and CPU usage. The mentioned complexity mainly includes the language parsing and how the subscription should be represented to the event system. The subscription grammar is also important for filter enforcement on the event. For instance, SQL[6, 11, 7], XPATH [8] and others [10, 13, 14, 15] may be proper candidates for this publish/subscribe system.

1.1.3 Event Types and Event System Architectures

As mentioned before, events are composite messages including a set of attributes. Each attribute may be utilized by the event system to distinguish between events. These attributes are also helpful for subscribers to define their interest and subscribe to the event system. The events which are published to the event system are *published-events*; those who are kept by the event system are called *stored-events*; and those which are verified by the filters, assigned to subscribers and dispatched through a multicast connection (or unicast connection) are called *notified-events* (figure 1.4).

The events are generally categorized into two species: *Messages* and *Invocations.*

- The *messages* or *data events* is composed of data encapsulated in a data structure with some attributes which describe the event. In other words, these events contain an string message as the payload and some header attributes to describe the message.

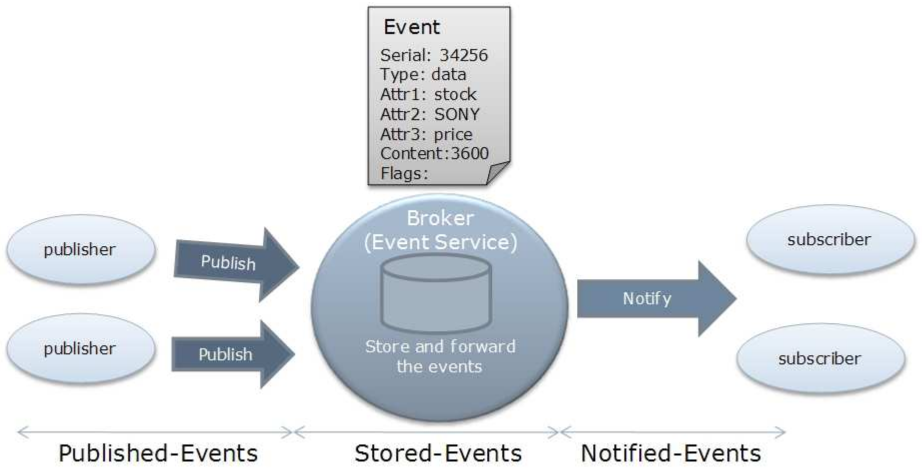


Figure 1.4: Different event types based on their processing level in pub/sub system

- *Invocation events* or in general *invocations* are a type of object which has specific meaning for the event system and the subscriber. While the subscriber receives this kind of event, the event handler on the subscriber will execute a predefined set of instructions based on the event header and some arguments which might be included as payload and sent from invoker to invokee(s).

As shown in figure 1.1 the publish/subscribe system consist of three main components, publishers, subscribers, and Event Service. To implement such system, one approach is to designate a broker node to act as a proxy between publishers and subscribers. The broker node receives the events from the publishers, stores and forwards them to the subscribers. In this so-called *centralized* architecture [11, 7, 6, 16, 17], the broker node supervises and handles the successful event delivery to the subscriber(s) based on its local database of subscriptions. The communication between publishers and the broker would be a point to point connection, whereas the a multicast connection is recommended between the broker and the subscribers. Broker node is able to guarantee the event delivery and data consistency as well as providing fairly good quality of service. The store and forward mechanism also provides decouplings expected from publish/subscribe systems. However, the centralized architecture suffers from a single bottleneck and single point of failure, in which if the broker or its subscription database fails, even for a short time, the whole system will be paralyzed in addition to losing all of subscription information and stored events which would be needed for absent nodes. This defect could be really perilous for critical applications such as banking applications, e-commerce software and etc. Finally, we should not overlook the fact that the centralized architecture might not support properly the scalability and high throughput.

An alternative approach is *distributed* and *decentralized* architecture in which we have no centralized broker entity; instead the publisher stores and forwards the events in its local queue to support asynchrony. In this architecture the subscriptions are broadcasted to the publishers network, and events are multicasted to subscribers. Such an architecture is interesting for applications seeking for fast and efficient event delivery like multimedia broadcasting. This approach is not really scalable since the subscription messages should be broadcasted to the publishers network which could be extremely large. It prevents us to have space decoupling since the publisher and subscriber communicates directly without relying on the third party. Moreover, the time decoupling could not be supported completely with this architecture due to the fact that publisher may fail and its stored-events would be lost so the new subscribers could not access the preceding published-messages. Besides, the publisher set should be fixed over time. Otherwise, for each new publisher all of the subscribers ought to execute all of their subscriptions which is not feasible in actual situations. And ultimately, Management and monitoring of such system is difficult especially if the network administrators want to enforce some policies (e.g. security policies) to the network and publish/subscribe system.

Another approach that encompasses the advantages of the first approach and avoids the disadvantages of the decentralized approach is to implement the event notification service on *distributed servers*. This method is embraced by many publish/subscribe applications [12, 15, 18], since we may have the reliability which is one of the basic requirements of pub/sub system and at the same time we evade from leaving the network with a single bottleneck and point of failure. These methods are also scalable, although they add some intricacy and complexity in the broker system. The type of communication in this approach varies according to the formation and position of the broker nodes which constitute the broker network. Thus, efficient data dissemination between different nodes and blocks is still an issue and open topic for research.

1.1.4 Some Popular Publish/Susbscribe Systems

Java Messaging System (JMS): Java Messaging System [11] is an API specification for J2EE package supporting the asynchronous communications, using a centralized architecture. JMS is a standard interface for Message-Oriented Middleware (MOM) supporting two main messaging models *point-to-point* and *publish/subscribe* in both synchronous mode and asynchronous mode where the communication parties are loosely coupled. It also offers reliable delivery which means it guarantees that a message is delivered and delivered once. To support reliability, JMS provides a feature on subscription which is called *durable* subscriber. Durable subscription enables application to receive their corresponding published message when they were inactive or unreachable at the publishing time (supports time decoupling).

In JMS the subscriber needs to subscribe to a "Topic" then either the subscriber blocks to receive the event (or may block for limited time) (synchronous mode), or registers to a *message listener* (asynchronous mode). When the message listener receives the event, it delivers the message along with calling a callback method "onMessage()" to handle the notified message.

Indeed, the publish/subscribe system in JMS follows the group messaging model in which Topic can be assigned as a central queue shared between all of the nodes. A common application for such system is when a group of applications want to notify each other about a particular occurrence, they can do it through a topic which every member application are listening to this queue asynchronously. The subscription in this architecture is registering to a group as a member to receive the group messages. JMS programming model is shown in figure 1.5.

CORBA Notification Service (COS Notification): The CORBA Notification Service (NS) [10] which is the extension of earlier Event Service (ES) [9] is the Object

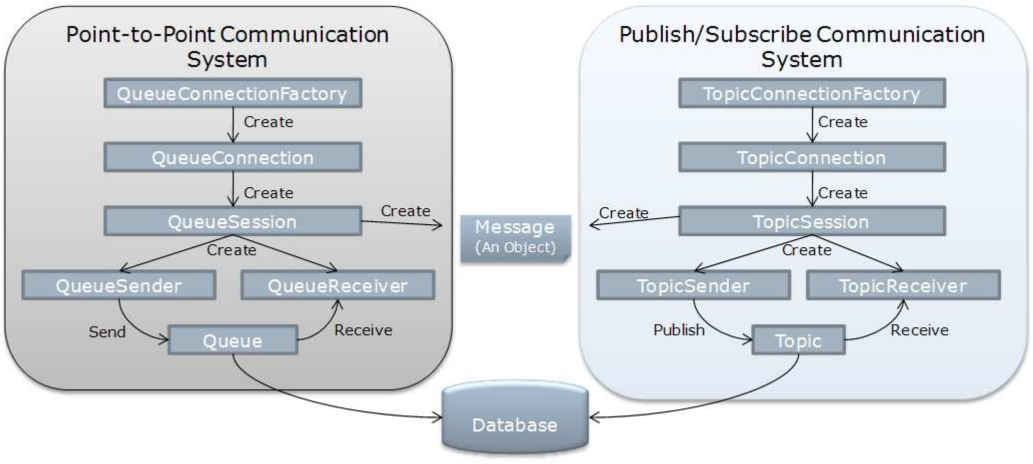


Figure 1.5: JMS programming model for point-to-point and pub/sub sytem

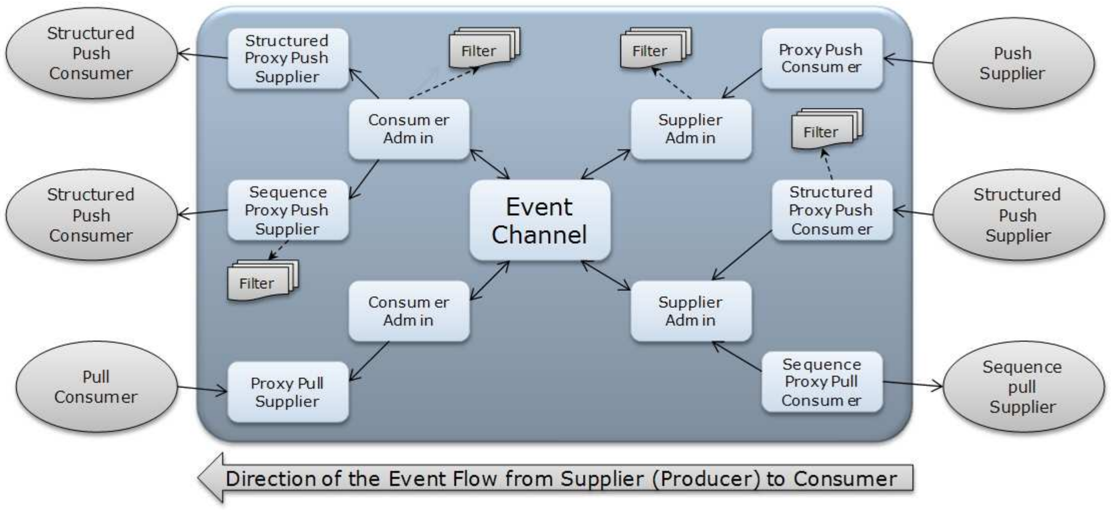


Figure 1.6: The main interface in CORBA NS

Management Group (OMG) specification for publish/subscribe messaging system. It provides an asynchronous event-driven system with various QoS properties.

The NS first creates an event channel. The event object channel is initiated with primitive QoS and administrative properties. Subsequently the created channel has default SupplierAdmin and ConsumerAdmin interfaces (inherits same QoS and administrative properties). When a supplier wants to publish, it needs to obtain a proxy consumer by contacting a SupplierAdmin and connect to it. Similarly, a consumer needs to obtain a proxy supplier by contacting a ConsumerAdmin and connect to it in order to receive the events. The proxies make parties completely decoupled. In this system admins and proxy objects has some filters by which they decide whether the incoming events should be forwarded or dropped (figure 1.6).

CORBA Notification Service support two fundamental event communications namely *push* and *pull* [20]. The push communication mode is typical publishing in pub/sub system. In push the producer push the event to the event channel and it is again pushed to the consumer. Whereas, in pull communication mode the event channel asks for events from producers by invoking a pull() operation on them. Subsequently, the event again is pulled by the consumer from the proxy object it is connected to.

Similar to JMS, CORBA NS is just an API specification. There are some implementations of NS like OpenORB [21], and OmniNotify[22], neither of which are supporting distributed systems. Figure 1.6 present a block diagram of CORBA NS main interface in both pull and push modes.

1.2 Publish/Subscribe Systems for MANETs

The publish/subscribe model is receiving great attention from the mobile ad hoc network community. As a matter of fact, mobile computing faces many difficulties due to loosely-connected wireless network with low power mobile nodes. Such systems embrace the reliable asynchronous communication facilities promised by pub/sub system. In fact, node mobility induces frequent disconnections and unreliable communications between nodes that may interrupt the "client-server" communication frequently used in distributed systems. In such situations, the publish/subscribe communication paradigm sounds promising considering its favorable decoupling features, node anonymity and asynchrony that helps the reliable data dissemination, network adaptability as well as scalability in this quite dynamic environment.

On the other hand, mobile networks are emerging and extending fast in recent years due to proliferation of wireless devices with low prices and quite good battery life time. Subsequently, the range of applications supporting mobile networks (preferably ad hoc networks, since real mobility requires no fixed network infrastructure) and multi-hop message forwarding is considerably broadened while traditional communication systems may not fulfill developers' expectations. Therefore, adapting the publish/subscribe system to unprecedented characteristics of mobile ad hoc network is one of the hot research topics lately. There are several papers and research studies [17, 23, 24, 12, 25] on adaptation of the publish/subscribe system to low power mobile nodes in a dynamic networks that tries to take advantage of the decouplings offered by this system to boost the communication quality on the network.

1.2.1 Pub/Sub Systems and Challenges in Mobile Environment

In order to start studying the publish/subscribe system on mobile environment, we first give a brief review on the basic pub/sub component in classical framework where we have event publisher creating the event, event broker forwarding the event to so-called event subscribers who receive the generated event. In applications mentioned in fixed and wired networks, the publishers and subscribers are mostly located in two different networks, and may communicate through the broker, whereas in mobile network and its corresponding applications the publishers and subscribers are not separated and they are all located in the same network. Moreover, a publisher could be a subscriber and vice versa at the same time. Thus, remembering the fact that different applications are using pub/sub system because the features it caters for dynamic environments, we are facing quite new assumptions which might need some adaptation and adjustment of the classical pub/sub framework.

Appropriate publish/subscribe architecture is another debatable issue in mobile environments. In the centralized architecture, a single node receives the subscriptions and forward the incoming events to their corresponding subscriber according to its local subscription database. Using this architecture enables a central management and working with properly organized system where indeed a powerful node (a supernode) should be considered as the broker. However, the main drawback of centralized system is revealing more than before. Having a single point of failure in homogenous mobile ad hoc networks in which nodes are prone to failure or temporary disconnection make the whole event system unreliable. The broker node failure induces losing the subscription table, losing stored events and finally losing the whole event system. Furthermore, single broker node requires the considerable bandwidth to handle all incoming events and dispatching them to the subscribers which is not really feasible in low bandwidth mobile nodes. Another factor that should be taken into account is the broker node conditions. Since all of network event transaction is done through the broker node and the fact that the multi-hop routing is used in mobile ad hoc network, the position of the broker node, the number of broker node links, and indeed the broker node power reservations are playing the important roles in performance of the system.

In spite of the mentioned problem, there are centralized approaches considered for mobile networks. For instance, Segall et al. in Elvin [14] proposed a centralized architecture with *quenching* technique. In quenching technique, the publisher verifies if there is a subscriber for this event prior to dispatching its created event to the broker. This verification is performed using the $c_{all} = c_1 \vee c_2 \vee \cdots c_N$ (where c_i is denoted the subscriptions). Although Elvin is not really addressing our main concerns, it significantly reduces the generated network traffic and broker node processing load. Yet it imposes some processing cost to the publisher side which might be a problem for some tiny sensors with low capabilities.

Huang et al. [17] studied using replication for centralized broker network. They believe the replicas will increase the availability and reliability of the event system when it faces the broker failure or network partitioning. However they introduce new concerns including *Orderness*, *Consistency*, and *Completeness,replication*. As we expect in classical centralized event system, the events need to be received in order they are sent which would be jeopardized using multiple replicas (Orderness). Consistency is mostly addressed to differences between set of notified events by replicated system and non-replicated system. Example of which could be the duplicated notified events which occurs because of the lack of harmony between replicas. The event duplication may cause substantial network traffic overload which is quite unnecessary and may cause some message confusion for certain applications running on consumer side. The completeness introduce stricter criterion than consistency. In completeness we required to ensure that the replicated system forwards all of the events to their subscribers. In an event system it could be referred to similarity of the replicas' different databases. Assume a subscriber subscribes to pattern of the events and the EBR1 is the only replica which receives the subscription. If in publishing time the EBR1 is not available anymore, the message will be received by other EBRs and they will discard it since there is no subscription for that so eventually the event will not be forwarded. The same story could be considered for stored events. Conclusively, the system completeness would suffer from mobility and temporary disconnections as well as network partitioning during the network operation.

The decentralized architecture may be considered for ad hoc networks. In fact in this framework, there is no broker in the event system and actually the publishers hold their own subscription table. Thus, the subscription messages are flooding the network to let all potential publishers to receive them. Although in this architecture the event computing is shared fairly among the producers, they (especially if they are sensors) are required to hold the subscription database and perform the event system computation in order to send the event to its corresponding subscribers. The nodes' tendency to failure or temporary failure results in producers to lose the subscription database as well as the data storages they have retained for persistent messages. They may also miss some subscription messages due to network partitioning or transient network detachments. Besides, the new publishers which join the network would not have the subscription and stored events database. And finally, using flooding in subscription phase bring about some scalability concerns over the network.

The third architecture, using distributed servers, seems more adaptable to mobile environments. In this architecture we have not a single centralized broker; however a group of nodes share this responsibility as the network of brokers. The formation of the broker network, the way the broker nodes communicates, the definition of broker nodes and finally the way they handle subscription and forward the matched events to subscribers are almost different between different available techniques and implementations. Therefore, all of these techniques in addition to challenges facing in the mobile network have their own argues. The main contribution of all proposed techniques using the distributed broker nodes is how to deal with mobility in general which includes the mobile clients, frequent mobile disconnection, "hand over" (while a mobile node leave its corresponding broker and attach to another one as a result of mobility), and multicasting the events with respects to their correct position [17, 23].

Finally, we may draw the conclusion that in employing such an architecture to MANETs the challenges are quite different. As it was described, there is a group of nodes interconnected, called broker network, and they need to route the published events to subscriber in a multi hop-distributed network. This network formation is constantly changing due to mobility and node attachment, detachment, failure or appearance. Some nodes are also flapping on and off to save their battery. Since the broker network is distributed all of these challenges and occurrences should be taken into account. The distributed event system generally tries to build a tree-based broker network (loop-less and scalable) to forward events between broker nodes and finally notify it to the subscriber(s). This communication over this network is usually a multicast communication since a broker tree is already provided. In the following we review some of the implementations currently available with MANET support.

1.2.2 Some Popular Pub/sub Systems Designed for MANETs

JEDI: proposed by Cugola et al. [12] stands for "Java Event-based Distributed Infrastructure" and is a distributed event system which supports mobility. JEDI uses the hierarchical strategy to pass along the events from publisher to subscribers. In JEDI the distributed architecture consists of a group of *Dispatching Servers (DS)* connected in a rooted tree structure. This tree is virtually implemented on set of nodes in tree

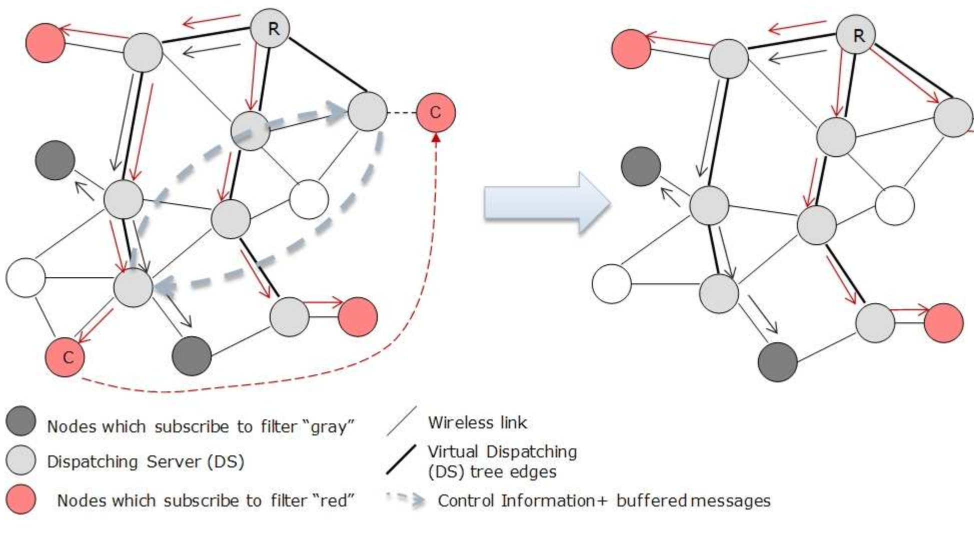

R
C
C
R
C
Nodes which subscribe to filter "gray"
Dispatching Server (DS)
Nodes which subscribe to filter "red"
Wireless link
Virtual Dispatching (DS) tree edges
Control Information+ buffered messages

Figure 1.7: JEDI mobility support schematic, before, during and after node displacement

structure which means, except the “root” node, every DS has only one parent DS. Event subscription and un-subscription requests are propagated by each DS upwards towards the root. Event notifications are processed similarly and forwarded by the local DS to its parent (upstream propagation). Upon receiving an event, each DS checks its descendants whether the event is matched against the subscription table or not , if it is the event is forwarded down the tree (downstream propagation). This strategy requires that a given DS knows the event requests of its descendants in order to make the forwarding decision. Moreover, since all requests and notifications are propagated up to the tree, the communication and processing overhead of the nodes near the root may become a bottleneck. If root be detached from the network by any reason, parts of the tree would be secluded. In this case the system needs to deal with segmentation and to be able to mend the tree or elect a new root and indeed a new tree.

JEDI supports mobility with *moveOut* and *moveIn* operations. The DSs in the JEDI architecture support mobility by allowing clients to “disconnect”, and move to a new DS, and “connect”. The DSs manage temporary storage for notifications. Afterwards, the new DS contacts the old one directly in order to receive the accumulated notifications. The old DS notifies its parent DS to route any further notifications for this client to the new DS. Figure 1.7 shows how node *C* is moving from one DS to another DS, and how JEDI manage to update the tree about this sudden change and forwarding the buffered notifications to its current DS while it was unavailable (hand over operation).

Mobility support in JEDI is still under consideration, for example the latency of updating the dispatching trees when clients are moving very frequently and in the case of abrupt disconnections is still open issues. Another issue could be what kind of abstractions are needed at lower levels in order to detect disconnections at upper level. Moreover, JEDI suffers scalability drawbacks; JEDI uses some global broadcasting (for instance, in time of tree creation when the group leader elected, it needs to announce itself to all of DSs available in the network).

SIENA proposed by Carzaniga et al. [18] is a distributed content-based pub/sub system consisting of a network of event brokers. Siena works based on Filter-based routing algorithms. In this routing mechanism, the event is verified by each node and is forwarded to the subscribers based on its content and the subscription filters. So each broker node needs to match the event against its filter list which imposes a considerable load on each node on busy networks. The multicast trees also in such cases are fixed and are not adaptable to mobile environment. It also deals with the trade-off between scalability and expressiveness. Hermes[30] uses the same distributed filtering algorithm to propagate events on the reverse paths of previous subscriptions.

However, Siena relies on a global broadcast operation to disseminate advertisements through the entire network, which limits its scalability. It does not support the notion of an event type, and does not provide any other middleware services. The network of brokers is not able to cope with failures because of the static logical topology.

Another example of such filter-based routing pub/sub system is Gryphon[29] which is developed by IBM. Gryphon is a content-based pub/sub system using hierarchical multicast tree from publishers to subscribers to forward the events. However, it does suffer from more or less same problems all of filter-based routing algorithms have with mobility and scalability.

STEAM (Scalable Timed Events and Mobility) proposed by Meier et al. [27] event system is specifically designed for wireless ad-hoc networks. In fact, STEAM uses three different filters: *Proximity Filter*, *Subject Filter* (implemented on producer side) and *Content Filter* (implemented on consumer side) to address the problems related to dynamic reconfiguration of the network topology. STEAM does not utilize any broker system and its applications are limited to proximity communications, like traffic management.

Events are identified by a name and a set of typed parameters. A subject filter is matched against the event and mapped onto a proximity group. A proximity filter corresponds to the geographical aspect of the proximity group. A proximity filter specifies the scope in which events are disseminated. A proximity filter applies to an event type and is established when the type is deployed. As matter of fact STEAM exploits a group communication service for notifying interested entities. Groups are geographically bound and nodes are identified using beacons.

STEAM does not really follow the classical architecture we defined for pub/sub system, since after an event is created to be published, it firstly needs to match against the subject and proximity, and then when it is received it should matches again against the content in the subscriber side. So in fact decoupling in this kind of system is not really tight. Because the producer who owns the proximity filter needs to know the location information from the subscriber.

STEAM uses a Proximity-based Group Communication Service (PGCS). In this service, groups are assigned certain geographical areas. A node that wants to join a group needs to be located in the group's area. STEAM provides a Proximity Discovery Service (PDS) that uses beacons to discover proximities. Once the proximity is discovered the associated events are delivered to the client if it has a matching subscription.

Similar work is done by Cilia et al. [28] which is a location and time based which improves the bootstrap process of the node. Because the location-dependent information will be changed due to mobility, it needs the new bootstrapping process to collect the location-dependent information.

Chapter 2

The Transhumance Project

2.1 Introduction

The Transhumance Project [31, 32] is a middleware for medium-scale mobile ad hoc networks and is partially funded by French Government ANR-RNRT. It targets slow mobility (pedestrian speed) with network size up to 20 nodes. The main contribution of this project is to have a power-aware and secure middleware supporting a wide range of applications such as chat, voting, games and etc.

As it is shown in figure 2.1, Transhumance is composed of three basic layers which are all in interaction with the security and the energy blocks for their operation adaptations. These three layers are:

- the *Services Layer* contains the basic services supported by middleware and could be utilized by developers to build their applications upon them. An example of which is "Data Sharing". Data sharing is almost utilized in all of the applications working in distributed fashion and need to communicate to each other by large data structures like files. Other services like File Transfer, Middleware Service Management and etc. are main components of services layer.

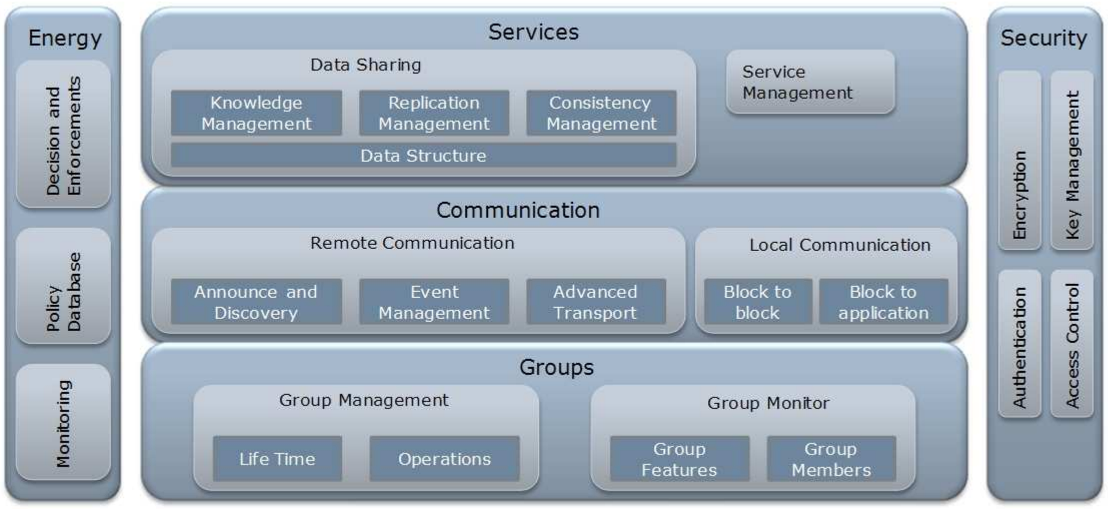


Figure 2.1: Transhumance middleware architecture

- The *Communication Layer* includes several blocks which provide an optimized communication and information transaction infrastructure not only among nodes but also between different local middleware blocks. An Advanced UDP transport layer, Event System which support publish/subscribe system, and Announce/Discovery block are examples of this layer's elements. In this layer power adaptability is one of our main concerns since the major part of node power consumption is because of its communications to rest of nodes. So it is tried to be designed as efficient as possible, more specifically with less overhead and taking advantage of some novel approaches to reduce the number of data transmission, to preserve energy effectively.

- The third layer, *Groups Layer*, basically shows the nodes management strategy utilized by middleware. In Transhumance the nodes management is done through defining groups and communities including nodes with similar interests. Therefore, in this paradigm nodes use group communication methods for message and data exchange. This layer provides many group management features to facilitate the group administration and communication.

Security and Power (energy) modules are other fundamental ingredients of Transhumance which interact with all of the layers and most of the blocks to supply them some capabilities to make whole middleware secure and power-aware. The security module contains access control features to distinguish users and clients based on their identity. It also provides each group with a symmetric key for ciphering the messages handed over through the network within a virtual group. On the other hand, the power adaptability module is holding a policy database in which each block actions is predefined with respect to the level of available node power. Therefore, middleware blocks needs to retrieve their operation parameters from this module before executing any operation.

In the remainder of this chapter we present the specification for event system and review the requirements and expectations of event system as a block categorized in communication layer.

2.2 Event System Specification

2.2.1 The basic Requirements and Definitions

The Event system is one of the basic blocks located in Transhumance middleware. It is responsible for generating, managing, propagating and notifying the events in the mobile ad hoc network. In fact, the event system in Transhumance is designed to be a communication layer through which services are able to exchange events in the network. Events in this system are composite data units composing of several attributes which represent the message in different dimensions.

The event system may work in different communication modes with different functionalities. These modes are:

- Point-to-Point communication: The event system shall be able to provide communication to a specific destination node declared as an event attribute.

- Point-to-Multipoint communication: The event system shall be able to send an event to a list of network addresses via a multicast system with minimum resource overhead.

- Publish/Subscribe system: The event system shall be able to provide publish/subscribe system introduced in Chapter 1. In such systems, events have no particular destination point and would be notified to those nodes who subscribe their interest in such events. The notification is required to be done through a resource-aware multicast system.

The different event attributes are:

- Type: Declares the event type with respect to the middleware blocks, the typical event types are: data, announcement, discovery, migration, a subscription/unsubscription event, and others.

- Identifier: A unique identification for an event in the network.

- Group Identifier: Utilized in group communications which means that the nodes in a common group can subscribe to events with specific group identifier (GroupID) to receive the messages dispatched within a group. These messages might be encrypted by the group key provided by "security" blocks.

- Subject: The subject of the event content (used to distinguish between different messages with the same type)

- Content: The event content contains the event payload.

- Sender ID: It includes the network address of the event publisher or producer.

- Destination: Depending on different communication modes stated before, destination field would be left empty (pub/sub mode), or filled by one IP Address or a list of IP Addresses (Multicast and unicast mode).

- Lifetime: The events are retained in the network until their lifetime is over. So in publish/subscribe mode, the subscribers are able to subscribe to an event after it is published and be notified in the event lifetime duration. Moreover, for other communication modes, if the destinations are not reachable at the publishing moment, the network would hold the event for lifetime period and deliver the event in case the node(s) returns or be accessible again.

- Persistency: Persistency is a flag in event data structures which denotes that the event should be kept in the network with no lifetime consideration. And it is expected that all of the subscribers including those who are attached to the network at the publishing moment and future subscribers which will attach then to be notified. Since the persistent events are important, the event system needs to ensure if they arrive to destination node(s) properly.

2.2.2 Event System functionalities in detail

The main functionalities and procedures considered for event based system are:

- Even Creation/Destruction: The event system shall be able to create and destroy an event. Additionaly, it needs to provide methods for upper-layers to fill the different attributes, and perform some authorization procedures for event creation (explained in section 2.2.3), and calculating a unique event identifier.
- Subscription/Unsubscribing: As a part of publish/subscribe system, nodes are able to show their interest in a specific kind of event by specifying some designated event attributes such as the *event type, sender ID, subject, content,* and *group ID.* On the other hand, they are also able to unsubscribe to the events they have had subscribed to.
- Event Delivery: The event system is expected to implement the different communication mode considering resource constraints in wireless mobile networks. We have three kind events delivery system which needs to be supported:
  - Real-time Events: These events are delivered to their destination node(s) if they are connected to the network and are available at the time the event is published. In this type of events, the event will be destructed after it is dispatched. Thus, the unreachable nodes will not receive the event.
  - Events with Lifetime: As it is explained in previous section, these events are kept in the network (by redundant nodes) and will be delivered to the nodes in their lifetime duration. Therefore, destination nodes which do not show up during the specified time period will not receive the message.
  - Persistent Events: The event with persistency flag will be kept in the network will not be deleted. Additionally, this group of the events must be delivered with acknowledgement to make sure that the destination nodes receive the event successfully. However, since there are some nodes that might never come back to the network, the persistent message repository is a limited queue, so when it exceeds the capacity and new event comes, the oldest event will be discarded (a FIFO queue).
- Event Notification: Last phase of event delivery is notification. The nodes needs to be able to receive the events from network.

2.2.3 Event System Security

The only security check which is necessary in event system to be performed is in group communication while the `groupID` attribute in an event is specified. To avoid nodes to publish an event to a group which they are not member of that, the event system is required to verify the membership of the publisher in the group whose groupID is indicated in `groupID` attribute in the event.

Moreover, since we utilize encryption in Transport Layer to address the confidentiality concerns, no encoding methods are exploited in the event system. So the messages going through event system are plain text.

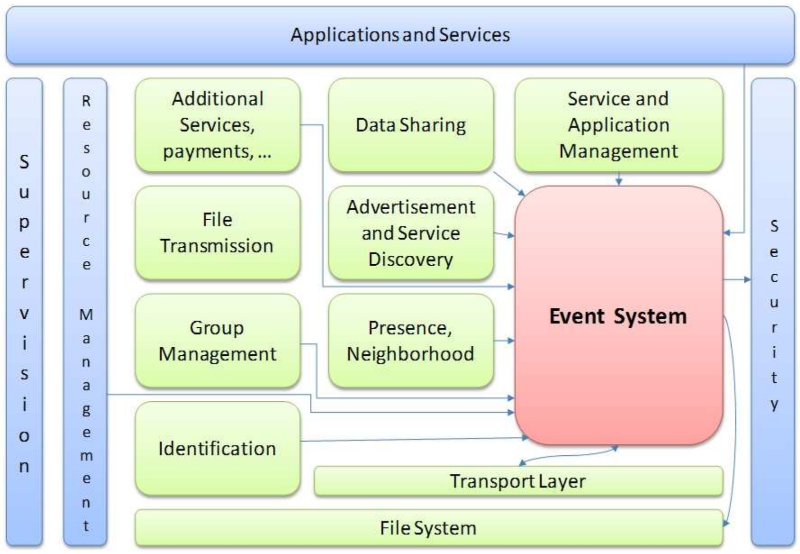


Figure 2.2: The Transhumance block diagram and event system interactions

2.2.4 Event System Resource-Aware Adaptation

Some event system functionalities will be deactivated when nodes are in power shortage or any other critical resource situations. These functionalities could be defined according to the methods and the algorithms by which the event system is developed. Besides, the power adaptation strategies must not have serious impact on the system, so the possible and potential damages should be anticipated and minimized. The energy adaptation strategies in event system will be explained in subsection 4.1.3 .

2.2.5 The Event-Based System Interaction with Lower- and Upper-Layers

As it is shown in figure 2.2 the event system is in interaction with several blocks in middleware and with other layers as well. These interactions are classified as:

The services event layer provides to other service blocks and layers: Event creation/destruction, event dispatching and notification, subscription and unsubscribing might be useful for blocks like Advertisement and Service Discovery, Service and Application Management, Group Management, and Data Sharing. Moreover, event system interacts with Resource Management block to manipulate the event system methods and functions based on the node available resources.

The services provided by other layers to event system: Sending and receiving events are done through the Transport Layer. The File System block also facilitates the accessibility of the files to the event system. Moreover, the Identification block provides information about the different usernames and together with Security block support the group authorization mentioned in subsection 2.2.2.

Then in further chapters a novel technique for event system will be presented. Other required functionalities and the adaptation and adjustment of the new method to fit to

Transhumance Event Management block specifications and demands are covered in next chapters.

Chapter 3

Chapar: A Cross-Layer Pub/Sub System for OLSR-based MANETs

3.1 Introduction

Chapar is a full featured publish/subscribe system in which a distributed event broker network is constituted on nodes to store and deliver the events in the network. In fact, Chapar is a cross-layer implementation over mobile ad hoc networks which utilize OLSR [3] as their routing protocol. The basic idea beyond this method is to use a set of nodes to form the broker network and put them in charge of multicasting the event from the publishers to the subscribers. They are also responsible to keep the events which are supposed to be stored in the network up to their lifetime is over. In other words, Chapar is an overlay network making its own multicast trees using OLSR routing table to forward and propagate the events to the subscribers. In this novel approach, we address the critical complications and problems imposed by mobility as well as tendency of tiny ad hoc nodes to failure or temporary unavailability. Besides, regarding the crucial ad hoc network resource constraints, the proposed solution comprises light calculations along with reasonable network overhead. Moreover, a replication mechanism is provided in order to preserve the subscription database and stored messages in case of node failure or disconnection. This mechanism is working based on mirroring MPRs to maintain the data in the network if some nodes are missed.

In the remainder of this section, we give a brief overview on basic concepts such as OLSR routing protocols, then we introduce some essential data structures and lookup tables as well as filtering techniques we utilize. Next, we present the methodology for subscription and detailed algorithm for multicasting the events to consumers. Finally, the extra features to fulfill different decouplings in publish/subscribe systems mentioned in section 1.1.1 are reviewed.

Figure 3.1 shows how Chapar implements a typical publish/subscribe system to an OLSR-based mobile ad hoc network.

3.1.1 An overview on Optimized Link State Routing (OLSR)

The OLSR [3] is a proactive and table-driven routing protocol for mobile ad hoc networks. In other words, the routing table of each node is updated periodically by routing messages

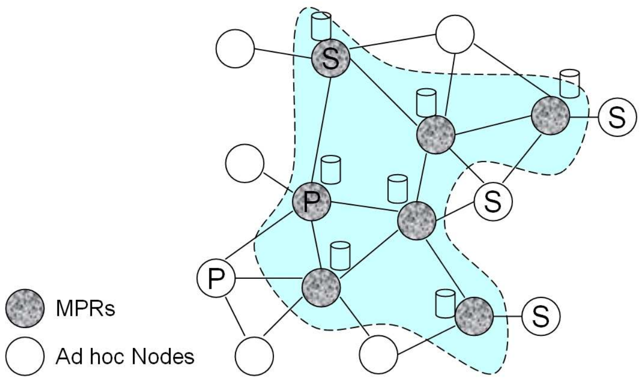


Figure 3.1: An example of Chapar network in which MPRs play the event broker role

coming from different nodes in the network. The essential idea behind this protocol is to reduce the number of flooded messages in the network significantly thanks to Multiple Relay (MPR) nodes. Basically, in OLSR, nodes need to report their link state to selected MPR nodes. Subsequently, the MPRs collect this information, generate control messages, and broadcast them efficiently through the network (Figure 3.2).

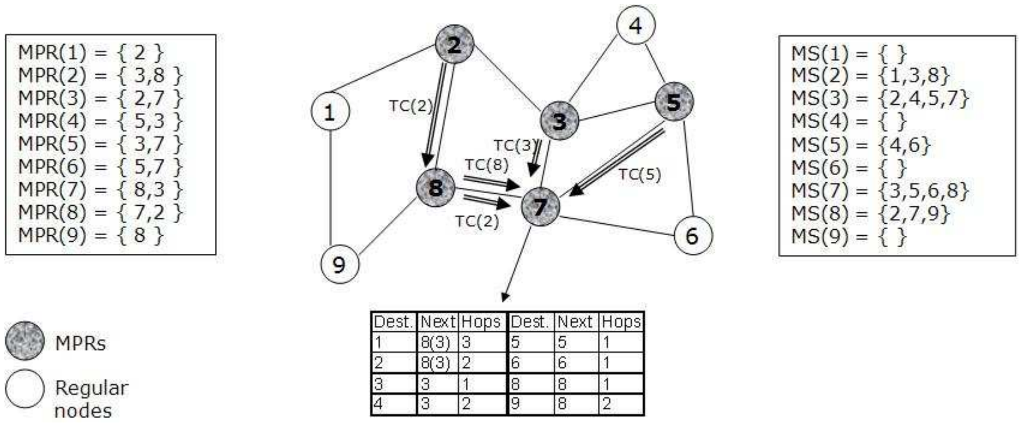
The definition for different OLSR terms and terminologies are as follows:

- Multipoint Relay (MPR): Each node i has a set of 1-hop neighbors, called multipoint relays MPR(i), which connect i to the 2-hop neighbor(s). MPR nodes process the HELLO messages but do not forward them.

- MPR Selector (MS) set: The MS(i) is the set of nodes which have chosen node i as their MPR.

- HELLO message: Each node sends HELLO messages periodically to its neighbors for link type discovery, neighbor detection and MPR selection. The default period time between two HELLO messages is 2 seconds.

- Topology Control (TC) message: The MPRs generate and forward TC messages to all nodes. TC messages contain the MPRs' MS set in order to accomplish the



MPR(1) = { 2 }
MPR(2) = { 3,8 }
MPR(3) = { 2,7 }
MPR(4) = { 5,3 }
MPR(5) = { 3,7 }
MPR(6) = { 5,7 }
MPR(7) = { 8,3 }
MPR(8) = { 7,2 }
MPR(9) = { 8 }

MS(1) = { }
MS(2) = {1,3,8}
MS(3) = {2,4,5,7}
MS(4) = { }
MS(5) = {4,6}
MS(6) = { }
MS(7) = {3,5,6,8}
MS(8) = {2,7,9}
MS(9) = { }

Dest.	Next	Hops	Dest.	Next	Hops
1	8(3)	3	5	5	1
2	8(3)	2	6	6	1
3	3	1	8	8	1
4	3	2	9	8	2

Figure 3.2: An example of OLSR routing protocol for an ad hoc network

topology declaration of the network. The TC messages will be received by all of the nodes attached to the network. By these TC messages each node may calculate its routing table. The default period time between two TC messages is 5 seconds.

OLSR in contrary to classic link state or classic flooding protocols, suits to dense network with random and sporadic network traffic. The routing tables are quickly updated when a link status changes. Therefore, if a node joins to or detaches from the network, all of the nodes are able to detect it and recalculate their routing table. The OLSR routing table allows a node to have a list of nodes which are connected to the network at the moment.

The default period time of HELLO and TC messages may be changed according to the mobility rate of the network. This mobility rate may defined by the number of link creation and disconnection of the nodes per time unit. The adaption of this time period by mobility rate may have a substantial difference in node power consumption.

In OLSR routing scheme the MPRs are acting as routers which forward messages from one node to another node. So the group of MPRs is always connected and forms a core network to connect all nodes together. The basic idea in Chapar is to use MPRs as the Broker Nodes (BN) to hold the subscription table, and forward the published events to their subscribers. And as it is mentioned, these BNs are all holding same information and doing same job. In other words, we have replicated nodes in the broker network. However, in some especial cases, BNs may not be just MPRs, and some other nodes will be selected as BNs. These special cases will be studied in subsection 3.3.4.

3.2 Overview on different data structures

3.2.1 Events and Filters

Events are considered as the basic unit of information (or an object) in an event-based middleware including set of fields and attributes which can be interpreted also as a header and payload. The header part is basically expressing some information related to the content like the message topic, and whatever information may be demonstrative for subscriber to opt whether they need to receive the event or not. On the other hand, consumers utilize filters to express their interest into specific kind of event. This is what we have already seen in subject-based event systems. Here we use some complex kind of filters providing multiple fields available for the consumer to accurately express its interest. However, the publish/subscribe system and the broker network distinguish the filters by their filter ID (FID) which is a array of bytes representing the hash result of each attribute of the filter. Likewise, an event is also recognized by the broker network by its Event ID (EID) which is calculated similarly but using different fields and attributes in the event.

$$FID = H(subjectInt)\|H(contentInt)\|H(senderInt)\|H(typeInt)\|H(groupIDInt) \tag{3.1}$$

$$EID = H(subject)\|H(content)\|H(senderID)\|H(type)\|H(groupID) \tag{3.2}$$

where the "$\|$" denotes concatenation and H represents the hash function with fixed length result which could be chosen based on possible number of variation of each field. The hash function arguments are also some event attributes which may be chosen by the filter as an Interest.

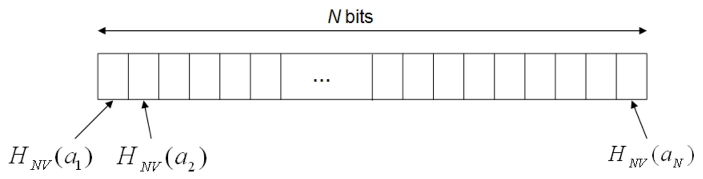


Figure 3.3: Nodes Vector (NV) is a bit array with length of N. The hash function $H_{NV}()$ maps each node address (a_i) to a bit index in bit array.

This kind of filter with its corresponding filter ID provide us the possibility to categorize the events into the event subject, event content, event senderID (publisher), event type, and the groupID that event belongs to. On the other hand, subscription can be performed by expressing the interest into each of them or combination of them.

3.2.2 Nodes Vector

Nodes Vector (NV) is a simple data structure which supports membership queries. As a matter of fact, a NV is a finite bit array in which each bit is appointed to a node. So the length of such a bit array (N) is chosen based on the maximum number of nodes that may join the network. We need also a hash function ($H_{NV}()$)or mapping table to map each node to index of a bit in the bit array. This hash function could be so simple if we have an organized system of addressing for nodes. For instance, if we consider the IP addresses of the nodes are selected from a IPv4 [43] class C address pool, the last byte of the node's IP address can be used as the index of its representative bit in NV.(Figure 3.3)

In fact, in Chapar, we utilize the nodes vector as a data structure representing an abstraction of nodes status for different functionalities. And since working with nodes vector and querying from it could be simply and efficiently done by logic operation like *and* (&) and *or* (|), using NVs are recommended. However, there are alternative data structures and methods like *Bloom Filters* which do not have limitation in the node address ordering, yet working with them as well as querying them need more calculation and suffers false positive error.

3.2.3 Bloom Filters

Bloom filter is a method introduced by Burton Bloom [33] in 1970 for representing a set of $A = \{a_1, a_2, \ldots, a_n\}$ of n elements (also called keys) in order to supporting membership queries. The bloom filter is a array of m bits, initially set to zero and will be filled by k independent random hash functions $h_1, h_2, \ldots, h_k$ with range of $\{1, \ldots, m\}(h_i : A \mapsto [1..m])$. So in order to position $a_i \in A$ in the vector with length m, we utilize the mentioned k hash functions whose result is the index of 1 in the bit array. (Figure 3.4) .

For membership query, to check whether the input element x (here in Chapar node x) is included in the set A, the QueryBitArray is made by calculating $h_i(x)$, $1 \leq i \leq k$ and compare it with bloom filter. If `(QueryBitArray & BloomFilter == QueryBitArray)`the node is assumed to be the member of the set. However, the false positive is probable in bloom filter method which means that there is a possibility that specific bits in bloom

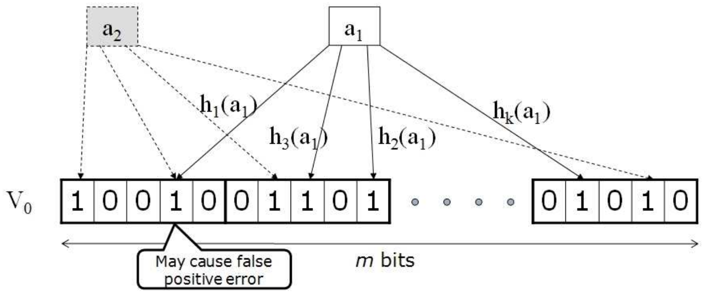


Figure 3.4: Bloom Filter

filter is set to 1 by the hash function result of the other members. Yet, the false negative is impossible, so in case the above equality is not true, we can certainly assume that the node is not in the set.

Although, Bloom Filters supports *adding* and *querying* functions (as it is already explained), *removing* an element from a simple bloom filter is impossible. Because, in case you want to remove the corresponding 1s from a bloom filter, since it could be belong to the hash function of another element, it may cause false negative error which is not allowed. Fan et al. in [34] introduced an extended version of bloom filter called *Counting Filter* in which for the array positions is extended to be more than one bit(to n bits). So it is considered that we can have a counter for each array position, so for inserting an element the counter respective bits will be incremented and for removing an element the counter will be decremented.

On the application point of view, in some cases the false positive error is allowed, although it may have some impacts on the performance and efficiency of the system. In order to minimize this impact, firstly we have a look on $P_{false\ positive}(x)$. The probability of false positive in bloom filter is[1] $\approx (1-e^{-kn/m})^k$. Thus, to minimize this probability with given m and n, we need to find a optimum value for k (the number of hash functions). Subsequently, the optimum $k_{min} = (ln2)(m/n)$ by which the probability is minimized to $(1/2)^k = (0.6185)^{(m/n)}$. Practically, the bloom filter may be highly effective and convenient when $m = 8 \cdot n$ and $k = 5$, so the probability will be $0.02168 \approx 2\%$ which is reasonable and acceptable in many cases.

Comparing Nodes Vector and Bloom Filter, NV is false positive error free, using one hash function instead of k hash functions, and we may easily delete nodes from the bit array. Besides, the required reserved bit array in NV is just equal to available elements, whereas in bloom filters, to have a reasonable probability for false positive error, we need to have a bit vector with the length of ≥ 8 times bigger than the number of available nodes. However, in NV we need to have an organized set of elements with an adaptable hash function to allocate just one single bit uniquely for each element in bit vector.

3.2.4 Filter Mapping Table (FMT)

The novel idea in Chapar is to utilize the bloom filters or nodes vector for keeping the subscription information in very simple and small database could be kept and delivered by nodes with a trivial resource engagement. FMT is a simple mapping table, including two

[1]For detail calculations please refer to appendix A.

columns one dedicated to Filter IDs (FID) and the second column presents its respective Node Vector[2]. Each BN in the network holds its own FMT which is similar to the others. The FMT will be eliminated once an BN is dismissed (the node is not MPR any more for example), conversely, if a node is selected as BN it will receive the FMT from its adjacent BN.

The *Subscription Nodes Vector (SNV)* is a NV mapped to a FID and representing a set of nodes which has subscribed to that filter ID. Thus, the FMT will be updated when subscription/un-subscription events are submitted. The records related to the FIDs with empty SNVs ($SNV_i = 0$) will be deleted automatically from the table. Figure 3.5 shows a sample of a FID Mapping Table.

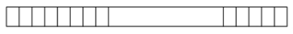
FilterID	SNV or SBF
0xFFFFFFFFFFFFFF	

Figure 3.5: FID Mapping Table (FMT)

3.2.5 Memorized Event Table (MET)

For time decoupling purposes, there are some events which has lifetime, which means that they need to be kept for a period of time in the network to notify those subscribers which may unavailable at the publishing moment or those may subscribe to an event after it is published. These *Memorized Events* are kept in a specific table called *"Memorized Event Table (MET)"*. Similar to FMT, MET is also retained by all MPRs and will be eliminated if they are dismissed and be inherited by a node is assigned as MPR. MET has three columns:

- The event or a pointer to where the event is stored on the MPR

- USNV (Unavailable Subscriber Nodes Vector): a representative of the subscribers which were absent in the publishing moment and their filter matched the event, so for each FID we have
$$USNV_j = SNV_j \ \& \ !ANV \tag{3.3}$$
and finally,
$$USNV = USNV_1 \mid USNV_2 \mid \cdots \mid USNV_L \tag{3.4}$$
where ANV stands for *Available Nodes Vector* which is a NV represents all nodes which are connected to the network at the moment (The list of connected nodes are retrievable by node's routing table). And L denotes the number of filters matched to the event.

- The event expiration time

[2]In Chapar specification and this document we suppose we have a systematic addressing for the nodes and we try to exploit the NVs instead of bloom filters. However, using BF requires some considerations and altering some algorithms. In section 3.4,7 we will review these alterations.

In fact, MET is a mapping table which maps the event to its respective USNV and expiration time. Updating USNV, when a subscriber is notified by the network its corresponding bit in USNV sets to 0. The record would be deleted from the MET at or after its expiration time.

3.3 Chapar Method Description

3.3.1 Subscription/Un-Subscription

In Chapar, a node creates its filter and dispatches it by a subscription message to the broker network. In fact, a subscription message is a special class of events which is created by the subscriber and sent to one of its BNs[3]. The incoming subscription event which contains the “Filter ID” along with the “Subscriber Address” is flooded in the BN network by which the BNs would be able to update their FMT for new subscription. The duplicated subscription messages will be discarded by BNs.

When a BN receives an subscription message, it compared the included FID with its FMT. if it is already in the FMT:

$$(SNV_i)_{new} = (SNV_i)_{old} \mid HNV_{subscriber}) \tag{3.5}$$

otherwise, a new record will be added by the name of FID and its corresponding SNV ($SNV = HNV_{subscriber}$)). HNV_i denotes then Host Node Vector which is a NV representing the host i.

We have similar algorithm for nodes interested in unsubscribing of some filters. Accordingly, the unsubscribtion message will be received by first BN and then flooded to the rest of the BNs, but a received unsubscription message will be looked up in the FMT, if it is not found, it will be dropped, otherwise

$$(SNV_i)_{new} = (SNV_i)_{old} \ \& \ !(HNV_{subscriber})) \tag{3.6}$$

this will unset the subscription bit pointed to the node in SNV_i.

Note that in case we use Subscription Bloom Filter (SBF) instead of SNV, the BN could not unsubscribe a node from SBF, since removing an element from BF is not allowed. In this case, either the counting filter should be exploited or we may not let subscribers to unsubscribe to an event.

After the FMT operation on the subscription message, it is matched against the MET table. If any of the stored events does match the filter (matching process is explained in the section 3.3.2 Lookup Process) the BN will notify the subscriber by its desired event(s) instantly (*Instant Notification*).

3.3.2 Publishing

As it is mentioned in Introduction part, in this approach we are trying to implement an overlay multicast network over MPRs as an Event Broker network to forward the published messages to their related subscribers. As it is shown in Figure 3.6, this publishing process is divided into three basic phases: Firstly, the publisher publishes the event to the broker

[3]if the subscriber is already a BN, the subscription message will be sent to its BN daemon

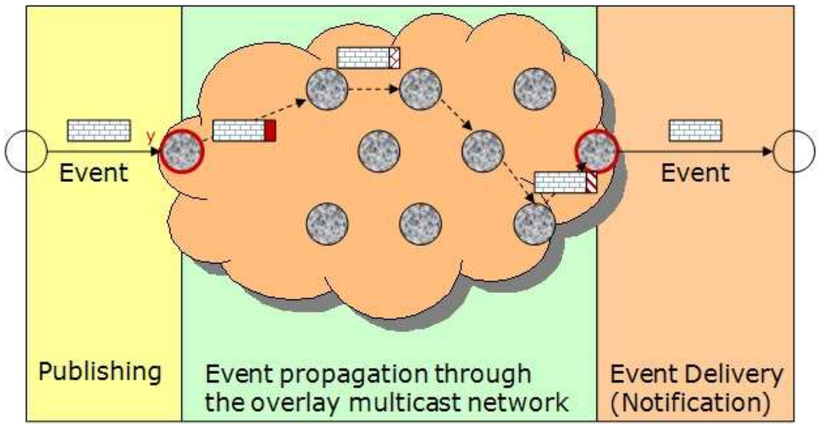

Event
Event
Publishing
Event propagation through the overlay multicast network
Event Delivery (Notification)

Figure 3.6: Publishing diagram in Chapar

network. So the event is dispatched by a publisher to one of its BNs (to avoid message duplication and multiple multicast trees). In next phase, the BN *looks up* the event in its FMT and based on the corresponding SNV, initiates the multicast tree to deliver the message to the subscriber(s). In the last phase, the *notification phase*, the BN forwards the message to the subscriber.

Lookup Process

Lookup process is started when a BN receives an event from its publisher. The FMT look up process should be inclusive, since a published event may be matched against several filter IDs interested in the event according to different event and filter attributes. For example, in (3.1) and (3.2), the available parameters for filters and events are subject, content, type, sender ID, and group ID. So an event may match against $2^5 - 1$ possible filters in FMT [4].

Upon an event arrival, the BN calculates the Event ID (EID) of the received event by (3.1) and verify it by all of the FIDs ($i \in [1, F]$) included in FMT [5]. So it starts filter matching process by checking each element j of the FID to be matched against each element of EID as follows:

$$FID_i[j] = EID[j] \& FID_i[j] \tag{3.7}$$

if all of the elements match against their corresponding ones which means ($FID_i = EID\&FID_i$). If it is,

$$ASNV_i = SNV_i \ \& \ ANV \tag{3.8}$$

and accordingly,

$$ASNV = ASNV_1 \mid ASNV_2 \mid \cdots \mid ASNV_L \tag{3.9}$$

where ASNV stands for *Available Subscribed Nodes Vector* and represents the subscribers which are available at the moment and may receive the published event and L is the number of filters which has matched the events. The ASNV will be used in next step for building the multicast trees.

[4]this number in Transhumance is 2^4 since the filters should have specific group ID

[5]the filter set of elements (S_F) is the subset of Event attributes set (S_A) ($S_F \subseteq S_A$)

If the $FID[i,j] = 0$, means that the filter i does not follow any interest in the element j. In fact in 3.7 we use the identity properties of the 0 in the & operation. However, the matching process algorithm suffers from false positive error (calculated in the Appendix B.). If we consider that the hash sum is fairly distributed in the 16 bit space the probability of false positive error is ≈ 0.01. Nonetheless, the impact of false positive on the network and event system functionality is trivial enough to be overlooked.

Multicasting

The lookup process is followed by comparing ASNV by the neighbor set. Prior to start multicasting, the BN should exclude the subscribers which are in its neighborhood from ASNV, because they should be notified by the BN and they do not need to be multicasted. So to notify the subscribers which are the BN neighbors (or the BN itself) we need to determine the Notified Subscribers NV (NSNV):

$$Notified\ Subscribers\ NV = ASNV\ \&\ (h(a_{node}) \mid (NNV \oplus BNV)) \tag{3.10}$$

where NNV and BNV denotes *Neighbor Nodes Vector* and *Broker Nodes Vector*, respectively.

The un-notified subscribers represented by *RNV (Residual Nodes Vector)* (calculated by (3.11)) are looked up in the BN's routing table to determine the next hop that the event should be forwarded to. Then, all of the subscribers are grouped by their corresponding next hop.

$$RNV = ASNV \oplus NSNV \tag{3.11}$$

For each group the event will be forwarded to its respective next hop along with a nodes vector RNV_i which represents the subscribers in that group i (3.12).

$$RNV = RNV_1 \mid RNV_2 \mid \cdots \mid RNV_n\ ,\ (n \leq numberofnode'sMPRs(orBNs)) \tag{3.12}$$

In other words, we are trying to constitute a rooted tree in which the root is the publisher MPR, the next hops will be the child vertices (nodes) and each group includes the subscribers that may be routed by that child vertex (next hop). Thus, when the RNV for each group is settled, the events will be forwarded to the next hop.

The next hop performs the similar operation as (3.10) by replacing the ASNV with the event RNV. So

$$Notified\ Subscribers\ NV = RNV\ \&\ (h(a_{node}) \mid (NNV \oplus BNV)) \tag{3.13}$$

$$RNV_{new} = RNV_{old} \oplus NSNV \tag{3.14}$$

and this operation will be similarly repeated by using RNVs instead of ASNV to multicast the event through the network (3.14) and (3.13).

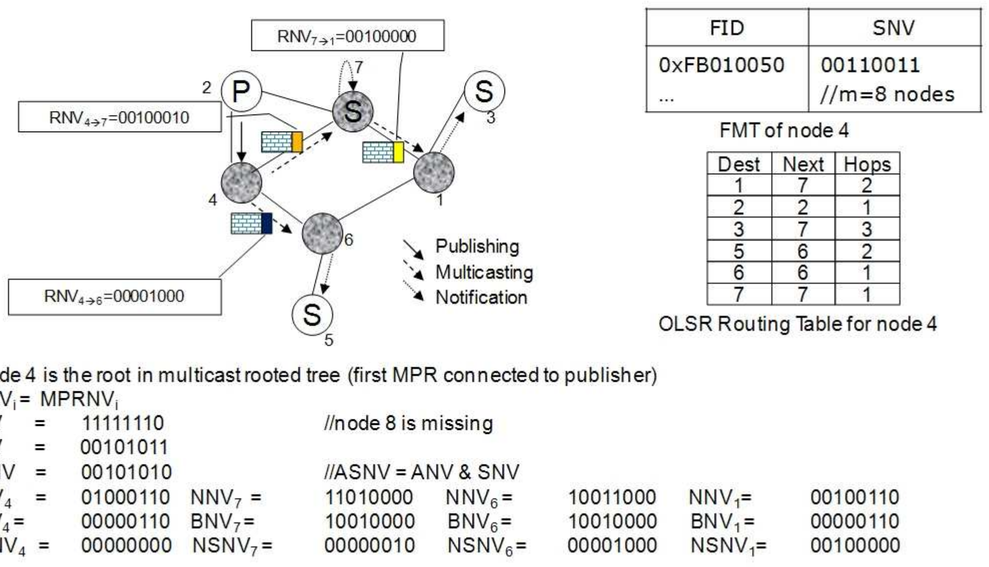
Figure 3.7 shows an example of multicasting a published event in mobile ad hoc networks. In this example, we assume that the maximum number of nodes may not exceed 8 nodes (N=8), so the SNV length is set to 8 bits. The ANV shows that the first 7 nodes are available and the 8th one is missed. The publishing process starts when the publisher (node 2) publish the message to one of its MPRs (node 4 is selected). Node 4 calculates the EID of the incoming event and looks that up in FMT and subsequently retrieves the corresponding SNV. Here in this example, the SNV illustrates that the 4 nodes are subscribed to the event, but according to ASNV, one of them is missing. Firstly, node 4 tries to generate the Notified Subscribers Nodes Vector (NSNV) based on (3.5). Then, to build up the multicast tree, node 4 looks for the next hops for the subscribers indicated in ANSV in its OLSR routing table. The results will be divided into groups based on their next hops (node 6 and 7). It sets the RNVs based on the groups ($RNV_{4 \to 6}$ and $RNV_{4 \to 7}$) and dispatches the events together with their related RNV. Nearly same thing takes place in next hop using (3.13) (The results for each step is demonstrated in the Figure 3.7). Accordingly, Node 7 finds itself in $NSNV_7$ and notifies the event by looping back the event to its subscription daemon and prepares the next RNV to forward the event to next hop attached with its corresponding RNV.

Taking advantage of OLSR, we avoid loops in this multicast overlay network and subsequently nodes are relieved from duplicated messages. Besides, using RNV, we degrades the number of FMT lookup process to once performed by the first MPR (the tree root). Finally, the trivial network overhead (RNVs) as well as quite light calculation suits this method for mobile ad hoc networks.

Notification

Notification, as it is shown in Figure 3.6, the last part event publishing, is generally called to the event delivery process from the broker to the subscriber. The notification is taking place based on (3.10) and (3.13), along with the publishing process and before the event forwarded to the next hop (in case the MPR or BN is the subscriber as well, the event should be received by the broker daemon of BN and will be looped back to its subscription daemon).

In Chapar, using the fact that MPR network (broker network) is always connected to



FID	SNV
0xFB010050 ...	00110011 //m=8 nodes

FMT of node 4

Dest	Next	Hops
1	7	2
2	2	1
3	7	3
5	6	2
6	6	1
7	7	1

OLSR Routing Table for node 4

```
// node 4 is the root in multicast rooted tree (first MPR connected to publisher)
//BNVi = MPRNVi
ANV    =   11111110              //node 8 is missing
SNV    =   00101011
ASNV   =   00101010              //ASNV = ANV & SNV
NNV4   =   01000110  NNV7 =      11010000   NNV6 =     10011000   NNV1=      00100110
BNV4=      00000110  BNV7=       10010000   BNV6=      10010000   BNV1 =     00000110
NSNV4  =   00000000  NSNV7=      00000010   NSNV6=     00001000   NSNV1=     00100000
```

Figure 3.7: An example of event multicasting in Chapar. Note: In this example MPRNV is same to BNV since MPRs are BNs.

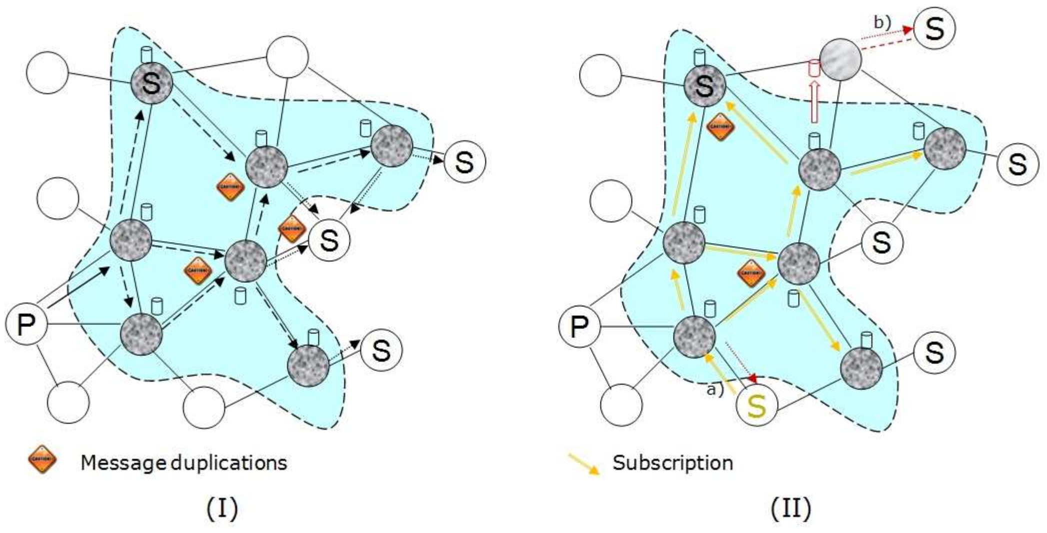


Figure 3.8: Memorized Events in Chapar

the subscribers, we utilize one-hop notification; this helps to preserve the network from message duplications and network overhead. This feature shows off in notification of memorized events which will be explained in the following subsections.

3.3.3 Memorized Events Publishing and Notification

As it was mentioned in previous sections, the memorized messages should be kept in BNs for their lifetime. MET is the reference table for memorized messages which is available in all of MPRs comprising the broker network. The memorized events despite of real-time event publishing, floods the MPR network to make sure that all MPR nodes receive the event and store a copy of them in their MET for redundancy purposes (Figure 3.8.I).

When an event is received by the first MPR, it is checked weather it has a lifetime or not. If it does not, it will be multicasted to the subscribers as it is already explained, otherwise, the first MPR checks the event in its FMT and retrieves the SNV_j and calculate $USNV_j$ based on (3.3) and subsequently $USNV$ by (3.4) and save them in MET.

The message notification is done by (3.13) for subscribers who are connected at the moment (Figure. 3.8.I). They are two kind of subscribers that still may receive this event during its life time:

- *Future Subscribers* are called to the subscribers who would subscribe to an event after the event is published. To notify the buffered message for these subscriber, the MPR which received the subscription event and took it into account for updating its FMT will check whether `(FID == EID[i] & FID)` to verify if the subscription is included to any of the memorized event or not. If it is, the BN will notify the subscriber immediately (instant notification) (Figure 3.8.II.a). Other BNs will not perform this verification because the subscription sender ID is not their neighbor.

- *Unavailable Subscribers* are called to the subscribers who has subscribed to an event prior to the publisher publishes the event and were not available at the publishing moment. They will be notified by their adjacent MPR (and indeed BN) as soon as they connect to the network. Afterwards, they will be removed from USNV of all METs in BNs, since all BNs are acknowledged about the subscriber availability by their routing table. Thus, during this so-called *Purging Operation*, the BNs need to periodically calculate the NSNV and USNV for each record of their MET as follows:

$$NSNV = USNV \ \& \ NNV \tag{3.15}$$

$$USNV_{new} = USNV_{old} \oplus (USNV_{old} \ \& \ ANV) \tag{3.16}$$

For example in Figure 3.8.II.b when an unavailable subscribers return and connects to the network, the new node is chosen as MPR, so it subsequently inherits the FMT and MET, calculate its NSNV, update its MET and notify the subscriber. Due to the next routing table updates, other MPRs will be informed about the presence of the unavailable subscriber so the updated USNV or $USNV_{new}$ based on 3.16 does not contains the new notified node.

Taking advantage of the one-hop notification, there is duplicated event delivery neither for Unavailable Subscriber nor for Future Subscribers. Additionally, using OLSR periodic routing update, the MPR that performed the notification does not need to acknowledge other MPRs, because they can be informed about new subscriber by the ANV extracted from the updated routing table. However, the forwarding memorized messages in broker network and between MPRs are done by flooding, so duplicated messages are expectable and MPRs will discard this sort of messages.

3.3.4 Some Special Cases

In some especial cases we may not define MPRs as BNs. So we need to generalize BNs to not only MPRs but also some selected nodes. Thus, to select the BN nodes a method for BN election should be provided optimized enough to be fit in our consideration for mobile nodes. In fact, These especial cases are regarding to some situation that number of MPRs (M) are less than number of required BNs (N_{BN}) predicted for the event system($M < N_{BN}$). This includes networks with no MPR ($M = 0$) or just one MPR ($M = 1$), since the minimum required number of BNs is two ($N_{BN} \geq 2$) due to redundancy purposes. Because of the fact that the brokers are mirrors of each other and in case of broker node(s) failure or disconnection, there should be other BN(s) to preserve the databases from losing.

These special cases which happen quite often may jeopardize the reliability of the event system. In the following we review the occasions in which $M < N_{BN}$, and their corresponding election methods.

Single Node

Due to mobility, node might be frequently secluded from the network and be disconnected from the rest of the nodes for a while. This causes a considerable unreliability for some applications dealing with important information that requires the basic quality of service and guarantee for the message delivery. They might also subscribe to some events in the time they are apart from the network without knowing about that. To counter such inconveniences, when a node detects that it is alone ($NNV = 0$) it assigns itself as a

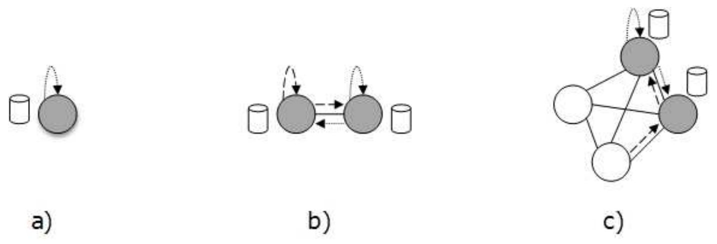


Figure 3.9: Special types of networks where there is no MPR in the network. a) Single node, b) Two connected nodes (full-mesh network $N = 2$), c) Full-mesh network ($N = 4$)

BN and send its subscription as well as published events to its broker daemon (figure 3.9.a). In the merging time, thanks to the broker check-out procedure as well as table integrity check procedure, the rest of the nodes will be notified by the new subscriptions and memorized events.These procedures are explained in the subsequent sections.

This feature helps applications to publish their *important* messages with a reasonable life time regardless if the node is connected or not, and be confident that their messages are not lost and the subscribers will receive it after nodes reassembly.

Full-Mesh Networks

In full-mesh networks, once all nodes are connected to each other and communicate directly without hop forwarding, we may not have any MPR (M=0).

$$NNV_i = ANV \oplus HNV_i \tag{3.17}$$

Such network might be formed quite often when all nodes are gathering in a limited area such as meeting rooms or any rendezvous spots. In this case, BN shall be selected by an election algorithm. We propose an address-based election to avoid any extra signaling and unnecessary network and resource overhead. Therefore, the broker nodes would be simply those with lower index in ANV. In this paradigm, each node first check whether it is in a full-mesh topology or not (if $MPR_i = 0$ or if (3.17)), then based on the election algorithm it will check if it has the lowest index in ANV or not, if it is it will be self-assigned as BN. It also finds its BN set with same technique. Hence, the N_{BN} nodes indexed from the beginning of ANV will be chosen as the BN set of the nodes. Afterwards, all of subscriptions and published messages will be sent to them and they will handle and forward the events to their corresponding subscribers.

Networks with one MPR

Due to redundancy purposes, we need at least two broker nodes ($N_{BN} \geq 2$). In OLSR the nodes are not able to realize how many MPRs are available in the network. So it is not feasible for nodes to know how many redundant nodes are available as MPR to be assigned as BN. Thus, the question is: could it be possible to guarantee $N \leq 2$ BNs in the network?

One solution is providing a designated node to calculate the number of brokers by some extra signaling and probably complex algorithm. This solution is not convenient since we are avoiding any kind of centralized architecture in our distributed system. The alternative solution is using the OLSR routing table. The hop count column in the routing table enables nodes to figure out whether the network contains more than one MPR or not. In fact, the greatest number in hop count column minus one is the lower bound for number of MPRs. So,

$$M \geq max(hc_i) - 1 \tag{3.18}$$

Where hc_i is the hop count number for destination node i.

So, if we look to the boundary of this inequality, when $(max(hc_i) > 2)$ it means that at least two MPRs are present in the network which is the minimum number of BNs the network needs for replication. But if $max(hc_i) = 2$, this conclusion could be drown that there is only one MPR in the network, if the node itself is not an MPR (for an MPR $max(hc_i) = 1$ (an star-like network)) and indeed nodes $MPR_i = 1$. So in these situation nodes starts performing the election and choose a replica for the current MPR. This election shall be done in both MPRs and regular nodes. After election, nodes are aware of the BN replica, although they prefer their MPR to be as their BN to dispatch their events to, since their MPR is accessible directly in compare to their elected BN which could be one hop away.

3.3.5 Broker Nodes Table Integrity Check

The BN tables which are supposed to be the same since they are mirrored could be different because of many reasons. Network partitioning would not let the flooded subscriptions/unsubscription to be received by all BNs, so their FMT may differ from each other. The frequent transient node disconnection may also affect the BN tables depending on the node position in the network. Packet loss in wireless communication could be another reason, since we use unreliable UDP connection for forwarding the events through the network. Moreover the divergence of the tables may be raised over time because of inheritance process which will be explained in the next subsection. Thus, coping with this phenomenon, we need to have periodic process to synchronize the BN tables together. The proposed algorithm to do so is shown in figure 3.10. In this algorithm, each BN tries

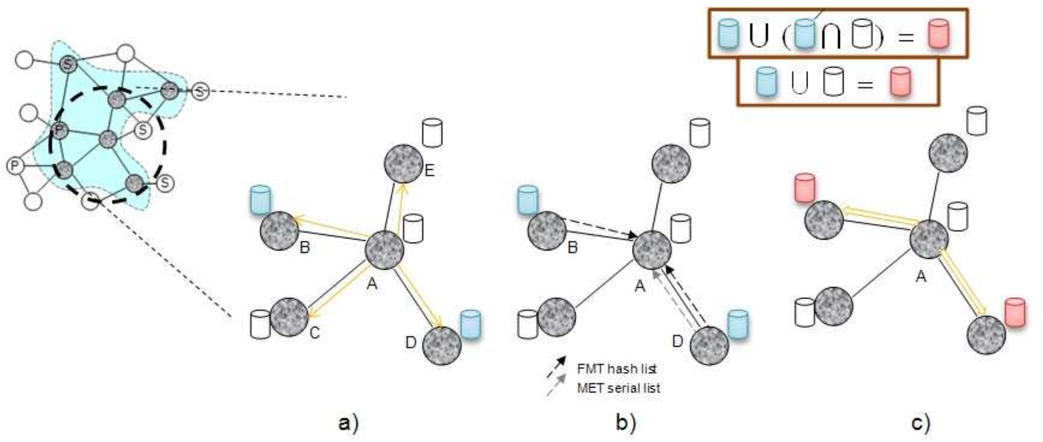


Figure 3.10: a) node A sends an Integrity Check req. to all of its adjacent BNs b) node B and D send their FMT hash list and MET serial list c) finally node A notifies them based on its FMT and MET tables

to synchronize itself with its adjacent BNs. The repetition of such content synchronization process leads the network to have an almost synchronized broker system in real time.

To see how the table integrity function works we review the main three steps in figure 3.10. In thisexample we assume:

$$D(A) = D(C) = D(E), \tag{3.19}$$

$$D_{MET}(A) \neq D_{MET}(D), \tag{3.20}$$

$$D_{FMT}(C) \neq D_{FMT}(D) \neq D_{FMT}(B) \tag{3.21}$$

where $D_{MET}(X), D_{FMT}(X)$, and $D(X)$ denotes the MET content, FMT content, and FMT and MET content (in general) of node X, respectively.

In the first step, node A calculates the hash result of each table ($H_{IC}(D(A))$), encapsulate them in an event, and flood it to the broker network. The adjacent broker (node B, C, D, and E) will receive this and compare it with its own tables' hash results (figure 3.10.a).

$$H_{IC}(D(A)) = H_{IC}(D(C)) = H_{IC}(D(E)) \tag{3.22}$$

$$H_{MET-IC}(D_{MET}(A)) \neq H_{MET-IC}(D_{MET}(D)) \tag{3.23}$$

$$H_{FMT-IC}(D_{FMT}(A)) \neq H_{FMT-IC}(D_{FMT}(C)) \neq H_{FMT-IC}(D_{FMT}(E)) \tag{3.24}$$

If they are the same, the integrity check process will be terminated (node C and E)(3.22). But if the FMT hash result differs (node B, and D)(3.24), an event along with the list of hash results of each record is created and sent to node A. However, if the MET table disaccords (node D) (3.23), the created event includes the list of the event serials it stored in the table (figure 3.10.b). Node A will compare each hash result (for FMT) and event serial numbers (for MET) to its own tables. For the FMT it sends node B and D each record of FMT whose hash result was not listed in their response (which means the record does not exist in the their FMT table) ($D_{FMT}(A) \cap (D_{FMT}(D))'$). To handle the node D integrity check response for MET, node A disseminate the missed memorized events ($D_{MET}(A) \cap (D_{MET}(D))'$) one by one along with their new lifetime (figure 3.10.c). The received events will be added to node D MET table. Now, its table is completed by node A events. To show integrity check process in this example by mathematical semantics, if we assume that node X starts the integrity check by sending its hash table to node Y and $D(X) \neq D(Y)$, we have:

$$D_{new}(Y) = D_{old}(Y) \cup (D(X) \cap (D_{old}(Y))') \tag{3.25}$$

$$D_{new}(Y) = (D_{old}(Y) \cup D(X)) \cap (D_{old}(Y) \cup (D_{old}(Y))') = D_{old}(Y) \cup D(X) \tag{3.26}$$

In conclusion, in this paradigm, each broker notifies its fellows about the information it has and they do not have. Besides we should note that the table modification in node B is completely accumulative (figure 3.10.c). It means that nothing will be deleted or altered; the new information is just added to previous ones. Since there is no reference that we can rely on to verify which table is correct, we need to follow this strategy although it may add some false positive errors to the tables. For instance, if node A has not received an unsubscription message whereas node B has received, it may change node B "correct" FMT table as the previous subscription is still valid. Although such wrong information imposes a trivial overhead on the system, it does not have an impact on the system functionality. These false positive errors are just restricted to the FMT table. The memorized events in MET with their updated lifetime will not suffer from this genre of errors.

3.3.6 Broker Node Appointment and Dismissal

Because of mobility of the nodes, the network topology is frequently altering. Consequently, in OLSR, the nodes according to their position in the network, would be appointed as MPRs or discharged to be the regular nodes. So when a node is appointed as an MPR, it will be a new BN in broker network. The new BN requires the broker system tables which would be accessible from one of its adjacent BNs (or MPRs). The process of handing over the tables from the senior BNs to new BNs is called *inheritance*. In this process, both FMT and MET table contents are dispatched to the new BN after it sends an *inheritance request*. The inheritance request is just sent to one of the adjoining BNs and the received is required to answer this message. Once the tables are completely transferred, the new BN would be able to process the incoming published events and forward them to their subscribers.

On the other hand, the broker nodes will be dismissed to the regular node due to network topology alteration. Before a dismissed node wipe out its table form the memory, it notifies its neighbor BN about the information it has including subscription table (FMT) as well as memorized events table (MET). This notification called *check-out* procedure is similar to integrity check procedure.[6] In check-out procedure node would be able to notify its adjacent broker about the tables it has. This information will be propagated by integrity check process to rest of the brokers. Thanks to check-out procedure, neither any subscription information, nor any memorized event will be lost in the network. Finally, as soon as the check-out process is finished the node is allowed to drop its tables from memory (figure 3.11).

There are some considerations regarding to mobility in BN appointment and dismissal. Since the BN appointment and dismissal is costly in term of number of exchanged messages and events between new BN and the former ones, an optimization strategy is needed to reduce the number of BN appointments and dismissal. As it is mentioned the BN appointment/dismissal is triggered by the mobility and based on the OLSR concept. So basically, our contribution in this optimization strategy should be focused on the *Appointment Transient Time (ATT)* period in which the node is appointed to be a BN as well as *Dismissal Transient Time (DTT)* period in which the node is discharged of

[6]In Transhumance, the check-out implementation is done with trivial differences to integrity check procedure which will be explained in section 3.4.5.

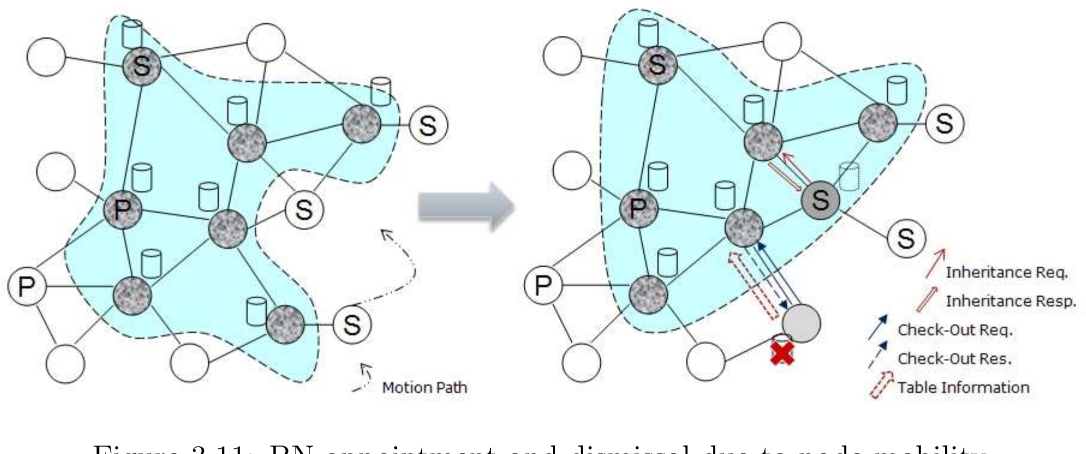


Figure 3.11: BN appointment and dismissal due to node mobility

being a BN. In fact, ATT must tend to 0 ($A \rightarrow 0$) in order to have a connected broker network. Long ATT will cause the disconnectivity of publishers and subscribers to the broker network. So, as soon as the node is assigned to be the BN, it has to start working by triggering the BN daemon which would be followed by the inheritance request since its FMT is empty.[7]

However the story is different for DTT. DTT period should equals to a reasonable value based on the network mobility rate. Nodes may become a BN and obtain the tables and dismissed and again due to network mobility can be reassigned as the BN. After BN dismissal, node will keep the tables for DTT. Once the DTT period is over, the tables are not valid anymore and should be dropped after the check-out procedure. The reasonable DTT may reduce the appointment/dismissal traffic in time of node BN reassignment. If the node be reassigned in the DTT period, it does not need to send the inheritance request and it may use the tables it already has. Even though the tables are outdated, they will be updated after a short while due to other BNs integrity check process. Choosing an appropriate DTT value will be more discussed in section 3.4.2.

3.4 Chapar Method Analysis

3.4.1 General Considerations

The Chapar structure design is inspired by the Transhumance middleware block design in which event system is designated as a module to support asynchronous communication in the network. Moreover, the group-natured architecture of the Transhumance makes the event system a critical module for group communications. The group communication is fundamentally categorized in topic-based publish/subscribe system. But since the event system is not only exploited for group communications, the topic-based pub/sub system in Chapar should support more features and is expected to have a satisfactory flexibility for higher-level applications. Hence, Chapar is a multidimensional event system which supports different communication modes such as point-to-no-specific-point (Pub/Sub system with space decoupling) point-to-point, point-to-multipoint (multicast) and combination of them. It is multidimensional in the sense that the proposed subscription system can be applied to any kind of events with known structure. Actually in this paradigm the events should be described by the set of possible interests as the characteristics of the event. For instance, the example in the subsection 3.2.1, the event is delineated with event subject, content, group, sender and the type.

Categorizing Chapar into the basic architectures, it does not necessarily fit in any classical architecture we have described for event systems. In fact, Chapar could be called as a *distributed system with replicated databases*. It is distributed because it utilizes multicast to route the events from their publisher to their subscribers. Meanwhile, there are many (≥ 2) nodes that hold the whole database of subscription and memorized messages which seems like event systems with centralized architecture. Although the latter property of the Chapar may not a good approach for tiny nodes (like sensors) but it is an essential factor to guarantee the required quality of service.

[7]The node just check FMT table, because rest of tables could be empty but if FMT is empty it means that node does not have the tables at all, since all of the tables are created and dropped together. However the inheritance request is asking for all of the tables.

Chapar is using OLSR routing protocol to make its multicast trees and form its broker network and support mobility. However, the networks with other proactive (table driven) routing protocols (for instance, DSDV [35]) may use Chapar with some modification in algorithms. The important thing that Chapar needs from whatsoever routing table it refers to, is that a frequently updated routing table is required to generate the multicast trees to forward the events to mobile nodes as well as a mechanism to define the broker nodes and a connected broker network.

And finally, we should declare that the main goals considered for Chapar are basically limited to prototyping an efficient and full-featured event system in mobile ad hoc network as a module in a compound middleware that provides a considerable reliability in moderate-rate mobility and medium-scale network environment. However, Chapar functionalities are going beyond this limit and provide much more services than it was required. We analyze the Chapar and its functionalities in different scopes in following subsections.

3.4.2 Mobility

Chapar supports mobility using OLSR routing protocol. In Chapar, there is no fixed multicast tree to be broken by node mobility or needed to be repaired because of node detachment and reattachment. The multicast tree is build virtually when an event is published. This virtual tree starts from the node's BN (or MPR) to all of the subscribers. To make sure that the multicast tree is valid for a published event, we exploit the fact that the mobility rate is significantly lower than the propagation delay from producer to consumer(s). Otherwise, the created tree for the published event would not address correctly to the displaced consumer(s). With this paradigm we avoid extra signaling for mobility. For instance, as it is mentioned in section 1.2.2, the survey about JEDI implementation, two operation “move-in” and “move-out” were used to support the mobility. These operations impose significant traffic to the network especially if the nodes frequently detach from their DS and reattach to a different branch of the DS tree.

To handle mobility with different mobility rates, OLSR uses two constant varibales namely `HELLO_INTERVAL` and `TC_INTERVAL` to adapt the routing daemon to different mobility rates. Thus, Chapar as a higher-level service should not be worried about the high mobility rates for its publishing system. However the high mobility rates may induce the frequent network topology changes. Accordingly, the central MPR network changes more often (new nodes will join and some nodes would be dismissed). This would cause more BN appointment/dismissal in the event system. In spite of the fact that we introduce the DTT to tackle the fast mobility rates, still some extra signaling (mostly regarding to table integrity check) may be required since the BNs' table content synchronization will be affected. To deal with this problem, we need to consider a trade off between table synchronization and the imposed traffic. Chapar provides a constant called `TABLE_INTEGRITY_CHECK_INTERVAL` which is the time duration the BN daemon waits between each table integrity check. Big `TABLE_INTEGRITY_CHECK_INTERVAL` value may cause the tables contents not really symmetric in high mobility rates, however, the small `TABLE_INTEGRITY_CHECK_INTERVAL` will leads to more event system traffic, although the table integrity check is designed in the way that if the tables are the same (tables in both parties have same hash results) the integrity check processhas trivial impact on the syste. Firstly because the integrity check request data diagram is an small message (containing the hash result of the tables) and secondly, once the tables are same the process will be immediately terminated.

The rest of the Chapar mobility considerations are concentrating to network partitioning problems. So in the next subsection we will focus on what may happen to the network due to the mobility and what the countermeasures are.

3.4.3 Network Partitioning

One of the major challenges that almost any kind of overlay network encounters on mobile ad hoc network is network partitioning. Network partitioning is generally called to a situation that the ad hoc network is divided to some subnetworks due to mobility. So the overlay network design has to cover all of the incidents that may happen not only in subnetwork seclusion period but also in time of subnetworks merging. Network partitioning takes place quite often during the time of network operation especially if the network is not dense enough and nodes moves in a quite random pattern. Node failure depending on the node position may also help for network partitioning.

For centralized overlay network, network partitioning may cause unreachablility of the centralized servers for clients which are located in different sub-networks. However, in distributed system nodes suffer from incompleteness of the information and services they need. So, In Chapar we proposed using many replicas to be appointed as BN nodes to counter such situation. With this architecture services are available in all of network partitions and servers could work with proper information.

The partitioned subnetworks may be:

- A single node: A disconnected single node could be considered as a sub-network. Since there are some services installed on the node and they are working regardless of the node connection status, the event system as a middleware module should be active to serve the applications. The explanation about how Chapar works on secluded single node is presented in subsection 3.3.4.

- Full-mesh small sub-networks: In full-mesh sub-networks, as soon as the sub-network is formed the nodes start the BN election process and nodes send their subscription messages as well as their memorized messages to the elected BNs. And the rest of the operation follows the basic Chapar methodology.

- Subnetworks with MPR: These subnetworks are considered as a complete networks, so they will follow the basic Chapar methodology in which the MPRs are assigned as BNs and handle the subscription, publishing and notification of the memorized messages.

Thus, in any kind of subnetwork, the event system is alive and its overlay network will be formed on the top of the BN nodes. In other words, disconnected nodes or subnetworks will not suffer from the absence of event system. On the other hand, when the subnetworks merged, thanks to the integrity check procedure as well as the checkout process, the new information regarding to subscription and memorized events will be exchanged between BNs and subsequently the event databases will be updated with all information. This enables the network to work properly.

For instance, figure 3.12 shows an example in which the partitioned networks are merged. As it is shown each partitioned subnetwork has its own event system database. Our approach is to have a compounded network whose database equals to the union of databases of each subnetwork. In the first step, node A adjoins the network $\overline{\underline{BC}}$ [8], since the node A is not BN anymore (we assume after the election node B and C will remain as the BN and A will be dismissed), it will start check-out process. So the information existed in node A database which is not in B database ($D(A) \cap D'(B)$) will be delivered to node B and will be saved there (I). Meanwhile, node B and E would connect and node C needs to check-out as well. So if B and C have had integrity check before check-out process (assuming DTT is long) $D(C) = D(B) \cup D(A)$, otherwise $D(C) = D(B)$. After check-out process, node E database is completed by node C database (II). Right after node B connection to node F, node F should change its status to an MPR and subsequently a BN. It inherits the database from G as a successor (III). So in next integrity check, which may start by a request coming from node E it will complete its database by all of the subscriptions and events of the partitioned subnetworks (IV, V). The integrity check will be repeated during the time and after a while the BN databases would be synchronized.

3.4.4 Redundancy

As it was mentioned, Chapar is a distributed event system with redundant databases. However, nodes in this distributed system are symmetric, so it can be considered as “a replicated event system”, although it does not follow the classical replication architecture in which events are published to *all* of the replicated servers and they are all in charge to forward the events to their subscribers [17]. In the following we try to review three

[8]The notation of $\overline{A\underline{BC}}$ denotes the a connected network whose members are node A, B, and C and node B and C are the BNs

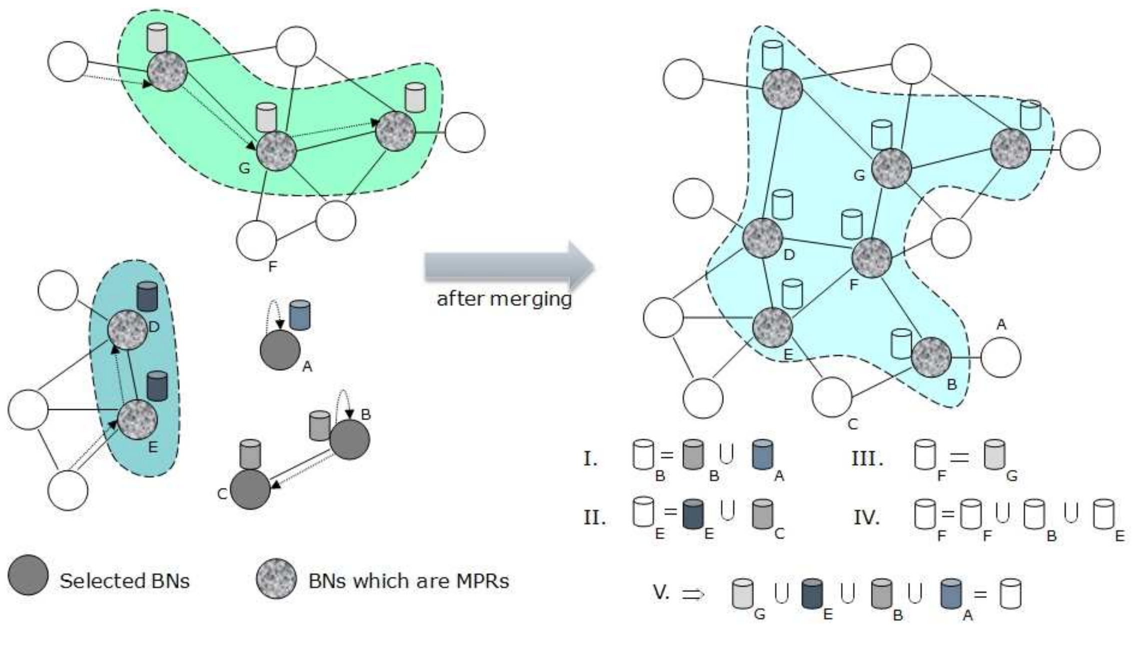


Figure 3.12: Network partitioning and merging example

desirable properties expected from replicated system (mentioned in subsection 1.2.1) in Chapar.

Orderedness

In many applications the order of arrival of the published events are important. The applications expect to receive the event is the sequence they are published. To show the importance, consider stock's price which is frequently changed. In this situation the sequence of arrived events which indicate that the stock price drops or climbs is very important. The situation would be more severe while we are working with real-time applications and we are not able to sort the events according to the events sequence numbers (e.g. games, stock market, and even multimedia).

In Chapar we have defined three kind of essential messages for pub/sub system. Subscriptions, real-time published events, and memorized events. The order of subscription messages is not really important. Subscription messages are sent to express the interest of user to an event. These messages are short and single, so there is no sequence of subscriptions to worry about the order of their arrival. On the other hand, the replicated system is not used for delivering the real-time published messages. They are dispatched through a single virtual multicast tree over the broker overlay network. So, since the path is the same for event delivery, we may not have problem of event reordering, unless an event is missed due to the wireless link packet loss in the network.

However, reordering in memorized events could be considerable. To look over the orderedness on memorized events, we need to divide them into two groups: The ordernedness in event publishing time (real-time notification), and the orderedness for the event subscribers who were absent and reattached to the network and/or the new subscribers who subscribed to some events which was already published (memorized notification).

As it was explained in section 3.3.3, the memorized messaged will be flooded in the BN network after they are published to first BN. For real-time notification, we may say that it is so unlikely to have reordering events. Because events are send through almost one path to the subscribers if we consider the fact that the propagation delay is negligible and mobility speed is slow enough to have no impact on change the delivery path from publisher to subscriber(s).

However, for the memorized notification, the probability of reordering is substantial. This reordering could be induced by BNs' MET content asynchronization, the order of the events stored in MET, and node mobility based on random pattern. For instance, consider the network $\overline{A\underline{BC}D}$ and series of eventsE_1 which contains a sequence of events from 1 to k, $E_1 =< E_1^1, E_1^2, \ldots, E_1^k >$, and node X is subscribing to E_1. Node D publishes E_1, while whose subscriber X is absent at the moment. When node X connects to the network $\overline{A\underline{BC}D}$, it receives some memorized events E_1^4 to E_1^{10} from its adjacent BN (node $\underline{B}$ for instance). Node X would connect to node $\underline{C}$ due to the mobility after a short while, subsequently, its BN is changed and it will receive some more events from E_1^1 to E_1^3 from $\underline{C}$. Therefore, as you may see reordering in the memorized messages is possible in memorized notification.

Nonetheless, recall the fact that memorized events may have a considerable resource overload on the network, so using them is practically allowed just in special cases that persistency of the event in the network is important (e.g. service announcements and etc.). Thus for services, using the *sequential* memorized messages is not pragmatically

	Orderedness	Consistency (avoiding event duplication)	Completeness
Subscription	$\surd$	suffers from duplicated copies in broker network	$\surd$
Real-time Event Publishing	$\surd$	$\surd$	$\surd$ with some exceptions
Memorized Event Publishing	Event reordering (partially)	suffers from duplicated copies in broker network and notification (partially)	$\surd$

Table 3.1: Desireable properties check list for different event system functionalities in Chapar.

recommended. Besides, memorized events are not applicable for real-time applications, so even in such cases, the sequence numbers implemented by the application may be used to sort the recieved events fragments.

Consistency

Consistency properties in replicated event system on ad hoc network mostly focused on the event duplication. In wireless mobile environment in which nodes are suffering from power deprivation and bandwidth shortages, the duplicated events are costly and should be avoided. Duplicated events can bring up some problems for some application as well. Consider an application which is a counter, and will count up is it receives the event E_+ for instance and will count down if it receives E_-. The duplicated events may impair the counter functionality. In such situations it is so difficult to distinguish between the duplicated events unless different time stamp and serial numbers be utilized.

Chapar bears the duplicated messages when we have partial flooding in the broker network (figure 3.8). This partial flooding is utilized in propagating subscriptions and memorized events. However, in Chapar we predict some strategies to control and degrade the duplication messages in the network. In fact it is tried to limit the event duplications not only inside the broker network and but also in memorized event notifications.

For inside the broker network, nodes use the event serial numbers to discern the duplicated messages and drop them. For real-time notification of memorized events, the subscribers may receive multiple copies of an event if they are connected to multiple BNs (figure 3.8.a). However, for memorized notification each MET provides a “purging operation” to remove those absent subscribers from USNV who reattached and are not the BN neighbor (3.16). Without this purging mechanism, whenever an absent node (in time of publishing) attaches to a BN it receives a copy of the event (similar the example in redundancy section, node X may receive E_1^4 to E_1^{10} from $\underline{C}$ again).

For real-time published messages, we may not have any duplicated messages since multicast through a single tree is utilizing.

Completeness

In completeness, the developers are basically looking forward to higher reliability of the system. They want to make sure that the broker node replication not only increase the reliability of the system but also itself does not bring about new flaws to the system.

Completeness is one of the basic objectives followed to design Chapar. In fact the major goal of design was to orchestrate an overlay network as an event system on MANET as functional and complete as a basic event system on fixed network with determined three parties. Surveying different event system part in Chapar, the subscription is done through flooding the network so ideally all of the broker nodes will receive it, yet those nodes have not received the subscription will be notified during table integrity check and check-out process. Inheritance is also designed in the way that a new BN would receive all of information in FMT table. Similarly, memorized events and databases would not be missed in the network.

The real-time publishing is also utilizing multicast over OLSR to forward the events from publisher to subscriber. So as it is mentioned, it is tried to cover the completeness of this event system infrastructure as much as possible. Nonetheless, there are some exceptions that may come up due to node mobility in which the event system may not forward the event from publisher to subscriber. For instance, considering figure (3.7) if node 2 and 3 detach from the main network and connect together and create network $\overline{23}$, if node 2 publish an event, node 3 as its subscriber may not receive the event, unless it subscribes to the event again which is not really feasible in this case. In such a situation, it is recommended that if the event is important, it be published as the memorized event, so when the two nodes come back to the network, the memorized event stored in each of which will be injected to the network and subsequently node 3 will be notifies.

Table 3.1 summarizes the desirable properties expected from an event system using replication in Chapar.

3.4.5 Resource Awareness

In this section we look over different resource awareness parameters on ad hoc networks, to study how Chapar is adapted to different resource restriction on mobile environment. The resource restrictions are basically classified in yet not confined to node *memory limitations*, *energy deficiencies*, and *bandwidth shortages*. So it is expected that the applications working on the mobile nodes utilize memory in critically restricted way (especially when nodes are sensors), and take advantage of simple algorithms for processing to minimize the node power consumption, and finally generate necessary and efficient data packets for data dissemination to rest of the nodes.

However, in most of the applications, lack of network infrastructure enforces some extra processing and signaling to the network members. Besides, applications are recommended to be distributedly implemented on nodes to share the processing load on all of the nodes (especially if the network is formed by homogenous nodes), this requires some operations to orchestrate the peers and some network resources to be allocated for communication of different peers in the network. Some of network properties such as mobility and its subsequent consequences increase the complexity of the system and the software. Therefore, finding a balance in such harsh situation would be so challenging in designing and implementation of various services on MANETs [37].

In the following we provide an analysis over Chapar on different scopes which are more concerned in mobile networks.

Node Memory Allocation

Because of time decoupling in event system, the event broker needs to provide services not only to deliver real-time events to their subscribers but also to store some events in the preserved memory. Moreover, the subscription information needs to be retained in the system safely. In Chapar, the broker system which needs to keep these information in its memory is formed by some nodes which are replicated BNs. These BNs are generally OLSR MPRs, which supposedly are small [9] subset of the network members. Similarly, in Chapar to degrade the memory in use, we need to have fewer nodes involved in broker network, so the dense network are more embraced than sparse networks.

As it was mentioned, the subscriptions and memorized events are kept in FMT and MET tables. The tables are tried to be desgined with memory deprivation considerations. Using NV and hash-based filter structure are not only facilitates the fast and light filter matching process but also provides the node the ability to save the filters and subscriptions very efficiently.

Referring to section 3.2.4 the size of the FMT table will be calculates like

$$S = F \cdot (F_e + N) \;\; (bits) \tag{3.27}$$

where S denotes the size of the table in bits, F denoted the number of the filters and F_e is the size of the filter which is

$$F_e = ID_e \cdot e_{size} \tag{3.28}$$

where ID_e is the number of filter ID (or event ID) elements and e_{size} is the element size (2 bytes (=16 bits) is proposed).

[9]OLSR is recommended for dense networks to have a really optimized number of routing messages and MPRs in the network[3].

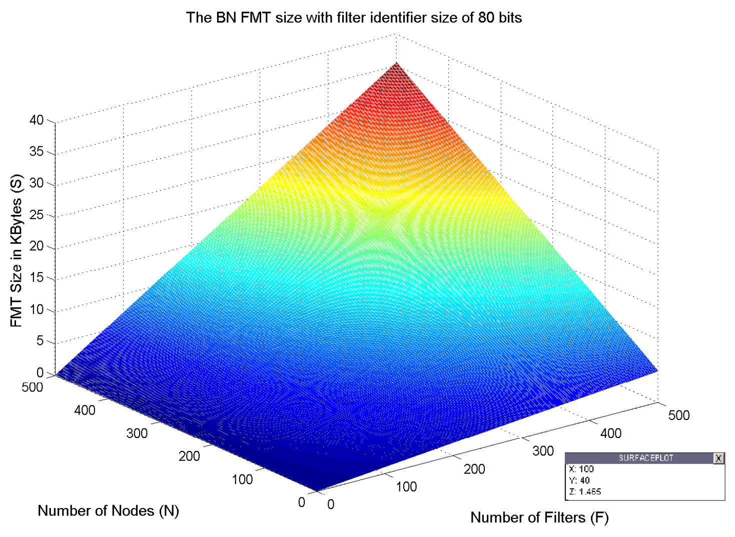


Figure 3.13: The FMT size (S) based on the number of nodes (N) and number of the defined filters (F)

As it can be concluded from (3.27) the size of FMT table is raising up linearly by respect to number of filters and coefficient of number of nodes along with the size of the filter identifier. As you may notice from figure 3.13 for instance, in a network composed of the 40 nodes in which there are 100 filters each of which has 5 elements, the FMT size will be up to 1500 Bytes ($\approx 1.5KB$) which is small enough based on our considerations.

The MET table would be considerably bigger than FMT. The MET does include the events, so the size of MET depends on the number events and the size of events which are stored in MET. Since the size of events is not determined and they are basically contingent upon the applications and services using event system, it is not really feasible to estimate the size of MET. However from the design of the MET this conclusion could be drown that the number of nodes has a very slight effect on the MET size. In fact, each expected node in the network is represented by a bit in each MET record, which induce weak MET size dependability to number of nodes.

There are also some other data containers in each BN nodes such as "duplication queue" in order to hold the received event serial numbers to tackle the duplication problem. Each subscriber also holds a map table to handle the received notified events and forward them to their corresponding thread which has subscribed to that specific events and now is waiting for its arrival. These tables and queues size are not really substantial to be taken into account (will be studied in details on subsection 4.2.4).

In the rest of the event system component such as real-time publishing system and the notification the mass memory allocation is not accomplished.

Conclusively, Chapar event system allocates a controlled and restrained amount of memory (considering the MET table size is controlled by middleware policies) which could be fitted in the critical resource situation we deal with in MANETs.

Node Processes Analysis

The CPU usage in each node is one of the parameters needed to be brought into account to design power-aware systems. The basic operations in real-time event notification are listed as filter matching for incoming events, and building a multicast tree for publishing the

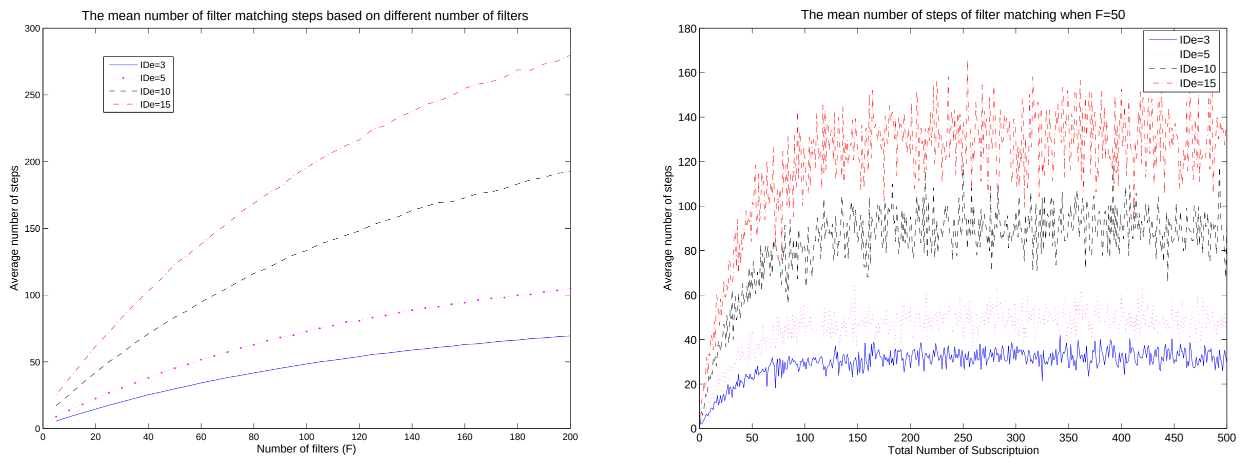


Figure 3.14: Expected number of steps for filter matching process with different ID_es

events. There are also some other operations to handle mobility in mobile environments. In Chapar, the filter matching algorithm is based on the basic binary operations with limited and controlled number of the steps. This process is not only triggered on all of the BN nodes when a subscription arrives but also executed on one of the BNs when an event is published. So the predominant process will be on the first part. Figure 3.14 illustrates the mean value for number of element checking (which is an & operation) with respect to the number of filters in FMT (F). In this figure we assume that the probability distribution of filter among all of subscriptions is normal with ($\sigma = 0.1F$). As you could notice, the expected number of steps is an almost linear graph with respect to F with a reasonable coefficient. This coefficient enlarges when the ID_e increases [10]. The figure shows that the number of the steps is reasonable enough based on our considerations for the mobile environment.

We take advantage of OLSR to maintain the multicast tree and handling mobility, however some extra operations are exploited to handle mobility completely. Using OLSR to construct a virtual multicast tree to route the published events to its subscriber(s) preserve substantial calculation load and extra signaling. As it was explained in publishing methodology, each BN as a vertex in multicast tree is receiving an event, read its destination NV, forward the event to its adjacent BNs with new destination NV. Besides, this multicast tree is built and valid for each packet, so we should not worry about node mobility, since the new multicast tree will be made for the node with new position.

The extra operations for mobility handling such as inheritance and table integrity check also do not involve any particular processes to violate the restricted conditions we have in mobile environment.

Node Signaling and Bandwidth Allocation

The third parameter we have specified for the resource-awareness is the volume of the traffic generated by the overlay network. On the OLSR-based networks, the periodic routing messages are engaging a considerable portion of the total traffic exchanged through the network. The routing messages are mainly including the HELLO messages and TC messages. The quantity of the traffic in given time period are changed according to the number of the messages and size of the messages. The number of the HELLO messages is determined by `HELLO_INTERVAL` parameter in OLSR. In fast networks (high degree of mobility), to provide nodes connectivity the less interval value is required which raises the total traffic allocation in specific chunk of time. A HELLO sent periodically to all of node's neighbors contains information about the node's neighbors, the nodes it has chosen as MPRs, and a list of neighbors whose bidirectional links have not yet been confirmed (symmetric links)[3]. So HELLO messages size may vary based on the number of the neighbors a node has. The number of neighbors per node is defined as the network density in some literatures [36] which has a considerable impact on the size of HELLO messages on the network.

On the other hand, TC messages are also sent periodically (based on `TC_INTERVAL`) but not from all of mobile nodes. They are just sent and forwarded by network MPRs So the number of MPRs and the size of MPR network is important. However the number of MPRs is tightly related to network density. If the network is dense, nodes are communi-

[10]The working point for Transhumance would be the first graph with ($ID_e = 5$) and less than 50 filters.

Simulation Parameters	
Simulation Specifications	Tool: NS-2.28+UM-OLSR [39] Duration: 3600s Simulation time slot (T): 30s
Network Specifications	Size: variable Number of nodes: 40 Network routing protocol: OLSR (RFC3626) TC_interval=5s, HELLO_interval=2s
Node Specifications	Radio-propagation model: Two Ray Ground Antenna: OmniDirectional MAC type: 802.11 Queue type and length: Drop-tail/Priority queue 50 Effective coverage range (R): 250 m
Mobility Specifications	Mobility Model: Random Speed: 1 m/s
The Traffic Spcifications	Susbscription: Almost constant ($65B \leq size \leq 70B$) PDF: Uniform Susbscrption rate: 6 subscription/minute (3 subs/T) Memorized Messages: minimum: 94B, average: 362B, PDF: log-normal (Figure) Memorized Events Rate: 2 event/minute (1 events/T)

Table 3.2: The simulation specifications

cating directly without any intermediate nodes. This reduces the number of MPRs and subsequently the number of TC messages. In fact, the size of TC messages is based on the MPRs' Selector set (MS) size. Yet if we consider the network generally the overall size of TC messages is constant if we do not experience network partitioning. Thus, we may conclude that the number of TC messages is predominantly defined by the number of MPRs and number of TC messages. However in HELLO messages the total number of HELLO messages is almost constant but their size may vary by number of neighbors each node has. So basically both of the parameters are variable upon the density of the network.

To have a discussion on the generated traffic in event system, we categorize traffic in three types: The real-time published events, the subscription messages, and the memorized events. We also have some other source of traffic which is provided to compensate the incidents may be occur by mobility. They are namely inheritance traffic and table integrity check traffic (check-out traffic is included in this part).

The real-time events are multicasted in the network so they impose an insignificant traffic on the network. Whereas, the subscription and memorized messages are flooded in the MPR network so their traffic is considerable. The rest of the event system traffic will be analyzed at the end of this part.

To compare the event system traffic and the OLSR routing messages, we provide a simulation on a network composing of 40 nodes with random mobility in different areas with variable sizes. This simulation is done by NS2 [38] with UM-OLSR package [39]. The simulation specifications are shown in Table 3.2. In this simulation, we chiefly focus on the subscriptions and memorized events traffic since they are significantly important to evaluate the traffic of the system. We also defined a time slot (T) of 30 seconds in which, in average, three subscription messages as well as one memorized event is generated. The whole simulation time is 3600s which is 120 time slots. The subscription messages has almost constant size of ($65B < S_{subscription} < 70B$) whereas the memorized events has random size with log-normal PDF with ($min(S_{memorized}) = 94B, mean(S_{memorized}) \simeq 362B$). The nodes produced traffic for subscription and memorized messages is shown in the figure 3.15. As it is mentioned the simulation is based on different densities. The density in this literature is expressed as a *quality* parameter, since the simulation is done based on different diameters of a square area ($D = N\sqrt{2}$ for a NxN square) to show how larger areas may have influence on the OLSR messages as well as event system traffic. The different levels of density are classified as:

- Full-mesh: In which all of the nodes are located in each other's effective coverage range. They are connected to each other with no intermediate node and in event system they are treated as full-mesh networks indicated in section 3.3.4.
- Dense networks: The nodes are tightly connected and few intermediate nodes are chosen to route the packets among nodes. These networks do not suffer partitioning and its consequences, so they are the favorable networks in which nodes and application are working almost in the ideal performance.
- Sparse networks: The nodes are loosely connected, and network might counter partitioning at any moment due to the random mobility. It should be tried to avoid such situation, because network segmentation would be damaging for network applications and network performance.

As it was described, for areas with diameters ($D \leq R$)[11] the network will be indeed full-mesh, and network will not experience multihop communication, and subsequently

[11] R is the node effective coverage range

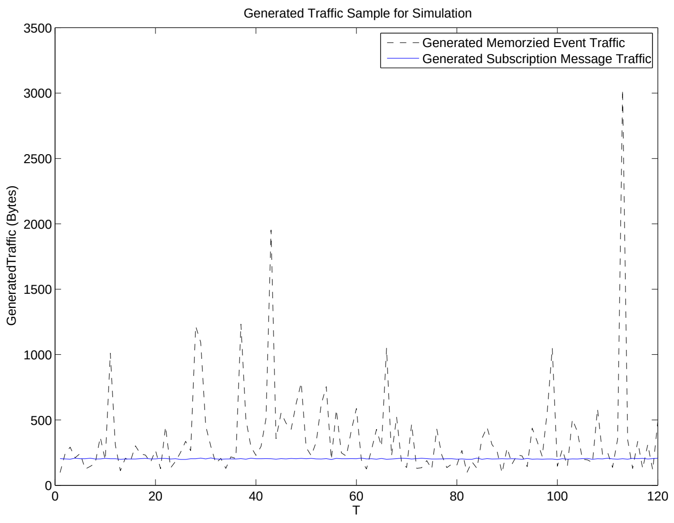


Figure 3.15: The initial generated traffic

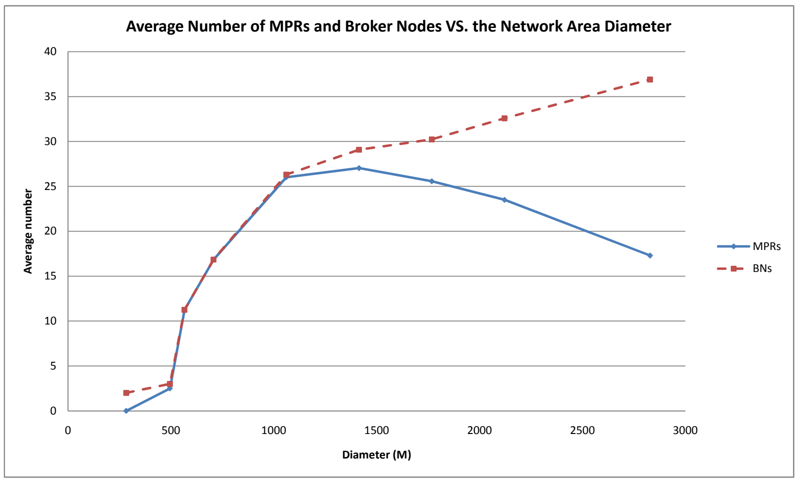


Figure 3.16: The number of MPRs and Brokers based on the network working area

network does not belong any MPR. The simulation shows that the network still remain in the full-mesh node even if the D raises to 353m still the average number of the MPRs tends to zero which proves the fact that the network is full mesh. In such situation, nodes elect two of them as the BN to be in charge for the event system tasks. As it could be noticed in figure 3.16, in full mesh networks when $D \lessapprox 450$, $M \rightarrow 0$, but $N_{BNs} \geq 2$. However when $D \gtrapprox 450$ meters network has more than two MPRs and does not need to perform election for BNs.

Figure 3.16 shows when the D raises and subsequently the density decreases, the number of MPRs rises to almost 67% of the number of the nodes. However, after D goes up to 1400 the average number of MPRs shrinks . This is the beginning of the criteria so called sparse network. The reason of this reduction does come back to the network partitioning. When the area size grows up, the network will be segmented into smaller subnetworks completely secluded from each other. Each subnetwork, however, has their own MPRs, whose total number is less then the previous state. With continuation of increasing the size of the area, we will face some individual nodes disconnected from the network as well as many small networks (composing less than 5 nodes). We call this state as highlt sparse network which happens when $D \gtrapprox 1800$ meters. In such a decomposed network, we may face many small subnetworks together with some single nodes, double nodes and small full-mesh networks, this increases the number of BNs in the network which is clearly illustrated in the figure 3.16. The complete simulation results are demonstrated in figure 3.17. Comparing the OLSR message and event system messages, the event system messages are substantially less then the OLSR messages even with very pessimistic initial traffic we consider for simulations and it was shown on figure 3.15. Figure 3.17 points out that in the full-mesh networks and very dense networks the HELLO messages are dominating the total bandwidth of the system where the TC messages and event system messages are negligible. But as the diameter grows, the TC messages increases and the HELLO

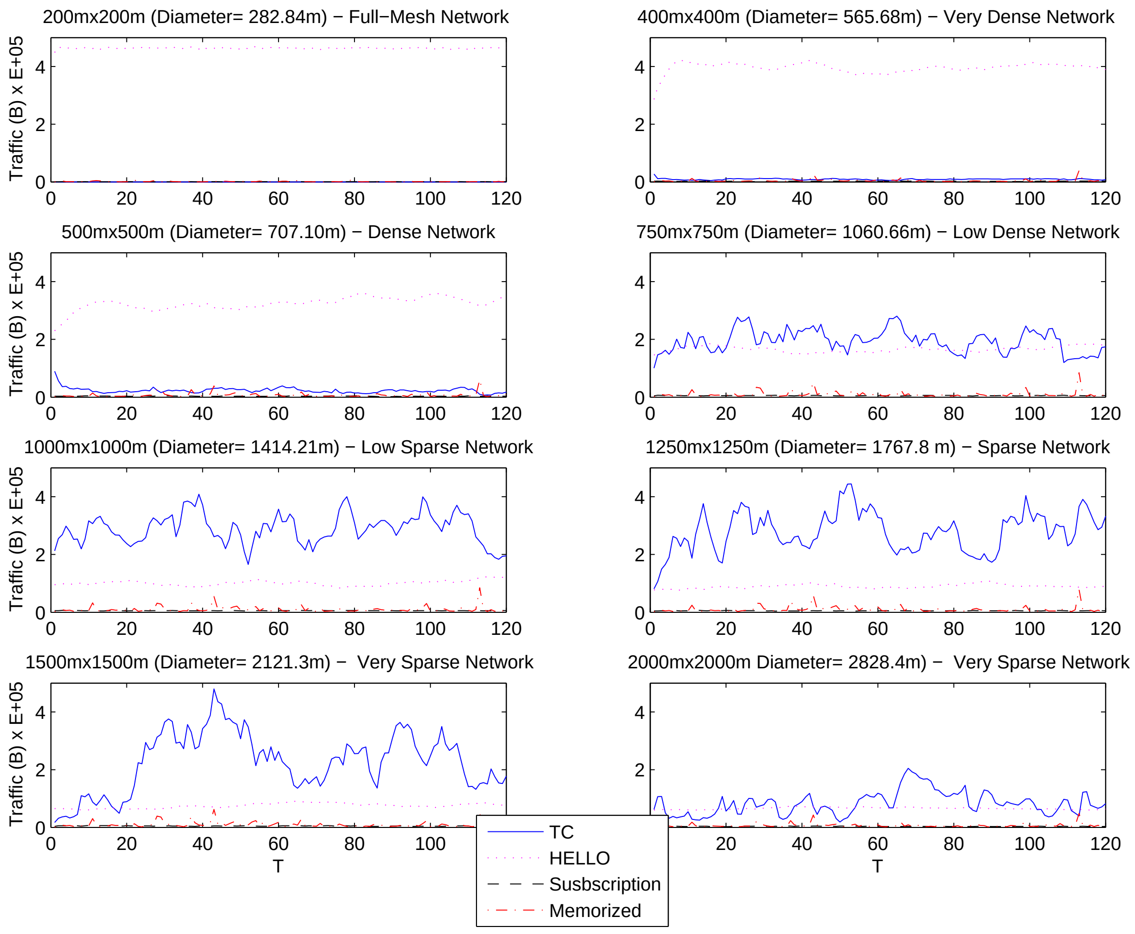


Figure 3.17: The OLSR routing and event system total traffic in 120 time slots (3600s) with different densities

messages traffic degrades. This is not only because of the size of HELLO messages which shrinks due to reduction of node neighbors, but also because of the number of TC messages raises due to growth of the MPRs. And finally in highly sparse networks we may notice the total OLSR traffic falls because of the decomposed network.

To study the event system traffic, figure 3.18 which shows the total traffic in logarithmic scale. This would show better the event system's element traffic fluctuations. The subscription traffic as it was expected is almost constant. Since the sizes of subscription messages does not differ considerably and they sent in a constant rate. However the size of the events may differ in memorized events. The spikes in figure 3.18 indicates the impact of the large memorized events on total bandwidth. Although the OLSR messages traffic is extremely dependent upon network density, the event system traffic changes are more reasonable. The subscription traffic for dense networks is almost 5 times more then the full-mesh networks and this number goes to 10 times more for sparse networks. Similar variation does almost happen for memorized events. Figure 3.19 shows the average traffic of event system and OLSR routing daemon on different densities. Although the total traffic for event system is rising sharply when network density degrades, it is still almost 200 times in full-mesh networks, 25 times in dense networks and 10 times in sparse networks less than the total amount of OLSR messages traffic.

And finally, studying traffic for inheritance and table integrity check, inheritance and check-out traffic according to level of mobility could be controlled by DTT. If DTT is configured based on the level of mobility we would have minimal number of inheritance and check our request. Another parameter that could affect the volume of the traffic of such processes is the size of FMT and MET. Indeed, having large FMT and MET make the inheritance and check-out process heavy in traffic point of view. The table integrity check process is periodically executed and its traffic would be minimized if the network is stable and network experience a few network partitioning over time.

In conclusion, the simulations shows that the density of the network could have may play a considerable role on the generated traffic not only on event system but also in OLSR routing messages. It also illustrates that even though event system uses the flooding in MPR network for subscription and memorized events, its total generated bandwidth is significantly less than the OLSR network.

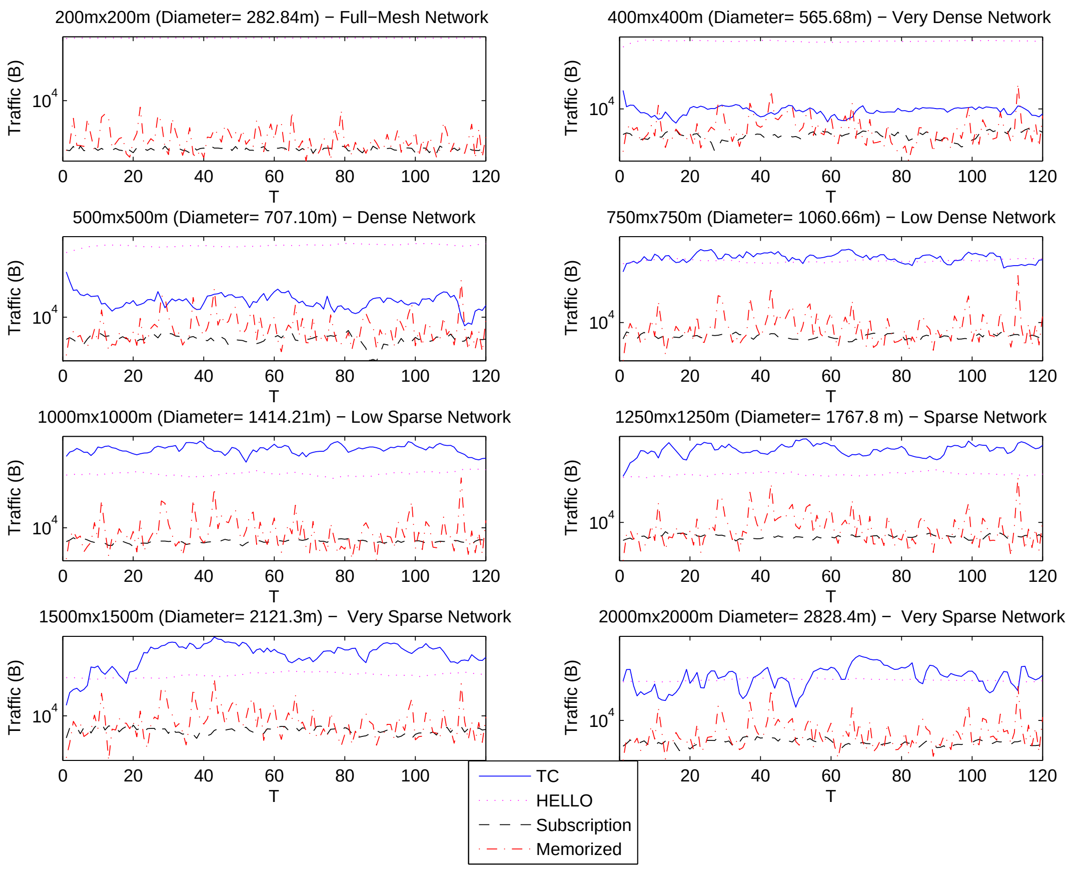


Figure 3.18: The OLSR routing and event system total traffic in logarithmic scale in 120 time slots (3600s) with different densities

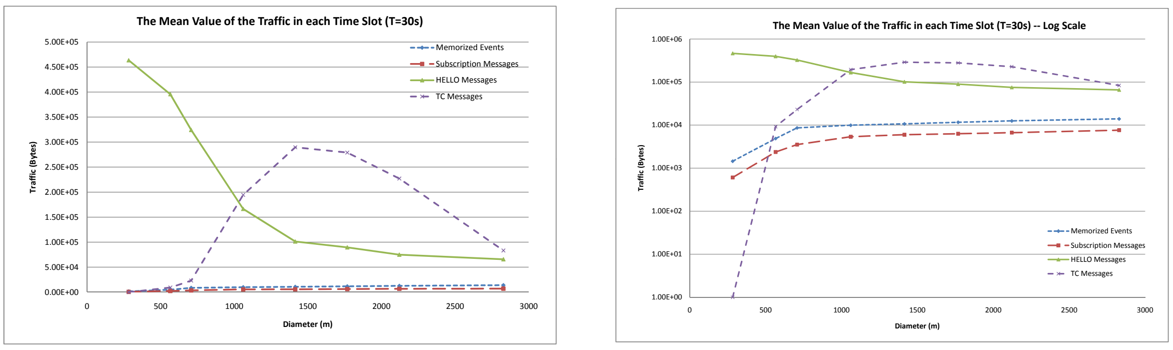


Figure 3.19: Comparison between the average value of the different traffic types based on the different network area diameters.

3.4.6 Scalability

The last feature we investigate on Chapar is the Scalability. Since Chapar is an overlay network based on OLSR, its scalability is tightly related to the scalability of the OLSR. Although OLSR's performance is proved to be satisfactory on small- and medium-scale networks, feasibility of using OLSR on large-scale networks is debatable. This is basically because of the size of TC and HELLO messages needed to be dispatched in the network. However there are many contributions to improve the scalability of OLSR [40, 41, 42].

As a metter of fact, Chapar tries to subdue the scalability impacts on memory by using NVs. The main purpose of using NV is avoiding the utilization of complete node address as well as accelerating the event system operations in BNs. For instance, Equation 3.27 and figure 3.13 shows the influence of the number of the nodes on FMT size. In fact, (3.27) demonstrate the fact that that the size of FMT (similarly with MET) has linear relation with the number of nodes with fixed coefficient of number of filters ($S = F \cdot N + F \cdot F_e$). Conclusively, the scalability does not have a considerable impact on the node memory.

Using NV, the esclation of number of nodes imposes a trivial effect (in order of bits) on the size of the events published in the network. Thus, in practice, the event system functionalities would not be affected by number of nodes and, of course, the allocated bandwidth per event would differ insignificantly.

The publishing mechanism is also independent from number of nodes. In spite of some multicast mechanisms proposed for event systems in which the multicast tree is defined in publishing time [23], the multicast tree utilized for publishing in Chapar is a virtual tree and it is constructed by the nodes in each stage of event forwarding. So it is repetitive process started from the tree root(first BN) and terminated when the event is notified. Thus, no matter how many nodes are forming the network this repetitive process is the same, although the number of repetition would be increased (which may cause some delays).

In conclusion, we may deduce that Chapar by itself is scalable and could be used in large-scale networks thanks to the NV. However since it has a cross-layer architecture over OLSR, its scalability is confined by OLSR scalability issues.

3.4.7 Bloom Filter Impacts on Chapar

As it was discussed in subsection (3.2.3), the bloom filter could be an alternative for NVs if the network nodes' addresses are not organized enough to be mapped to a bit array. However, using BFs requires longer bit array (8 times more than NV) to avoid the false positive error(reduce the probability to $< 2\%$). It also imposes considerable processing overload on system, since the BF is useful for query processes, whereas in Chapar NV is used as node address container. In other words, from BF itself it would not be possible to derive some information, and to list the BF content, all of the possible records should be queried. While each NV represents a set of the nodes and to query from a NV, a simple (&) operation is performed between the container and the NV of the query elements.

Another stated problem with Bloom filter is the inability of the removing elements form a BF, unless counting bloom filter is utilized. The counting bloom filter has larger size than bloom filter and its not fixed for all counting BFs which induces some complexity in packets including counting BFs in their header.

Therefore, an attentive utilization of bloom filters are recommended just for large scale networks with unorganized address pattern to preserve network from using the real node addresses (4 byte for each IPv4 address [43] and 16 byte for each IPv6 addresses [44]).

Conclusion

In this report we introduce Chapar as a full featured event system which provides point-to-point, point-to-multipoint, and publish/subscribe system (or combination of them) following an asynchronous paradigm. Chapar publish/subscribe system supports all of the desirable decouplings in pub/sub systems and provides data dissemination reliability and consistency in harsh mobile environment. On the other hand, Chapar is a distributed event system with replicated data containers. Using this architecture we tried to design Chapar to not to be dependent to any specific node and be a ubiquitous event system capable of being implemented on homogenous networks.

This method uses OLSR as an underlay network and exploits the node routing table to build its multicast tree through which it notifies the subscribers by published events. OLSR proactive routing protocol provides Chapar with updated information about the available nodes in the network and the direction to reach them. This information is utilized by Chapar to build its multicast tree for each published event individually in real-time, which leads to supporting node mobility by event system. It also uses OLSR multi-point relay concept not only to reduce the number of brokers from all of the nodes to a small group of nodes, but also to foster the situation for the nodes to be self-assigned as the broker node.

This event system utilized a data structure called node vector (NV) which is a bit array and is representing the list of nodes. Using NV, provide us some facilities to reduce the multicast tree calculations overhead by using simple logical & and | operations. Moreover, NV offers a form of abstraction in list of nodes which is considerably helpful in different preserved tables such subscription tables and memorized event table. This abstraction is also useful to reduce the size of the dispatched events which is convenient in resource deprivation situation in mobile environment.

Using different techniques like inheritance and check-out process helps to have a self-configuring event system independent from any centralized management. Moreover, extra processes like table integrity check helps to synchronize the broker table to provide a reliable service and encounter the complications caused by network partitioning. The simulations and method analysis on different concerns on mobile environment demonstrates that the proposed system could be implemented conveniently on mobile networks working by OLSR routing protocol.

This project is done as the event management module on communication layer of Transhumance Middleware Project and implemented by C++ programming language on UNIX platform.

Bibliography

[1] P. Eugster , P. Felber , R. Guerraoui , and A. Kermarrec, “The many faces of publish/subscribe,” ACM Computing Surveys (CSUR), v.35 n.2, p.114-131, Jun 2003.

[2] P. Eugster, “Type-based publish/subscribe: Concepts and experiences,” ACM Transactions on Programming Languages and Systems (TOPLAS), v.29 n.6, Jan. 2007.

[3] T. Clausen and P. Jacquet, “Optimized Link State Routing Protocol,” RFC 3626, Internet Engineering Task Force, Oct. 2003.

[4] D. Powell, “Group communication,” Communications of the ACM, v.39 n.4, p.50-53, Apr. 1996.

[5] K. P. Birman, “The process group approach to reliable distributed computing,” Communications of the ACM, v.36 n.12, p.37-53, Dec. 1993

[6] R. Lewis, Advanced Messaging Applications with MSMQ and MQSeries. QUE 1999.

[7] Oracle9i Application Developer’s Guide– Advanced Queuing. Oracle, Redwood Shores, CA. 2002.

[8] M. Altinel , and M. J. Franklin, “Efficient Filtering of XML Documents for Selective Dissemination of Information,” Proceedings of the 26th International Conference on Very Large Data Bases, p.53-64, Sept. 2000.

[9] OMG. CORBA Event Service Specification. Object Management Group, Needham, MA, Mar. 1995.

[10] OMG. CORBA Notification Service Specification. Object Management Group, Needham, MA. Aug. 2002.

[11] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, “ Java Message Service,” Sun Microsystems Inc., Santa Clara, CA 2002.

[12] G. Cugola , E. Di Nitto , and A. Fuggetta, “The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS,” IEEE Transactions on Software Engineering, v.27 n.9, p.827-850, Sept. 2001

[13] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, “Design and Evaluation of a Wide-Area Event Notification Service”. ACM Transactions on Computer Systems, 19(3):332-383, Aug. 2001.

[14] B. Segall, and D. Arnold, “Elvin has left the building: A publish/subscribe notification service with quenching,” In Proceedings of the Australian UNIX and Open Systems User Group Conference, 1997.

[15] G. Banavar , T. D. Chandra , R. E. Strom , and D. C. Sturman, “A Case for Message Oriented Middleware,” Proceedings of the 13th International Symposium on Distributed Computing, p.1-18, Sept. 1999.

[16] T. W. Yan , and H. Garcia-Molina, “The SIFT information dissemination system,” ACM Transactions on Database Systems (TODS), v.24 n.4, p.529-565, Dec. 1999.

[17] Y. Huang , and H. Garcia-Molina, “Publish/Subscribe in a mobile enviroment,” Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access, p.27-34, Santa Barbara, CA, May 2001.

[18] A. Carzaniga, D. Rosenblum, and A. L. Wolf, “Design and evaluation of a wide-area event notification service,” ACM Transactions on Computer Systems (TOCS), v.19 n.3, p.332-383, Aug. 2001

[19] M. Caporuscio, P. Inverardi, and P. Pelliccione, “Formal Analysis of Clients Mobility in the Siena Publish/Subscribe Middleware,” Technical Report, Department of Computer Science, University of Colorado, Oct. 2002.

[20] M. Hauswirth , and M. Jazayeri, “A component and communication model for push systems,” Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, p.20-38, Toulouse, France, Sept. 1999.

[21] The Community OpenORB Project, available at: http://openorb.sourceforge.net/

[22] Free High Performance CORBA Notification Service from AT&T Laboratories, available at: http://omninotify.sourceforge.net/

[23] G. Cugola, A.L. Murphy, and G.P. Picco, “Content-based Publish-Subscribe in a Mobile Environment,” In Mobile Middleware, Ed. P. Bellavista and A. Corradi, pp. 257-285, Auerbach Publications, 2006.

[24] G. Cugola, and H.-A. Jacobsen, “Using publish/subscribe middleware for mobile systems,” In ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6 , num. 4, Oct. 2002.

[25] E. Anceaume, A. K. Datta, M. Gradinariu, and G. Simon, “Publish/subscribe scheme for mobile networks, ” Proceedings of the second ACM international workshop on Principles of mobile computin, Toulouse, France, Oct. 2002.

[26] Y. Huang , and H. Garcia-Molina, “Replicated condition monitoring,” Proceedings of the twentieth annual ACM symposium on Principles of distributed computing, p.229-237, Newport, RI, Aug. 2001.

[27] R. Meier , and V. Cahill, "STEAM: Event-Based Middleware for Wireless Ad Hoc Network," Proceedings of the 22nd International Conference on Distributed Computing Systems, p.639-644, July, 2002.

[28] M. Cilia , L. Fiege , C. Haul , A. Zeidler , A. P. Buchmann, "Looking into the past: enhancing mobile publish/subscribe middleware," Proceedings of the 2nd international workshop on Distributed event-based systems, San Diego, CA, June, 2003.

[29] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," In Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS), 1998.

[30] Peter R. Pietzuch , Jean Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," Proceedings of the 22nd International Conference on Distributed Computing Systems, p.611-618, July, 2002.

[31] Transhumance Official Site. Availbale on: http://www.transhumance.info/

[32] G. Paroux, L. Martin, J. Nowalczyk and I. Demeure, "Transhumance: A power sensitive middleware for data sharing on mobile ad hoc networks," 7th international Workshop on Applications and Services in Wireless Networks (ASWN), Santander, Spain, May 2007.

[33] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, v.13 n.7, p.422-426, July 1970.

[34] L. Fan , P. Cao , J. Almeida , A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, p.254-265, Vancouver, British Columbia, Canada, Sept. 1998.

[35] C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers," In Proc. of ACM SIGCOMM'94, pp. 234-244, London, UK, Aug-Sept 1994.

[36] D. Q. Nguyen and P. Minet, "Analysis of Multipoint Relays Selection in the OLSR Routing Protocol with and without QoS Support," INRIA Research Report (RR-6067), 2006.

[37] W. Kiess , M. Mauve, "A survey on real-world implementations of mobile ad-hoc networks," Ad Hoc Networks, v.5 n.3, p.324-339, Apr. 2007.

[38] NS2, The Network Simulator, available on: http://www.isi.edu/nsnam/ns/

[39] F. J. Ros, Universidad de Murcia OLSR impelmentation for NS2, Available on: masimum.dif.um.es/um-olsr/html/

[40] D. Nguyen P. Minet, "Scalability of the OLSR Protocol with the Fish Eye Extension," Sixth International Conference on Networking (ICN'07) p. 88, Sainte-Luce, Martinique, Apr. 2007.

[41] Y. Ge , L. Lamont, L. Villasenor, “ Hierarchical OLSR - a scalable proactive routing protocol for heterogeneous ad hoc networks” IEEE International Conference on-Wireless And Mobile Computing, Networking And Communications, (WiMob’2005), Montreal, Canada, Apr. 2005.

[42] Y. Ge, L. Lamont, and L. Villasenor, “Improving Scalability of Heterogeneous Wireless Networks with Hierarchical OLSR,” in The OLSR Interop and Workshop, Aug. 2004.

[43] – “Internet Protocol ” RFC 760, Internet Engineering Task Force, Sept. 1981.

[44] S. Deering, R. Hinden, “Internet Protocol Version 6 (IPv6) ” RFC 2460, Internet Engineering Task Force, Dec. 2460.

Appendix A.

To calculate the bloom filter false positive error, we assume that the hash functions h_i are completely fair and each array position in the bloom filter is selected with equal probability. The probability of each array position to not to be set to one in a m bit bloom filter would be:

$$1 - \frac{1}{m}$$

So for all of k hash function this probability turns into

$$\left(1 - \frac{1}{m}\right)^k$$

And with n elements it subsequently is

$$\left(1 - \frac{1}{m}\right)^{kn}$$

This is the probability of one bit to not to be set to 1 by any h_i. So the probability of that to be one will be

$$1 - \left(1 - \frac{1}{m}\right)^{kn}$$

Now we are trying to query the bloom filter for an element which is not in the set. For the false positive error all of the k hash function results should be matched by 1s in the bloom filter. So if we assume that the probability of each node to be set to 1 is as above, now for k array position we will have:

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

Which could be estimated by the erroneously claim to

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$$

Assuming that by given m and n the optimal k we be calculated as follows:

$$f = exp\left(k \cdot ln\left(1 - e^{-kn/m}\right)\right)$$

$$g = k \cdot ln\left(1 - e^{-kn/m}\right)$$

So to calculate the optimal k, we need to calculate the $\frac{dg}{dk}$:

$$\frac{dg}{dk} = ln\left(1 - e^{-kn/m}\right) + \frac{kn}{m}\frac{e^{-kn/m}}{1 - e^{-kn/m}}$$

$$\frac{dg}{dk} = 0 \Longrightarrow k_{min} = (ln\ 2)\left(\frac{m}{n}\right)$$

And ultimately we have:

$$f(k_{min}) = \left(\frac{1}{2}\right)^{k} = (0.6185)^{m/n}$$

Appendix B.

Matching process false positive error probability Assume A and B are two n bit numbers and we want to calculate the probability of

$$A\&B = A$$

when $A \neq B$ (excluding the 0 identity property ($A = 0$)).

So to calculate this we focus on the fact that the identity property of each 0 in the A induces the false positive error. So the number of 0s will be the sumation of the all permutations of 0s in the n bits excluding the n 0s ($A = 0$).

$$\sum_{i=1}^{n-1} \binom{n}{i}$$

Each of these permutations of 0s could cause the errors of 2 with power of number of 0s. However one of which is for when $A = B$, so it will be excluded. $(2^i - 1)$.

If we consider A and B is chosen with uniform PDF (randomely), the probability of the false positive error will be calculated as

$$P_{false\ positive} = \frac{\sum_{i=1}^{n-1} \binom{n}{i} \cdot (2^i - 1)}{2^{2n}}$$

For instance: for $n = 8 \rightarrow P = 0.0923$, for $n = 16 \rightarrow P = 0.0100$, for $n = 32 \rightarrow P = 0.0001$

Dépôt légal : 2008 – 3ème trimestre
Imprimé à l'Ecole Nationale Supérieure des Télécommunications – Paris
ISSN 0751-1345 ENST D (Paris) (France 1983-9999)

TELECOM ParisTech

Institut TELECOM - membre de ParisTech

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - www.telecom-paristech.frfr

Département INFRES