IRACST – International Journal of Computer Networks and Wireless Communications (IJCNWC), ISSN: 2250-3501
Vol.2, No5, October 2012

AMFVoNet: An Active Messaging Framework for Volatile Networking

Jaime Galán-Jiménez, Alfonso Gazo-Cervero
Computer Science and Communications Engineering Dept.
University of Extremadura, Cáceres, Spain
{jaime,agazo}@unex.es

***Abstract*—Most of current mobile applications need to have mobility capabilities and are often required to work under different conditions, even in situations with continuous disconnections. Intermittently Connected Mobile Networks (ICMN) encompass environments with intermittent connectivity and long disconnection intervals. In this paper, we focus on defining a generic framework in order to implement existing routing protocols for ICMN, as well as on providing a tool to create a networking technology for volatile environments in which the message recovers some degree of choice to reach its destination.**

Keywords-volatile networking, forwarding code, ICMN, protocols, AFN

I. INTRODUCTION

Nowadays, the use of mobile Internet entails an exponential growth of the number of existing mobile devices. This increase is much rapid than any previous technology. Mobile devices penetration has grown from a global 5% in 1998 to a 55% in 2008, with more than 10 billion of mobile devices with Internet connection in 2010. It is estimated that the penetration rate will reach 96% in 2018 [1]. This increase of mobile devices is linked to the existence of a large number of applications which require to be continuously working under all conditions. However, complex features such as mobility can involve in some cases the existence of frequent disconnections.

Intermittently Connected Mobile Networks (ICMN) represent an example of this kind of situation. Connectivity is intermittent and nodes get frequent disconnections. Because of this, long disconnection intervals are assumed. As a side effect, a complete path from origin to destination might not always be established. Over time, different links come up and down due to node mobility. If the sequence of connectivity graphs over a time interval is overlapped, then an end-to-end path might exist. As a result, wireless connectivity is volatile and usually intermittent, as nodes move in and out of range from access points or from each other, and as signal quality fluctuates. This implies that a message could be sent over an existing link, get buffered at the next hop until the next link in the path comes up, and so on, until it reaches its destination. In these environments, applications which require available end-to-end paths have to deal with issues when trying to send messages from an origin to a destination when there is not an end-to-end path available. Therefore, traditional routing protocols tend to show bad performances in volatile connectivity environments. In fact, those protocols were designed for networks where most of the time there is an end-to-end path available between an origin and a destination. However, researchers have tried to alleviate this problem. Actually, a number of specific routing protocols have been proposed for networks that do not have stable end-to-end connectivity. Using these protocols, nodes can store messages for a period of time, until new forwarding opportunities can be taken towards the destination.

Routing protocols for ICMN are divided into two groups: deterministic and stochastic. If all the future topology of the network (as a time-evolving graph) is deterministic and known, or at least predictable, the transmission (when and where to forward packets) can be scheduled ahead of time so that some optimal objective can be achieved [2,3]. If the time-evolving topology is stochastic, routing protocols move the message closer to the destination one hop at a time. In this case, the nodes may know nothing about the network state and randomly forward packets to their neighbors. Protocols included in this category are globally known as epidemic [4-6]. However, if a node can estimate the forwarding probability of its neighbors, a better forwarding decision could be made. These routing protocols are based on history or estimations [7-9]. Moreover, if the mobility patterns can be used in the forwarding probability estimation, an even better decision may be made. Protocols in this category are referred to as model-based forwarding paradigm. Finally, network efficiency can be improved using a group of stochastic protocols which control the movements of certain nodes [10,11].

The main objective of this work focuses on studying the possibility of creating a networking technology for volatile network environments. Messages can participate in forwarding decisions based on different conditions they identify as they progress. For that, we have characterized a set of requirements for each routing protocol and we have made a proposal of a generic framework to implement each of those protocols. In addition, we have implemented one of the most relevant routing protocols specifically designed for these networks (Spray & Wait [5,6]). We have done so using our proposed generic framework so that our proposal can be a basis for researchers that work in ICMN routing topic in terms of implementing and validating their routing protocols. Using our generic framework, we expect to ease the creation of new potential protocols as well as the improvement of the existing ones.

The rest of the paper is organized as follows. Section II presents messaging framework features. Section III depicts the framework architecture, which is used in Section IV as the basis to implement one of the most relevant routing protocols in ICMN. Finally, some conclusions are drawn in Section V.

II. GENERIC MESSAGING FRAMEWORK FOR ICMN

All research to-date has focused on providing intelligent forwarding capabilities. Nodes can serve as a relay by forwarding messages across the network. In this way, messages traverse the network by being relayed from one node to another, until it reaches its destination. These nodes are considered as "custodians" because they typically use some form of store-carry-and-forward technique in order to approach messages to their destination. In fact, this seems a logical approach as the message source may be far away from places where particular forwarding decisions need to be made based on certain conditions.

Routing protocols proposed by researchers use specific strategies which depend on network design decisions. These decisions are made in a previous stage and cannot be modified to adapt to volatile environments features. Thus, we propose a communications framework whereby the source regains some degree of choice whilst acknowledging that it will be too remote from dislocations in the communication infrastructure to retain direct control of the message forwarding process. We consider messages themselves to be proxies for the source, i.e. they store data and information of forwarding preferences through the *forwarding code.*

The forwarding code has the ability of making routing decisions. It decides the next node towards the message must be forwarded. Moreover, it chooses if the message must be directly forwarded or stored inside the current node until a new forwarding opportunity. Thus, routing decisions are taken by the instructions specified in the forwarding code. These instructions use a set of primitives defined by the framework, which are intended to be able to perform basic routing operations.

Moreover, this code includes the number of copies of the message or some stateful information for the process of making routing decisions if necessary, and if this information can be updated in each node belonging to the path. For this purpose, full routing mechanism is programmed in the forwarding code which is included inside the message. When a source has a message to send, it decides the forwarding strategy that the message will be programmed with. This "program" provides forwarding decisions to be taken by the message during its travel towards the destination node. All the intermediate nodes can execute the forwarding code included in the message and forward it towards a next node considered as closer to the destination.

The forwarding code also includes the storage approach to be applied when there is a lack of forwarding opportunities. If a next hop is not available, the node will need to buffer the data until a new forwarding opportunity. Thus, if a node sends a message with a forwarding code within both forwarding mechanism and information about node storage management are included, that node could manage its storage area in a more efficient way (similarly to Differentiated Services [12]). In this way, the node holds its messages ordered by priority. This priority is provided by the users with the aim of managing their messages in the next time slot. Consequently, forwarding code will have enough information to permit the message to function as a learning device, akin to a mobile agent. In this manner, only the user (application) has the ability to decide the routing protocol to be used for its messages. Network can provide some stateful information, however it has not the ability to change the forwarding protocol for a certain message once this one has been sent from source. Next, we specify desirable features for the proposed framework:

1) Minimum size of the forwarding code in order to minimize message size and overhead.
2) Minimum size of the set of primitives. These primitives are available to be invoked in the forwarding code. With this minimal set of primitives we can assure that it is possible to program any routing protocol for ICMN.
3) Generality. Researchers can program each of the routing protocols proposed for ICMN.
4) Performance. The use of the framework must not mean a decrease in the network performance.

III. GENERIC FRAMEWORK ARCHITECTURE

In our generic framework we assume the presence of two key components:

1) Messages. They transport the forwarding code, the operative data required from this code and the payload.
2) Active Forwarding Nodes (AFN). They have the ability to execute the forwarding code of the messages they encounter. They provide an operating system and an execution environment.

Routing mechanism is fully implemented in the message, i.e. the source node adds this code into the message which is able to follow particular forwarding strategies towards its destination.

AFNs can execute the forwarding code and send the message towards the next node based on the logic of the code. Therefore, when a source has a message to send, it decides the forwarding strategy that a given message will be programmed with. This strategy can be a fixed set of rules or an algorithmic process that is configured with initial conditions. We do not attempt to specify the nature of these rules nor the algorithm; we simply require that they must adhere to a set of constraints. For example, the forwarding code could be expressed conform to some software interface standards for the exchange of data in and out from the process.

We have already mentioned that our focus is on ICMN where disruption is considered to be long lasting and unpredictable to a greater or lesser degree depending on the particular scenario. We therefore assume that there is no previously known end-to-end path and the transient natures of localized communication regions make it impractical to disseminate this reachability information over a large area. We

also anticipate the presence of one or more distinct communication technologies that typically employ incompatible addressing schemes, forwarding unit structures and so forth. Into this disparate and fragmented environment we rely on the presence of a number of AFNs. These nodes can be mobile or fixed nodes. They provide a means for processing the forwarding code as well as possess communication gateway functionality and may be linked to different technologies, i.e., several AFNs with different underlying technologies (Wi-Fi, Bluetooth, Wibree, ZigBee, etc.) may be present inside the same network.

Fig. 1 depicts the proposed architecture with the two key components: AFNs and messages. We assume that each node, from source to destination, is an AFN and may appear both inside the network and as an edge node to act as gateway. The three fundamental actions that an AFN can perform when it receives a message from a previous node are also represented. Messages have been represented using different colors to differentiate the possible actions to perform and to highlight they contain different forwarding codes. These actions are explained next:

- The message cannot be forwarded towards the next node at the moment and it is stored inside the current AFN (red color).
- The message is forwarded towards the next AFN without needing to be stored (green color).
- The message is replied and sent to two different AFN because this action is specified in the routing protocol which is included in the forwarding code (yellow color).

A. *Forwarding Code Triggers*

The forwarding code may include a set of rules to execute only a portion of the forwarding code after a specified situation. These execution rules or triggers are used in such a way that the occurrence of a particular event connotes the execution of the associated forwarding code. We have identified five forwarding code triggers, which are shown in Table I. Therefore, different functionalities can be executed at different times. In this way, several forwarding code fragments may be related to different triggers. Routing protocols may not implement a forwarding code for each of the five execution rules. They can also execute the forwarding code referred to some of them. As forwarding code size is a critical aspect when designing the framework, execution rules could be programmed using similar structures to registers. In this way, a forwarding code with minimal size could be achieved.

TABLE I. FORWARDING CODE EXECUTION RULES

Name	Description
RECEIVE_MESSAGE	When a node receives a message
SEND_MESSAGE	When a node sends a message
STORE_MESSAGE	When a message is stored in the node
DROP_MESSAGE	When a message is dropped by the node
NODE_TIMEOUT	Timer associated to a node

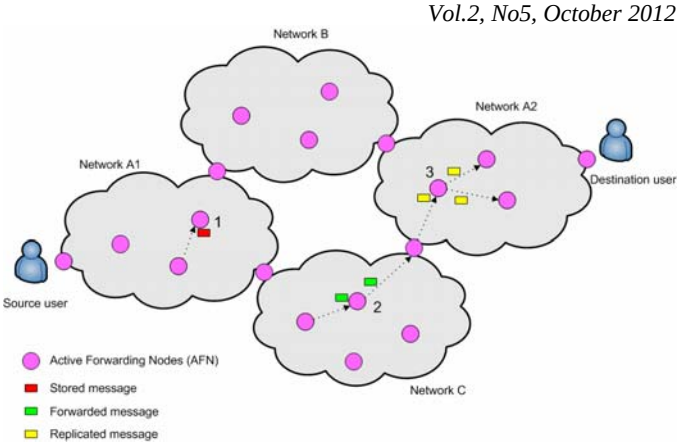


Figure 1. Messaging architecture for ICMN.

B. *Messages*

In this point, messages structure is explained. We distinguish between data and control messages. The former include the forwarding code, a storage area to collect data as they progress across the network and the payload containing application specific data. The latter are swapped by those nodes which need to know some information regarding other existing nodes. Both types of messages are forwarded by intermediate nodes.

1) *Data Messages*

We assume that all data messages have the same size and must include information about the destination node through an unique identifier. This information can also be assumed as destination address or hardware address to identify the node as single. Source node identification and a TTL field (similar to IP header), which is decremented after every hop, can also be included in the message header. We have already adopted that messages must include the forwarding code to be executed by the AFNs they encounter along the path. In this way, if a particular routing protocol requires some information from messages, it must implement the corresponding functionality in the forwarding code. Apart from the forwarding code, messages possess a data storage area called environment information heap. As they progress throughout the network, messages can store some necessary information in this area. Moreover, there is a timeout value in the message header in order to set the maximum time a message can be present inside the network before it is dropped. The source user sets this value keeping in mind the network volatility. We predict a high value for this parameter. The message environment information heap stores information required by the current AFN or history-based information needed for future executions of the forwarding code in the AFNs the message meets. One example of this type of information required by the current AFN may be the timestamp in which the message was stored into the AFN messages heap. As a history-based information example, we can consider the free space a previous node still has or even a list of nodes the message has crossed and the time spent to reach them. We can therefore assess the average time that a message lasts between two adjacent nodes. For this purpose, TLV (Type-Length-Value) fields can be used in a similar way than in IPv6. Data messages format is depicted in Fig. 2.

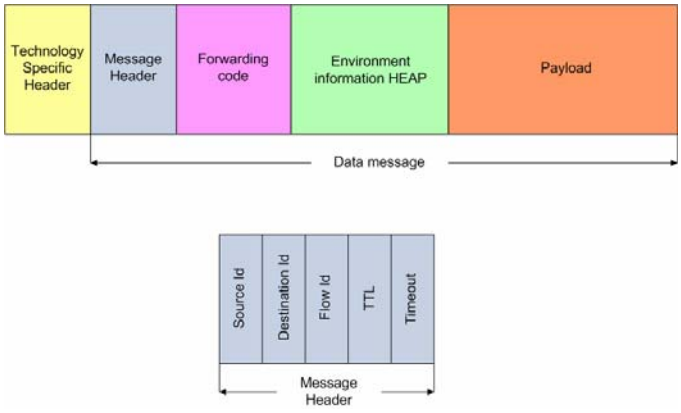


Figure 2. Data message format.

2) Control Messages

When the execution of the forwarding code is not enough to obtain required information from the rest of the nodes, control messages are used. This type of messages are exchanged between those adjacent nodes that explicit request a particular information. Piggybacking mechanism is exploited for this purpose. Control messages are always exchanged by two nodes with direct visibility. Once the requested information is transferred, both nodes can add it to the corresponding partition of their protocols heap. Future messages with the same routing scheme will benefit from this approach. Although data messages can be stored inside AFNs if there is no opportunity to directly forward them, control messages are not stored. This is due to prevent unnecessary consumption of the effective storage and bandwidth for forwarding. Fig. 3 shows control messages format.

C. Active Forwarding Nodes (AFN)

Each AFN contains a Forwarding Content Processor (FCP) to process the forwarding code in the arriving message stream. AFNs also provide a standardized way that injects local knowledge into the FCP to give the forwarding code the opportunity of using information it can glean from its current surroundings in a given moment. This information is typically related to localized reachability data and data referred to geographical regions or nodes which can be accessed by AFNs. In addition to providing a processing engine for the forwarding code extracted from the received messages, AFNs have the ability to encapsulate messages into different formats. These formats are appropriate for the underlying networks they have access. This feature includes the possibility of fragmenting messages and formatting header fields in accordance with the underlying technology constraints. AFNs capabilities are presented as inputs to the forwarding code so that it is aware of various forwarding technologies to consider. To deal with situations where no suitable forwarding technology is currently available, AFNs possess store-carry-and-forward capability in order to retain messages until communication resources become available.

Thus, an AFN is composed of an operating system and a framework (upper layer) that contains the execution environment. The latter has the ability of executing the forwarding code in the arriving messages. AFNs operation is explained next: they receive a message, execute the corresponding forwarding code and send it to a node according to the obtained result. The selected node has been chosen as the most suitable for the next hop. The next scheme explains AFNs components in the proposed framework:

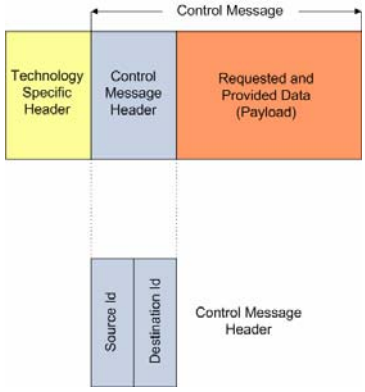


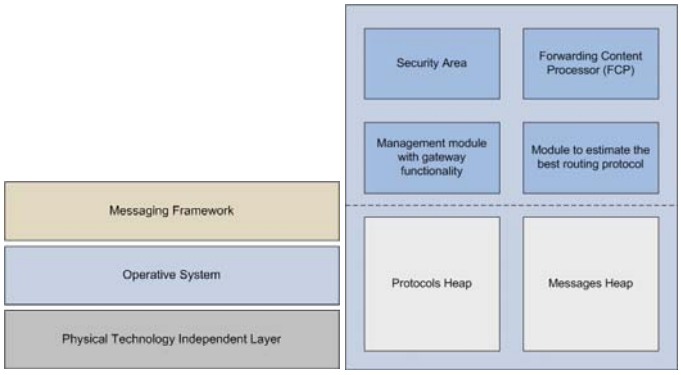
Figure 3. Control message format.

1) *Message Forwarding Framework.* It provides the programming language and the execution environment for the forwarding code.

2) *Operating System.* It is responsible of lower level management and planning.

In a lower level situated under the operating system, another layer, which is independent from the node physical technology, is present. As a proposal, we could use *Media Independent Handover* (MIH), a standard being developed by IEEE 802.21 to enable the communication between mobile devices with different underlying physical technologies [13]. The architecture of an AFN, which would implement this framework, is described in Fig. 4(a). The messaging framework gleans information from arriving messages in the AFN, executes the forwarding code included in messages and forwards it towards a next node chosen as the most appropriate. Thus, the AFN is an element with a basic function: executing the forwarding code included in the incoming message. Depending on the code, it forwards the message to a next node if possible, or stores it inside the AFN storage area. Regarding the logic structure, the AFN messaging framework is composed by six components:

- *Security Area.* It examines integrity and validity of arriving messages.
- *Forwarding Content Processor (FCP).* It executes the forwarding code.
- *Protocols Heap.* Storage area with the required information to execute routing mechanisms included in the forwarding code of arriving messages.
- *Messages Heap.* Storage area for those messages that could not be forwarded to the next node.
- *Management module with gateway functionality.* Module used to interconnect networks with different protocols and architectures.



(a) Architecture (b) Logic Structure

Figure 4. AFN Structure.

- *Module to estimate the best routing protocol.* Source nodes must also include a module to determine the best routing protocol to send their messages.

Fig. 4(b) shows the AFN logic structure, composed by the six aforementioned elements.

Due to the high volatility of the network, nodes may appear and disappear unexpectedly. With our proposal, nodes without any information about the network are able to participate in forwarding decisions to get the message closer to the destination. Hence, there are differences between a recently discovered node and another node which was discovered long time ago. Obviously, the latter has more information about the environment than the former and will be able to act more precisely. For that, when an AFN is discovered by the network, its protocol heap is initialized. This area stores information related to the routing protocols needed to execute the forwarding code in the incoming messages. It will be initially empty and updated as it receives messages with a forwarding code. The code contains the forwarding strategies corresponding to a certain protocol. If this protocol needs some information that must be provided by the network, the current AFN adds it to the protocols heap. Using this storage methodology based on protocol partitions, generality is obtained in such a way that an anomalous operation of a certain protocol does not affect the treatment of future messages from different protocols.

A buffer management scheme is used to administer protocols heap. We must define a maximum size for the heap although partitions may be variable for different protocols with different features. It is necessary to identify protocols needs and allocate storage space depending on them. This approach is used because some protocols need more information than others. For example, epidemic routing [4] only requires a list of neighbors to decide the next hop. Fig. 5 depicts an example where the protocols heap stores information for three different routing protocols. An AFN receives a message with a forwarding code for the routing protocol number 1. Then, the corresponding partition of the protocols heap is updated (or created) to execute the forwarding code of the message. Table II shows an example of information stored into one of the partitions included in the protocols heap of a certain AFN. Therefore, the protocols heap updates its information according to the routing protocols included in the forwarding code of the

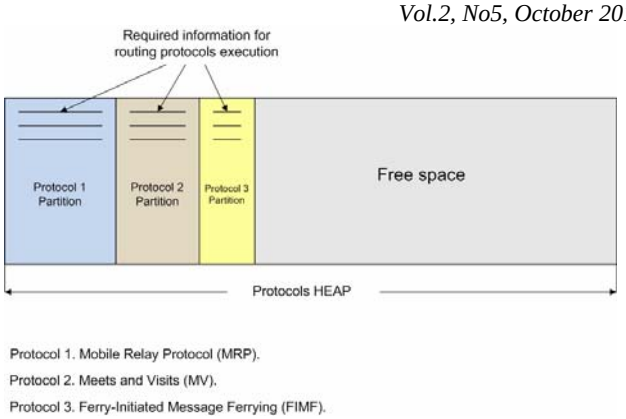


Figure 5. Protocols heap structure.

arriving messages. For those protocols not already included in the heap, the AFN allocates the corresponding space. This information stored in the protocols heap is only updated when the forwarding code of a message is executed. In this way, it can be updated either every time the AFN receives a message with the same routing scheme included in the forwarding code or every time the AFN receives a message. A timer can be also used, which may be the node local clock. Another possibility of execution could be either after a change in node neighborhood or when a new node is discovered.

Only stateful information can be stored in the AFNs protocols heap, i.e., the minimum information to let AFNs execute the forwarding code of a message. AFNs have a timer to release resources reserved for maintaining the information required by protocols if there are no updates for a predefined time. In this manner, protocols heap free space is increased to include new protocols that need to store some information after messages arrival. Apart from protocols heap, there is another area in AFNs for message storing: messages heap. Both storage areas are managed in a similar way. In this case, AFNs provide a finite area with the purpose of storing those messages which could not be forwarded after the execution of the forwarding code. An example of organization of this storage area is shown in Fig. 6. As in protocols heap, we need to define a maximum size for this area. In this case, the space reserved for each message is the same and, therefore, partitions in messages heap have all the same size. Initially, when a node is discovered, its messages heap is free, ready to be used if necessary. There is a timer to provide node dynamism in such a way that messages do not remain stored indefinitely. Those messages stored before the timeout was active are deleted in order to increase the free space available. Then, future messages can be stored.

TABLE II. EXAMPLE OF INFORMATION STORED INSIDE A PROTOCOLS HEAP PARTITION

Node identifier	Time since it was detected inside the coverage area	Is it a destination node for a stored message?
Node_ID1	00 min 35 sec 86 cent	No
Node_ID2	00 min 07 sec 23 cent	No
Node_ID3	00 min 02 sec 84 cent	Yes, Message_ID4

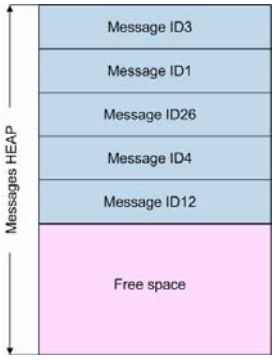


Figure 6. Messages heap structure.

Otherwise, if a message needs to be stored in an AFN and there is no space for it, the oldest message in the heap is deleted. Based on the AFN logic structure, we designed the AFN software structure. It depicts the way the incoming messages are treated in an AFN.

1) Security and Communication Module takes charge of receiving and sending messages. When an AFN receives a message, the first issue to do is related to security and integrity examinations.

2) When security examinations are successfully passed, the FCP check if the message includes the forwarding code. If so, the FCP executes it, else the Security and Communication Module sends the message to the next node randomly.

3) The FCP updates the corresponding partition in the protocols heap and performs some actions depending on the obtained result: if the message must be stored inside the messages heap, the FCP takes charge of storing it but if it must be forwarded then the FCP sends the message to the Security and Communication Module.

4) Finally, this module forwards the message towards the next node based on the result of the previous step.

Interaction between the source user (application) and the generic framework is limited to send and receive messages. The user decides to send a message and indicates the forwarding code to use. The first AFN includes this code inside the message and performs the first hop. It also acquires the information related to the network interaction pattern. In this way, the user can choose the most suitable forwarding code depending on the information expected to be received from the network. Some routing protocols may require operation parameters to be fixed at the beginning of the communication process. For that, the user can set the operating parameters needed to execute the forwarding code associated to messages. These parameters could require to be spread throughout the network. In that case, they should be transported inside the message, specifically in the heap of the message. For this purpose, the source node should include some execution code not only to obtain the best routing protocol to use but also to estimate the most suitable operation parameters to be included inside the message. Two possible parameters could be the number of copies of the message or the mobility degree of the network.

TABLE III. PRIMITIVES AND FUNCTIONS OF THE GENERIC FRAMEWORK

Group	Name of primitive/function
Primitives	*requestNodeCoverage, sendControlMessage, receiveControlMessage, sendDataMessage, receiveDataMessage, setMessageEmptyTimer, addProtocolHeapPartition, updateProtocolHeapPartition, getProtocolHeapPartition, storeMessage*
Protocols heap control functions	*setProtocolHeapUpdateTimer, setProtocolHeapEmptyTimer, setProtocolPartitionEmptyTimer*
Messages heap control functions	*setMessagesHeapEmptyTimer, storeMessage, isStored, replaceMessage, dropOlderMessage, getBufferFreeSize*
Stateful information management functions	*getTimeActive, getNodesMeetingFrequency, getUtilityNodesCoverage, setCoverageDistance, getHistoricalInfo, getConnectivityChange*

D. *Forwarding Code Primitives*

We propose a minimum set of primitives to be invoked by the forwarding code in order to implement the routing protocols with the proposed generic framework. The main goal is to guarantee that this minimum set of primitives is enough to program any routing protocol in ICMN. The framework also provides a set of functions which are implemented in AFNs. Both primitives and functions are described in Table III.

IV. IMPLEMENTATION OF A ROUTING PROTOCOL: SPRAY & WAIT

Spyropoulos et al. propose a routing protocol called Spray & Wait [5,6] in two versions (Source Spray & Wait and Binary Spray & Wait). Desirable goals are explained next:

- Perform fewer message transmissions than epidemic and other flooding-based routing schemes, under all conditions.
- Generate low contention, especially under high traffic loads.
- Achieve a better delivery delay than existing single and multi-copy schemes, and close to the optimal.
- Be highly scalable, i.e. maintain the performance behavior despite changes in network size or node density.
- Be simple and require as little knowledge about the network as possible, in order to facilitate implementation.

Spray & Wait is one of the most relevant routing protocols in ICMN. It consists of the following two phases:

- *Spray Phase:* For every message generated at a source node, L copies are initially spread to L distinct "relay" nodes (intermediate nodes).
- *Wait Phase:* If the destination is not found in the *Spray* phase, each of the L nodes carrying a message copy performs direct transmission.

In next sections, we present the implementation of the two versions of the routing protocol Spray & Wait using the generic framework.

A. *Source Spray & Wait*

Source node (first hop)

- It sends *L* copies of the message to the *L* first nodes it encounters. *L* value is set by source user (application). It is a necessary operation parameter (*num_copies*).
- If the number of neighbors in the neighborhood *n* is less than *L*, the source node sends *n* copies of the message, one copy to each neighbor in the list. After that, it waits the chance of forwarding the remaining *L-n* copies. In this case, the node will send the rest of the copies in future executions.
- No copies of the message will be stored inside the messages heap of the source node, because all the copies are distributed throughout the network.
- If a node belonging the coverage area is the destination node of the message (destination address check), the source node sends it a copy of the message and stops sending copies to the remaining nodes.

Pseudocode of the forwarding code to be executed by the source node is described next. Invoked primitives are written in italics.

```
start
    info_protocol=getProtocolHeapPartition (Source_Spray_And_Wait)
    if (isEmpty(info_protocol)) then
      L = num_copies
    else L = info_protocol (num_copies)
    end_if
    counter = 0
    destination = false
    nodesList = requestNodeCoverage
    while (NOT end nodesList) AND (destination == false) do
      if (counter < L) then
        sendDataMessage (msg, nodesList(counter))
        counter = counter + 1
        if (isDestinationNode (nodesList(counter))) then
          destination = true
        end_if
      end_if
    end_while
    if (destination == false) then
      if (counter != L) then
        addProtocolHeapPartition(Source_Spray_And_Wait,num_copies,
                                                    L-counter)
      end_if
    end_if
end
```

Intermediate nodes

- It receives a copy of the message which was sent by another node.
- It requests some information from the neighborhood.
- In case of no neighbors, it stores the message inside the messages heap.
- If it has neighbors, sends the copy of the message randomly to one of them.

```
start
    cases: RECEIVE_MESSAGE, NODE_TIMEOUT

    nodesList = requestNodeCoverage
    if (isEmpty (nodesList)) then
      storeMessage (msg)
    else
      node_id = random (nodesList)
      sendDataMessage (msg, nodesList(node_id))
    end_if
end
```

B. *Binary Spray & Wait*

Source node (first hop)

- It sends *L/2* copies of the message to the *L/2* first nodes it encounters. *L* value is set by source user (application). It is a necessary operation parameter (*num_copies*).
- If the number of neighbors in the neighborhood *n* is less than *L/2*, the source node sends *L/2* copies of the message to each neighbor in the list and waits the chance of also forwarding *L/2* copies to the *(L/2)-n* remaining nodes. In this case, the node will send the rest of the copies in future executions.
- It stores *L/2* copies inside the messages heap.
- If a node belonging the coverage area is the destination node of the message (destination address check), the source node stops sending copies to the remaining nodes.

```
start
    info_protocol=getProtocolHeapPartition(Binary_Spray_And_Wait)
    if (isEmpty(info_protocol)) then
      L = num_copies
    else L = info_protocol (num_copies)
    end_if
    counter = 0
    destination = false
    nodesList = requestNodeCoverage
    while (NOT end nodesList) AND (destination == false) do
      if (counter < (L/2)) then
        for i from 0 to (L/2) do
          sendDataMessage (msg, nodesList(i))
        end_for
        if (isDestinationNode (nodesList(counter))) then
          destination = true
        end_if
      end_if
    end_while
    if (destination == false) then
      if (counter != (L/2)) then
        addProtocolHeapPartition(Binary_Spray_And_Wait,num_copies,
                                                    (L/2)-counter)
        storeMessage (counter,msg)
      else storeMessage (L/2, msg)
      end_if
    end_if
end
```

Intermediate nodes

- It receives *X* copies of the message which was sent by the source node.
- It requests some information from the neighborhood.
- If *X > 1*, it sends *X/2* copies of the message to the *X/2* first nodes it encounters.
  - It stores *X/2* copies inside the messages heap.
- If *X = 1*, it sends a copy of the message to each node in coverage.
  - No copies are stored inside the messages heap.

```
start
    cases: RECEIVE_MESSAGE, NODE_TIMEOUT

    nodesList = requestNodeCoverage
    if (num_copies > 1) then
      counter = 0
      while (NOT end nodesList) do
        if (counter < (num_copies/2)) then
          for i from 0 to (num_copies/2) do
            sendDataMessage (msg, nodesList(i))
          end_for
        end_if
      end_while
      storeMessage (num_copies/2, msg)
    else
      for node in nodesList do
        sendDataMessage (msg, node)
      end_for
    end_if
end
```

V. CONCLUSION

In the last few years, researchers have proposed a wide variety of routing protocols for ICMN. However, these protocols are designed to be executed under particular conditions. Moreover, researchers show homogeneous results which cannot determine the best routing protocol for a particular scenario. This entails the absence of tools to give a node the ability of executing some of these protocols in a generic way.

In this work, we propose a generic framework to execute some of these ICMN routing protocols as a function of user choice. For that, we have defined the AFN architecture and messages format, both data and control messages. We have shown that routing protocols can be implemented with this framework using a reduced set of primitives and execution rules. Moreover, two storage areas (one in the AFN and another in the message) are needed to store stateful and history information respectively.

We have also demonstrated that one of the most relevant routing protocols in ICMN, Spray & Wait in both versions, can be implemented using the generic framework. Furthermore, we estimate a small size for the forwarding code to be executed by AFNs due to its simplicity. This fact entails a small message size. Thus, our proposal can be used as starting point for researchers in ICMN routing: they can implement and validate their routing protocols and create new potential ones to achieve better results.

ACKNOWLEDGMENT

This work was supported in part by Gobierno de Extremadura and European Regional Development funds (Ref: PRE09184 and GRU10116).

REFERENCES

[1] A. Shark. The new public square. American City & County. December 2010. Accessed September 2012 http://americancityandcounty.com/technology/e-government-applications-201012/

[2] A. Ferreira. Building a Reference Combinatorial Model for MANETs. IEEE Network, vol. 18, issue 5, pp. 24-29. September-October 2004.

[3] S. Jain, K. Fall, R. Patra. Routing in a Delay Tolerant Network. Proceedings of the ACM SIGCOM 2004, pp. 145-158. Portland, Oregon, USA, 30 August-3 September 2004.

[4] A. Vahdat, D. Becker. Epidemic Routing for Partially Connected Ad Hoc Networks. Technical Report CS-200006, Department of Computer Science, Duke University, Durham, NC, 2000.

[5] T. Spyropoulos, K. Psounis, C. S. Raghavendra. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking. pp. 252-259. Philadelphia, USA. August 2005.

[6] T. Spyropoulos, K. Psounis, C. S. Raghavendra. Single-copy Routing in Intermittently Connected Mobile Networks. Proceedings of the IEEE SECON 2004. pp. 235-244. Santa Clara, CA, USA. 4-7 October 2004.

[7] A. Lindgren, A. Doria, O. Schelen. Probabilistic Routing in Intermittently Connected Networks. ACM SIGMOBILE Mobile Computing and Communications Review, vol. 7, issue 3, pp. 19-20. July 2003.

[8] Y. Wang, H. Wu. DFT-MSN: The Delay Fault Tolerant Mobile Sensor Network for Pervasive Information Gathering. Proceedings of the IEEE INFOCOM 2006, pp. 1-12. Barcelona, Spain. April 2006.

[9] E. Jones, L. Li, Ward. Practical Routing for Delay Tolerant Networks. Proceedings of the ACM SIGCOMM-DTN Workshop 2005. Philadelphia, USA. 22-26 August 2005.

[10] Q. Li, D. Rus. Communication in Disconnected Ad Hoc Networks Using Message Relay. Journal of Parallel and Distributed Computing, vol. 63, issue 1, pp. 75-86. January 2003.

[11] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. Welch. Virtual Mobile Nodes for Mobile Ad Hoc Networks. 18th International Sympousium on Distributed Computing. Amsterdam, Netherlands. October 2004.

[12] K. Nichols, S. Blake, F. Baker, D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474. December 1998.

[13] A. De La Oliva, A. Banchs, I. Soto, T. Melia, A. Vidal. An overview of IEEE 802.21: media-independent handover services. IEEE Wireless Communications, vol. 15, no. 4, pp. 96-103, August 2008.

AUTHORS PROFILE

Jaime Galán-Jiménez received his Engineering degree in Computer Science Engineering at the University of Extremadura (Spain) in 2007, where he is currently working in the Computer Science and Communications Engineering Department. In 2009, he received a PhD grant from the Regional Government of Extremadura to conduct his research. His main research topics are intermittently connected mobile networks, energy efficient networks and interferences in wireless technologies.

Alfonso Gazo-Cervero received his PhD in computer science and communications from the University of Extremadura. He is currently the main researcher of the Advanced and Applied Communications Engineering Research Group (GITACA) of the University of Extremadura, where he also holds an assistant professor position. His research interests are mainly related to capacity planning, routing protocols, overlay networks and energy efficient networks.