IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 14, 2018, accepted May 4, 2018, date of publication May 16, 2018, date of current version June 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2837624

The Maximum Uniform Message Distribution Problem

HÉCTOR ZATARAIN-ACEVES, JOSÉ ALBERTO FERNÁNDEZ-ZEPEDA, AND CARLOS A. BRIZUELA, (Member, IEEE)

Department of Computer Science, CICESE Research Center, Ensenada 22860, Mexico

Corresponding author: Héctor Zatarain-Aceves (zatarain@cicese.edu.mx)

The work of H. Zatarain-Aceves was supported by the Mexican National Council on Science and Technology (CONACYT) of Mexico through the scholarship under Grant 339307.

ABSTRACT This paper introduces the maximum uniform message distribution (MUMD) problem, which can be present in delay-tolerant communication networks, where the destination of the messages is not present in the network. This type of behavior arises in scenarios of natural disasters or social conflicts where a global communication network is not available. In these scenarios, the people inside the affected area might use their mobile devices to communicate in an opportunistic manner. During this communication, the devices can duplicate, exchange, and gather messages with the intention of afterward delivering them to the global communication network. A device successfully delivers all messages in its memory when it reaches the global communication network. We model the MUMD as a computational problem and analyze the effectiveness of the existing opportunistic routing algorithms to solve this problem. Furthermore, we design an algorithm for the MUMD and perform experimental simulations to analyze its performance. Our results show that the greater the number of copies of messages and the more uniformly distributed through the network they are, the higher the probability that deliveries will be successful.

INDEX TERMS Emergency response, delay tolerant networks, maximum uniform message distribution problem, opportunistic routing algorithms.

I. INTRODUCTION

The breakdown of the communication infrastructure caused by unforeseen events such as natural disasters or social conflicts can hinder the relief work to people within the affected area. Some initiatives that have focused on mitigating the undesirable effect of failures in communications networks are the following. Google crisis response [1] consists of a set of informative tools about a natural disaster that can be helpful for people with Internet access. These tools include public alerts, a person finder, a crisis map, among others. The main limitation of this system is that it is useless in the absence of the Internet.

Twitmight [2] is a Twitter client for mobile devices with a disaster mode (activated by the user) to communicate without Internet access. Twitmight allows the transmission of tweets by using peer-to-peer communication. It creates a mobile ad-hoc network with the assumption that before the connectivity loss, all the mobile devices were connected to a server to obtain security certificates. This feature implies that new users cannot join the network during a disaster. Another drawback of Twitmight is that it is reactive and not proactive.

FireChat is a proprietary app for public and private message communication that works even without Internet access or cellular data [3]. This app has been used in several social conflicts where the Internet and cellular networks were censored or overloaded. Some examples are Iraq and Hong Kong in 2014, Ecuador and Catalonia in 2015, and more recently, at the Democratic National Convention in Philadelphia, USA in July 2016.

Other research efforts focus on analyzing different types of network technologies to determine which is the best to mitigate the lack of communication in post-disaster scenarios and to help the emergency response [4]–[8]. In general, delay tolerant networks (DTNs) or opportunistic networks (OPNETs) [9], [10] are suitable options to achieve this goal. The main purpose of DTNs is the dissemination of messages and the resource optimization of the network [11], [12].

In networking, a routing algorithm involves message transmission from a source to a destination. However, in some particular scenarios such as natural disasters or social conflicts, the destination may be not available temporarily in the network. For such scenarios, it is necessary to change the

paradigm of conventional routing to one that increases the probability that a message eventually reaches its destination.

One way to accomplish this task is by making copies of the message and distributing them across the network, ensuring that every vertex has a copy of the message. Since the location of the destination vertex is unknown, the above procedure intents to take the message ''closer'' to its destination; however, some aspects hinder fulfilling this objective, examples of them are the following.

If the network intends to transmit several messages and many of their destinations are not present, the previous approach might generate too many copies, overloading the communication channels, and decreasing the performance of the network, causing well-known problems, such as broadcast storms [13].

Because, for some applications, various network resources (storage capacity, bandwidth, battery, etc.) are quite limited, several vertices will compete for them when they try to maximize the number of copies of their messages. It is desirable to design a mechanism that restricts the generation of copies of each message to control this conflict of interest. Such a mechanism will also help to avoid network overloading.

Additionally, for this type of scenario in distributed systems, it would be unpractical to limit the number of copies by using a centralized entity. Thus, a distributed approach in which the controller makes decisions based only on local information seems to be more appropriate.

Taking into accounts these issues, in the following subsection, we describe a system that can help to temporarily mitigate the lack of conventional communication networks.

A. CONTRIBUTION OF THIS RESEARCH

This paper introduces the Maximum Uniform Message Distribution (MUMD) problem. Roughly speaking, the input to this problem is a dynamic graph with a set of messages stored in its vertices. The output for this problem is a distribution of copies of all the messages on the vertices of the network that maximizes the number of copies of each message, subject to the following restrictions. First, all the messages have the same number of copies in the network. Second, there is at most one copy of each message in each vertex. We modeled the MUMD as a computational problem, and to the best of our knowledge, this work is the first research paper to study this type of problem.

Our motivation to study the MUMD is that a solution for this problem can be useful for implementing a provisional communication network, based on wireless mobile devices, to temporarily replace traditional communication networks when they fail. This alternative network could allow the survivability of the network for successful delivery (to make a message from the affected area reach a functioning communication network). Although, the notion of survivability is not new, this issue is not entirely explored for mobile devices [14].

Additionally, we propose EDEN (Estimate, Disseminate and EmeNd copies), an opportunistic routing algorithm, which generates an approximate solution for the MUMD. Each mobile device executes EDEN in the provisional network to maximize the probability to successfully deliver a message. Finally, We analyze the performance of EDEN through numerical simulations.

B. CONTENTS OF THIS PAPER

The organization of this paper is as follows. Section II describes previous work related to opportunistic routing protocols. Section III formally describes the MUMD and introduces an application scenario. Section IV describes the EDEN algorithm. Section V describes the experimental design. Section VI discusses the results of the experimental simulations. Finally, Section VII presents various concluding remarks and ideas for future work.

II. RELATED WORK

This section contrasts the advantages and drawbacks of some existing routing algorithms for DTNs when we attempt to use them to solve the MUMD.

In the literature, there exist some routing protocols that can be useful, with some modifications, as an attempt to solve the MUMD. Many of them were designed for opportunistic networks. However, to the best of our knowledge, there is not any protocol that by itself solves the MUMD.

Existing routing protocols have adopted several types of forwarding strategies for relay selection, such as pure opportunistic, probabilistic or predictive-based, and social-based. The probabilistic or predictive strategy such as PROPHET [15], MaxProp [16], and MoVe [17] obtain promising results under specific scenarios [4], [18].

Because the MUMD assumes that the destination vertex is unknown or not available, a straightforward implementation of probabilistic or social-based routing algorithms is not possible. These algorithms require the presence of the destination vertex in the network to work appropriately.

One straightforward and naive approach for attempting to solve the MUMD is broadcasting the messages by flooding. One example of this approach is the protocol of epidemic flooding [19]. Unfortunately, flooding causes well-known problems, such as broadcast storms [13], overloaded communication channels, energy inefficiency, and unrestricted data redundancy.

Another more feasible approach that mitigates the above problems could be controlled flooding. One example of this mechanism is Spray and Wait (SnW) and its variants [20]–[22]. This algorithm could generate a controlled message distribution across the network.

SnW uses two phases. First, the spray phase restricts each vertex to generate only L copies of its message. This phase has two modes. In the simple mode, the source vertex sends one copy to L different vertices. In the binary mode, the source vertex sends L_i copies to the *i-th* different vertex, where $L_0 = L$ and $L_i = \lceil \frac{L_{i-1}}{2} \rceil$ for $1 \leq i \leq \lceil \log L \rceil$. Second, in the wait phase, each vertex with a single copy inhibits the

transmission of this message until it contacts the destination vertex.

SnW uses a parameter that depends on the number of vertices (assuming that each vertex knows the total number of vertices, n, in the network), which is a strong assumption in distributed computing systems [23]. SnW works by initially limiting the number of copies of each message to L. The suitable value for L depends on the application scenario; however, Spyropoulos *et al.* [24] suggest that for SnW with the binary mode, L should be between 10% to 15% of n.

The EDEN algorithm (see Section IV) does not assume that each vertex knows the parameter n. Furthermore, it does not require the presence of any specific destination vertex in the network. EDEN is a variation of SnW.

III. THE MUMD PROBLEM

This section formally defines the MUMD as a computational problem, i.e., as a relation between the input and the output. It also presents the application scenario and how to map the MUMD to such a scenario.

A. FORMAL DEFINITION OF THE MUMD

The input of the MUMD is an edge-dynamic undirected graph, defined by $\mathcal{G} = \{G_0, G_1, \ldots, G_{\phi-1}, G_\phi\}$, where in each undirected graph, $G_j = (V, E_j)$, $|V| = n$, and $E_j \neq E_{j+1}$ for all $j \geq 0$. Additionally, $|E_j| \neq 0$ for all $j \neq 0$ (G_0 is a graph with no edges). Each vertex $v \in V$ has a message m_v and a memory array b_v with the capacity to store κ_v messages.

At time slot j, nodes u and v can communicate if $(u, v) \in E_j$. Let $C_{i,j}$ be the set of copies of message i immediately after time slot j, for $1 \leq i \leq n$. Let M_j be the multiset of all messages in the vertices of V in graph G_j immediately after all the possible message transmissions occur through edges of E_j, i.e., $M_j = \bigcup_{i=1}^{n} C_{i,j}$. Let K be the maximum number of messages that V can store, i.e., $K = \sum_{j=1}^{n} \kappa_j$.

Let $\mathcal{C} = \min\{|C_{i,\phi}|\}$ for all $1 \leq i \leq n$, i.e., the number of copies of the message that has the minimum number of copies in G_ϕ.

The output of the MUMD is a multiset of messages M_ϕ that maximizes $\mathcal{C}$ subject to the following restrictions:

1) The number of copies $|C_{i,\phi}|$ is the same for all i.
2) There is at most one copy of each message in each vertex.

B. APPLICATION SCENARIO

Assume that there is a group of people in the affected area (without global communication) that have mobile communication devices. All of them want to send a message outside of the affected area (see Fig. 1a). Any pair of them can wirelessly communicate with each other via their devices if they are close enough (defined by the communication interface of their devices). In these encounters, these devices exchange copies of their messages to increase the probability of successful deliveries. The more copies they make, the higher the probability of successful deliveries. Eventually, somebody, somehow, could exit the affected area and access a global communication network with his/her mobile device. In this way, the mobile device can successfully deliver all the messages stored in it.

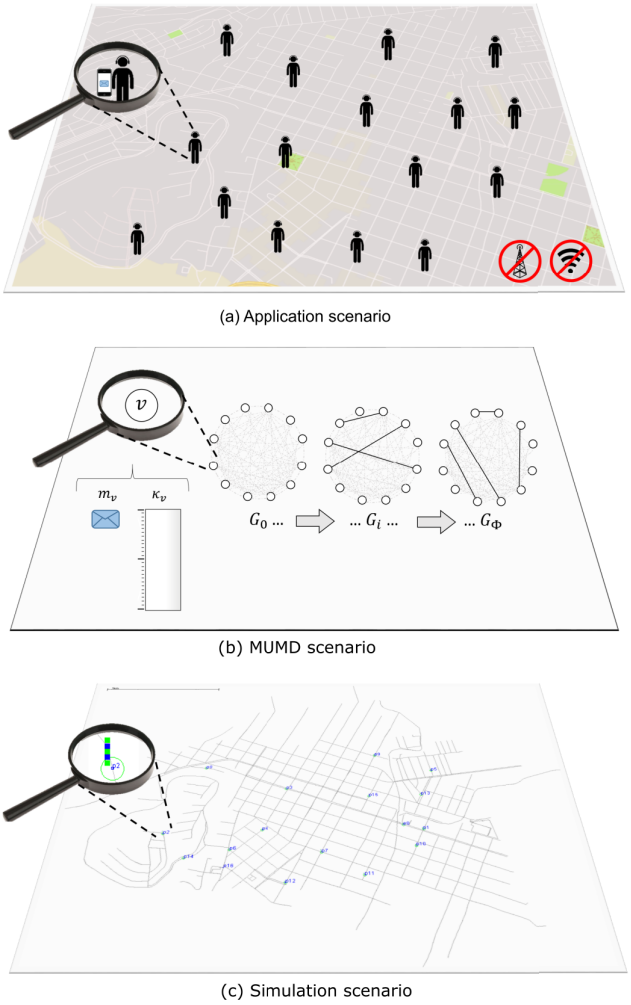


FIGURE 1. Three different levels of abstraction of the same scenario. (a) The application scenario of the MUMD, where there are a group of 16 people that have mobile communication devices in an area without global communication as described in Subsection III-B. (b) An example of input for the MUMD problem for the application scenario (described in Subsection III-C) that represents the set of all contact patterns between the 16 vertices. (c) A simulation scenario in the ONE simulator using an area with a map (described in Subsection V-C2) and 16 vertices.

To maximize the probability of a successful delivery for each message, the communication protocol of this provisional network would try to generate as many copies of each message as possible. This characteristic of the protocol is consistent with the maximization objective of the MUMD. At the same time, it has to eliminate the conflict of interest among the individuals. Because the sum, K, of the storage capacity of all the devices of the network is limited, this protocol restricts the number of copies L for each message to approximately $\lfloor \frac{K}{n} \rfloor$. This second characteristic of the protocol is congruent with the first restriction of the MUMD. Notice that in this scenario, it would be useless to have more than one

copy of a message in a device. For this reason, the protocol of the provisional network takes care of this issue. This third characteristic of the protocol is congruent with the second restriction of the MUMD.

Consequently, the communication protocol uniformly distributes messages over the entire network and uses the maximum possible storage capacity of the devices.

C. MAPPING THE MUMD TO THE APPLICATION SCENARIO

We model the application scenario with a dynamic graph. Some works use dynamic graphs to model dynamic networks [25], [26]. A dynamic graph changes its set of edges at any time t. Each vertex of this graph represents an individual with a mobile device (see Fig. 1b). This scenario assumes that the number of individuals is always n, so the number of vertices does not change in the graph.

Each edge in the graph, at time t, represents a contact between two mobile devices that can communicate (i.e., their distance in the application scenario is smaller than a given communication radius defined by their communication interface). Each communication contact is a consequence of the movement pattern of the individuals across the affected area in the application scenario. For this reason, the graph modifies its set of edges when the communication contacts change (i.e., the dynamic graph captures the movement of the individuals), as shown in Fig. 1b.

In this scenario, a vertex's capacity to store messages is proportional to the storage capacity of the mobile device that it represents.

Our assumptions for the communication model of this application scenario are the following:

1) All the vertices have the same communication interface.
2) Each vertex has a limited storage capacity to store messages.
3) Transmission opportunities are limited in duration and bandwidth (according to the movement model).
4) The vertices have no control over their movement.
5) The vertices have no a priori knowledge of the network size and connectivity.
6) The vertices do not know their geographic locations.

IV. EDEN ALGORITHM

This section describes the EDEN algorithm, its message exchange process, and its corresponding pseudocode. We use the 'binary transmission tree' to describe the message exchange process.

The EDEN algorithm spreads copies of the messages across the network considering the overall memory capacity of all the vertices in the network. The main idea of this algorithm is to allow bounded message replication (bounded to a suitable number of copies) of each message according to the global memory capacity in the network. The following subsection explains the message properties and the message spreading mechanism of this algorithm.

A. MESSAGE PROPERTIES FOR EXCHANGE

Each message m_v generated by vertex v has the following properties:

- $m_v.pnc$ indicates the permitted number of copies for message m_v.
- $m_v.sign$ is a binary suffix common to a set of copies of a message used in the message exchange process. We call this binary suffix the *signature*.
- $m_v.updated$ indicates whether $m_v.pnc$ was updated or not.
- $m_v.avIds$ is a sorted list of the available copy identifiers for message m_v. A vertex can generate this list by using the values of $m_v.sign$ and $m_v.pnc$ (see Subsection IV-B).
- $m_v.id$ indicates the unique identifier of a message in binary format.
- $m_v.anc$ indicates the available number of copies for message $m_v.id$.

Notice that a message only has to store the first three properties described above because, with them, a vertex can generate the others. The following subsection illustrates the message exchange process in the 'binary transmission tree.'

B. BINARY TRANSMISSION TREE

The procedure to make copies of the messages in the EDEN algorithm is a modified version of the spray phase with the binary mode of the SnW algorithm. This process is the following.

We define the 'binary transmission tree' to graphically illustrate the transmission process of message m_v over time, as shown in Fig. 2. The *binary transmission tree* (BTT) is a binary tree that has at most $(2^x - 1)$ BTT nodes, where x is the number of available copies for message m_v.

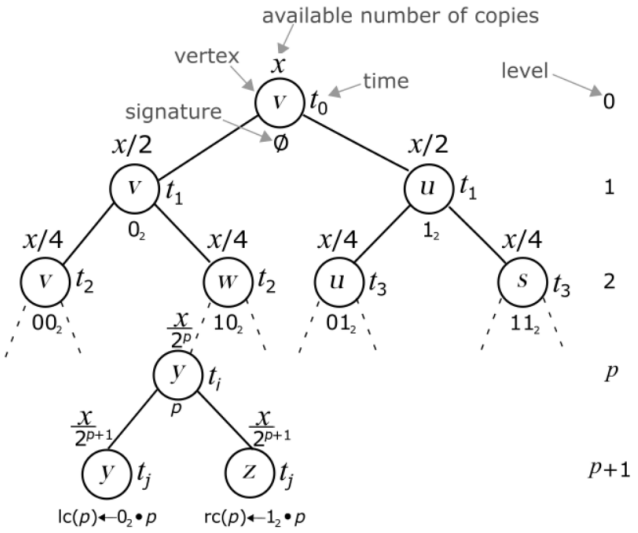


FIGURE 2. Example of the message replication tree for a single message m_v.

Each left child of the BTT relates to the same vertex as its parent (the transmitter). Alternatively, each right child associates with the vertex that receives a copy of the message

(the receiver). Each BTT node has a unique identifier specified by its signature.

Given a binary alphabet $\Sigma = \{0, 1\}$, Σ^n denotes the set of all binary strings over Σ of bit-length n. For example, $\Sigma^2 = \{00, 01, 10, 11\}$. Note, in Fig. 2, that Σ^i is the set of signatures of all BTT nodes at level i. The maximum level of a BTT of size x is $\log x$. The expression in (1) computes the set of signatures for the BTT nodes of a BTT of size x using the Kleene star [27] on Σ.

$$\Sigma^* = \bigcup_{i=0}^{\log x} \Sigma^i \tag{1}$$

where $\Sigma^i = \{0, 1\}^i$. Each BTT node represents a copy of message m_v. Note, in Fig. 2, that each BTT node contains various attributes, as explained next. The label inside the BTT node represents the id of the vertex that stores the copy of the message m_v. The number above each BTT node is the number of copies that the present copy can generate. The binary number below each BTT node represents the signature of the copy it stores. The level of a vertex is the number of relayed copies of that specific copy. The number on the right of each BTT node indicates the time at which the protocol generated/updated the signature of the copy. Next, we explain the generation process of the BTT for message m_v.

Now, assume that in the following contacts none of the contacted vertices have message m_v and that they have enough available memory space to store message m_v.

In general, let ρ be the signature of an internal BTT node. Let the signatures of its left and right children be $lc(\rho)$ and $rc(\rho)$, respectively. Then, $lc(\rho) \leftarrow 0_2 \bullet \rho$ and $rc(\rho) \leftarrow 1_2 \bullet \rho$, where the symbol '$\bullet$' indicates the concatenation of two strings.

Example 1: Assume that x is a power of two and that at time $t = t_0$, vertex v creates the initial copy of message m_v, with $m_v.pnc = x$ and $m_v.sign = \emptyset$ (the root of the BTT always has the signature $\emptyset$; see Fig. 2).

Assume that in the network vertex v contacts vertex u, and v transmits message m_v to u. Then, in the BTT, the BTT node with the signature $\emptyset$ generates two new BTT nodes, $lc(\emptyset)$ and $rc(\emptyset)$, with the signatures $lc(\emptyset) \leftarrow 0_2 \bullet \emptyset = 0_2$ and $rc(\emptyset) \leftarrow 1_2 \bullet \emptyset = 1_2$, both generated at time $t = t_1$. In Fig. 2, the two BTT nodes at level 1 represent these two copies stored in vertices v and u.

Similarly, assume that in the network vertex v contacts vertex w, and v transmits message m_v to w. Then, the BTT node with the signature 0_2 generates two new BTT nodes, $lc(0_2)$ and $rc(0_2)$, with the signatures $lc(0_2) \leftarrow 0_2 \bullet 0_2 = 00_2$ and $rc(0_2) \leftarrow 1_2 \bullet 0_2 = 10_2$, both generated at time $t = t_2$. The construction process of the BTT ends when it completes x leaves (i.e., after $x - 1$ transmissions of the message m_v).

In general, any vertex with signature ρ is able to generate all the copies of messages m_v whose identifiers, between 0 and $x - 1$, end with the suffix ρ. Specifically, after generating BTT nodes $lc(\rho)$ and $rc(\rho)$, each of them is able to generate all the copies whose identifiers are between 0 and $x-1$ and that end with the suffix $0_2 \bullet \rho$ and $1_2 \bullet \rho$, respectively. In the network, the transmitter keeps half of the available copies, and the receiver keeps the other half.

Initially, the number of permitted copies ($m_v.pnc$) is equal to the number of available copies ($m_v.anc$). The available number of copies decreases at half at each message transmission, i.e., $m_v.anc = m_v.pnc/2^{length(\rho)}$, where $length(\rho)$ denotes the number of bits (number of message transmissions) in ρ. The permitted number of copies ($m_v.pnc$) keeps the same value at each message transmission. It only changes at the time the estimator converges (see Subsection IV-D). The following subsection explains how to compute the permitted number of copies x used in the BTT and the message exchange process.

C. ESTIMATOR OF THE EDEN ALGORITHM

The SnW algorithm uses a fixed value L for all the vertices to limit the number of copies of each message. Meanwhile, the EDEN algorithm uses, for vertex v, a parameter $l_v(t)$ as a time-dependent local estimator of the optimum number of copies, L'.

Specifically, at time $t = 0$, vertex v initializes its local estimator to $l_v(0) = \kappa_v$. The EDEN algorithm regularly updates $l_v(t)$ during the execution of the algorithm by using the information collected from the local estimators of contacted vertices. This algorithm gradually adjusts each $l_v(t)$ until it approaches L', i.e., as time $t \to \infty$, $l_v(\infty) \to L'$. Previous works use a similar approach called distributed averaging [28], [29].

When a contact occurs between vertices v and u at time $t = t_k$, these vertices update their local estimators as follows: $l_v(t_k) = l_u(t_k) = \left(l_v(t_{k-\alpha}) + l_u(t_{k-\beta})\right)/2$, where $l_v(t_{k-\alpha})$ and $l_u(t_{k-\beta})$ are the previous values of the local estimators of v and u, respectively.

Each vertex uses an array called *historic* to store the *historic.size* most recent values of its local estimator. The local estimator is *stable* if the difference between the current value of the local estimator and each of the elements in the *historic* array is less than a predefined ϵ. Subsection IV-E1 explains the operation of this array.

Assume that at time $t = t_\sigma$, the estimator $l_v(t_\sigma)$ is stable. Then, at time $t = t_\sigma$, vertex v removes from its memory each message whose copy number is greater than $l_v(t_\sigma)$. Additionally, vertex v incentives the creation of new copies of each message whose copy number is smaller than $l_v(t_\sigma)$. The following subsection explains these two mechanisms.

D. UPDATING THE PERMITTED NUMBER OF COPIES

When a vertex v detects that its estimator $l_v(t)$ is stable, v compares $l_v(t)$ against $m_r.pnc$ for all the messages m_r stored in v. When they differ, vertex v performs the following mechanisms for each message in its memory. From now on, for simplicity, we refer to $l_v(t)$ as l_v.

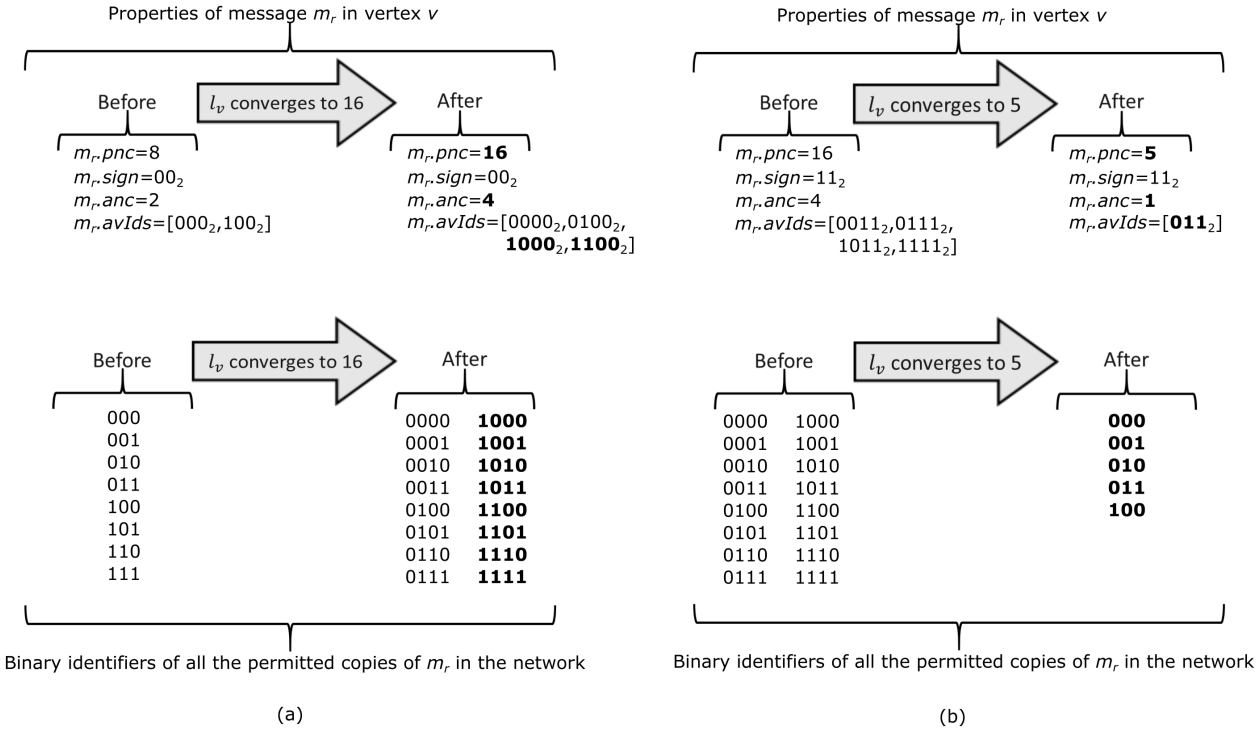


FIGURE 3. Examples of properties and binary identifiers for message m_r. Numbers in bold are the updated values. (a) Copy promotion. (b) Copy depuration.

1) COPY PROMOTION

If $m_r.pnc < l_v$, then message m_r requires more copies. To achieve this, v replaces $m_r.pnc$ with the value of l_v. For the algorithm, this change is transparent (i.e., it does not need to perform any special action to generate the extra copies for m_r).

Example 2: Assume that a vertex v contains a message m_r with a permitted number of copies equal to 8 (i.e., $m_r.pnc = 8$) and $m_r.sign = 00_2$. Also assume that v detects that $l_v = 16$ is stable. From these values, vertex v computes $m_r.anc = 2$ and $m_r.avIds = [000_2, 100_2]$; i.e., all the binary identifiers from 000_2 to 111_2 with suffix 00_2 (see the left side of Fig. 3a). Updating the value of $m_r.pnc$ from 8 to 16 means that there should now be 16 copies of message m_r with identifiers from 0000_2 to 1111_2. Note that from these 16 identifiers, there are 4 with suffix 00_2. Therefore, it results in more available copies of m_r in vertex v; i.e., $m_r.anc = 4$ and $m_r.avIds = [0000_2, 0100_2, 1000_2, 1100_2]$ (see the right side of Fig. 3a).

2) COPY DEPURATION

If $m_r.pnc > l_v$, then m_r was allowed an excessive number of copies. To correct this excess, the algorithm executes one of the following two cases for each message. Similar to copy promotion, the first case only updates the permitted number of copies, while the second case detects that the copy of the message is one of the surplus and then removes it.

- Case *a*: If $(l_v - 1)_2 \geq m_r.sign$, then vertex v replaces $m_r.pnc$ with the value of l_v and allows the procedure of message exchange to work normally.
- Case *b*: If $(l_v - 1)_2 < m_r.sign$, then vertex v removes the message m_r.

Example 3: Assume that a vertex v contains a message m_r with $m_r.pnc = 16$ and $m_r.sign = 11_2$. Also assume that v detects that $l_v = 5$ is stable. Then, vertex v performs Case *a* of the copy depuration mechanism. Before the execution of the copy depuration procedure, $m_r.avIds = [0011_2, 0111_2, 1011_2, 1111_2]$; i.e., all the binary identifiers from 0000_2 to 1111_2 with suffix 11_2 (see the left side of Fig. 3b). After executing the copy depuration procedure, $m_r.avIds = [0011_2]$; i.e., all the binary identifiers from 0000_2 to 0011_2 with suffix 11_2 (see the right side of Fig. 3b).

E. PSEUDOCODE OF EDEN ALGORITHM

Pseudocode 1 describes the EDEN procedure for vertex v. Each vertex v in the network locally executes this procedure. The EDEN procedure receives two integer values as input, $historic.size$ and ϵ, to evaluate the convergence of the estimator. The description of Pseudocode 1 is the following.

Step 1 initializes the estimator l_v equal to the value of the memory capacity of vertex v. Step 2 initializes the Boolean flag *stable*, which indicates whether l_v is stable or not. Step 3 initializes the *historic* array that contains the $historic.size$ most recent values of l_v. Step 4 initializes a *counter* that indicates the number of insertions in *historic*. In Step 5, vertex v waits until it detects the presence of some vertex u

Pseudocode 1 EDEN

Input: The $historic.size \in \mathbb{N}$ and a positive constant ϵ for the estimator values.

1: $l_v \leftarrow \kappa_v$
2: $stable \leftarrow false$
3: $historic[0] \leftarrow l_v$
4: $counter \leftarrow 1$
5: **while** $\exists\ u$ in communication range with v **do**
6: UPDATE-ESTIMATOR()
7: UPDATE-MESSAGES()
8: $s_v\ (2) \leftarrow$ CREATE-AVAILABLE-SUMMARY(2)
9: TRANSMIT($s_v\ (2)$)
10: $s_u\ (2) \leftarrow$ RECEIVE()
11: $r'_{v,u} \leftarrow$ CREATE-REQUESTED-SUMMARY($s_u\ (2)$)
12: TRANSMIT($r'_{v,u}$)
13: $r'_{u,v} \leftarrow$ RECEIVE()
14: EXCHANGE-MESSAGES($r'_{v,u}\ r'_{v,u}$)
15: **end while**

Pseudocode 2 UPDATE-ESTIMATOR

1: **if** $stable = false$ **then**
2: TRANSMIT(l_v)
3: $l_u \leftarrow$ RECEIVE()
4: $l_v \leftarrow (l_v + l_u)/2$
5: **if** $counter \geq historic.size$ **and** $|l_v - historic[i]| \leq \epsilon,\ \forall i \in [0, historic.size - 1]$ **then**
6: $stable \leftarrow true$
7: **else**
8: $historic[counter \bmod historic.size] \leftarrow l_v$
9: $counter \leftarrow counter + 1$
10: **end if**
11: **end if**

Pseudocode 3 UPDATE-MESSAGES

1: **if** $stable = true$ **then**
2: **for all** message m_r in b_v **do**
3: **if** $m_r.pnc > l_v$ **and** $m_r.sign > (l_v - 1)_2$ **then**
4: remove message m_r from b_v
5: **else if** $m_r.updated = false$ **then**
6: $m_r.pnc \leftarrow stochastic\_round(l_v)$
7: $m_r.updated \leftarrow true$
8: **end if**
9: **end for**
10: **end if**

in its communication range. When v detects such a vertex u, v verifies that u is not busy transmitting to another vertex. If u is available, then the data exchange process starts between both vertices (Steps 6 through 14). In Step 6, v updates its local estimator (see Pseudocode 2). In Step 7, v updates the messages in its memory (see Pseudocode 3). In Step 8, v generates its 'available message summary' (see Pseudocode 4). In Steps 9 and 10, v transmits this summary to u and receives the summary from u, respectively. In Step 11, v computes its 'requested message summary' (see Pseudocode 5). In Steps 12 and 13, v transmits this summary to u and receives the summary from u, respectively. In Step 14, v transmits all the messages requested by u and receives all the messages that it requested from u (see Pseudocode 6). Next, v continues the execution at Step 5 of Pseudocode 1.

Remark 1: The vertex with the smallest identifier takes the lead of the communication channel. In this explanation, we assume that the identifier of vertex v is smaller than that of u.

1) UPDATE-ESTIMATOR PROCEDURE

In the Update-Estimator procedure, vertex v evaluates if l_v becomes stable as described in Subsection IV-C. Pseudocode 2 describes this procedure.

The EDEN algorithm uses the *historic* array as a circular queue to store the $historic.size$ most recent values of l_v. This array computes the value ($counter$ mod $historic.size$) to determine the position in the array where the following value of l_v is going to be stored. The description of Pseudocode 2 is the following.

Step 1 evaluates if l_v is not stable. If l_v is already stable, then this step exchanges the estimators and exits; otherwise, continue to Step 2. In Steps 2 and 3, v transmits its local estimator l_v to u and receives the estimator l_u from u, respectively. Step 4 replaces the estimator l_v. Step 5 evaluates if l_v is stable. If it is not, the procedure inserts this value in the *historic* array, updates the *counter* variable, and exits (Steps 8 and 9). If l_v is stable, then Step 6 sets *stable* to true.

2) UPDATE-MESSAGES PROCEDURE

In the Update-Messages procedure, vertex v updates the messages in its memory according to the procedure described in Subsection IV-D. Pseudocode 3 describes this procedure.

Step 1 evaluates if l_v is stable. If l_v is already stable, then continue to Step 2; otherwise, exit this procedure. Steps 2 through 9 evaluate and update the available number of copies of each message in the buffer of v, as explained in Subsection IV-D, and exit. The number of copies must be an integer value; hence, in the case that l_v converges to a non-integer value, Step 6 rounds this value via stochastic rounding, e.g., if the fractional part of l_v is 0.6, choose randomly among $l_v + 0.6$ and $l_v - 0.4$, with probabilities 0.6 and 0.4, respectively.

3) CREATE-AVAILABLE-SUMMARY PROCEDURE

The Create-Available-Summary procedure receives as input an integer x and outputs a summary of messages in vertex v, called $s_v(x)$. Pseudocode 4 describes this procedure.

Let $b_v.size$ be the number of messages stored in b_v. Let $s_v(x)$ be the set of identifiers of the messages that have at least x available copies in vertex v, e.g., $s_v(1)$ is the set of identifiers of all the existing messages in b_v. Notice that $b_v.size = |s_v(1)|$. We assign the name *available message*

Pseudocode 4 CREATE-AVAILABLE-SUMMARY

Input: A value $x \in \mathbb{N}$, representing the minimum remaining copies of each message in the summary to be generated.

1: $s_v(x) \leftarrow \emptyset$
2: **if** $b_v.size > 0$ **then**
3: **for all** message m_r in b_v **do**
4: **if** $m_r.anc \geq x$ **then**
5: $s_v(x) \leftarrow s_v(x) \cup m_r.id$
6: **end if**
7: **end for**
8: **end if**
9: **return** $s_v(x)$

Pseudocode 5 CREATE-REQUESTED-SUMMARY

Input: A set $s_u(2)$, representing the summary of the available messages of the other vertex.

1: $s_v(1) \leftarrow$ CREATE-AVAILABLE-SUMMARY(1)
2: $r_{v,u} \leftarrow s_u(2) \setminus s_v(1)$
3: **if** $|\kappa_v - b_v.size| < |r_{v,u}|$ **then**
4: $\alpha \leftarrow (|r_{v,u}| - |\kappa_v - b_v.size|)$
5: randomly remove α elements of $r_{v,u}$
6: **end if**
7: **return** $r_{v,u}$

summary to the set $s_v(2)$, i.e., the set of identifiers that have at least two available copies (one to keep and one to send). Notice that $s_v(2) \subseteq s_v(1)$.

Step 1 initializes the summary, and Step 2 checks if there is at least one message in the memory of v; if so, Steps 3 through 7 evaluate the number of available copies ($m_r.anc$) of each message m_r in b_v. If $m_r.anc$ is at least x, then the procedure adds m_r to the summary. Finally, Step 9 returns the requested summary.

4) CREATE-REQUESTED-SUMMARY PROCEDURE

In the Create-Requested-Summary procedure, vertex v builds $r_{v,u}$, the *requested message summary* of v, i.e., the set of identifiers of the messages that v requests from u. Notice that $r_{v,u} \subseteq s_u(2) \setminus s_v(1)$. Let $r_{v,u}$ be at most $|\kappa_v - b_v.size|$ identifiers randomly chosen from the set $\{s_u(2) \setminus s_v(1)\}$.

This procedure receives as input the set $s_u(2)$. Step 1 computes the set $s_v(1)$ with Pseudocode 4. This set contains the identifiers of all the existing messages in the memory of v. Step 2 obtains $r_{v,u}$, the set of identifiers of all the available messages in u that v does not have. Steps 3 through 6 randomly choose from the set $\{s_u(2) \setminus s_v(1)\}$ the identifiers of those messages that b_v is able to store. Finally, Step 7 returns the requested message summary.

5) EXCHANGE-MESSAGES PROCEDURE

The Exchange-Messages procedure exchanges those messages specified by $r_{v,u}$ and $r_{u,v}$ between v and u. Pseudocode 6 describes this procedure.

Pseudocode 6 EXCHANGE-MESSAGES

Input: A pair of sets $r_{v,u}$, $r_{u,v}$, representing the requested messages of each vertex.

1: **if** $|r_{u,v}| > 0$ **then**
2: **for all** $m_r.id$ in $r_{u,v}$ **do**
3: TRANSMIT(m_r)
4: $m_r.sign \leftarrow 0 \bullet m_r.sign$
5: **end for**
6: **end if**
7: **if** $|r_{v,u}| > 0$ **then**
8: **for all** $m_r.id$ in $r_{v,u}$ **do**
9: $m_r \leftarrow$ RECEIVE()
10: $m_r.sign \leftarrow 1 \bullet m_r.sign$
11: store message m_r in b_v
12: **end for**
13: **end if**

When vertex v sends (receives) a message m_r to (from) u, it updates the signature $m_r.sign$ of m_r. To update $m_r.sign$, vertex v concatenates a zero bit (one bit) in the most significant bit of $m_r.sign$. Subsection IV-B describes this update procedure.

F. EDEN WITH WARM-UP ALGORITHM

The *EDEN with warm-up* (EDEN_{wu}) algorithm is a simplified version of EDEN. During the first stage (the warming-up), each vertex v only exchanges its estimator and is allowed to receive messages. Vertex v ends this stage when its estimator converges. During the second stage, each vertex v starts the message exchange procedure as in EDEN, but it ignores the estimator update, copy promotion, and copy depuration procedures. Notice that EDEN executes these two stages simultaneously.

The main reason to introduce EDEN_{wu} was to analyze which strategy has better performance:

- EDEN strategy: starting the transmission of messages with an estimator likely to be far from L' and requiring an additional mechanism for correcting the number of generated messages.
- EDEN_{wu} strategy: starting the message transmission after computing an estimator very close to L' and without requiring any a posteriori correction in the number of generated messages.

The pseudocode for EDEN_{wu} is the same as that of EDEN except for the following minor changes in Pseudocodes 1 and 6. In Pseudocode 1, insert before Step 6 the validation "**if** *stable* = *false*" and insert the line "**end if**" just after Step 7. Additionally, in Pseudocode 6, insert before Step 1 the validation "**if** *stable* = *true*" and insert the line "**end if**" just after Step 6.

V. EXPERIMENTAL DESIGN

This section presents the experimental design used to compare the performances of the algorithms used for the MUMD. This experimental design applies to all the experiments.

A. METRICS FOR PERFORMANCE EVALUATION

The opportunistic routing protocols use mechanisms to improve several performance metrics, such as throughput, delivery ratio, storage usage, battery lifetime, information spreading [25], among others. However, most of these metrics are meaningless or inadequate in the context of the MUMD, (e.g., the most relevant metrics are delivery ratio and average message delivery latency, but both cannot be computed without a destination). For this reason, we introduce a set of metrics that are suitable to measure the quality of the output generated by the algorithms that intend to solve the MUMD.

The purpose of these metrics is three-fold. First, to evaluate the uniformity in the number of copies of each message. Additionally, to measure the memory usage of all the vertices in the network. Finally, to compute the number of vertices that must be selected at random to obtain at least one copy of each message. The following subsections describe these metrics.

1) RELATIVE UNIFORMITY FACTOR

The *Relative Uniformity Factor* (RUF) evaluates the uniformity of the numbers of copies of the messages in the network. The RUF is equal to the product of the numbers of copies of all the messages divided by the average of these numbers, as shown in (2).

$$\text{RUF} = \prod_{i=1}^{n} \frac{C_i}{\bar{C}}, \tag{2}$$

where C_i is the number of copies of message i in the network and $\bar{C}$ is given by (3).

$$\bar{C} = \sum_{i=1}^{n} \frac{C_i}{n} \tag{3}$$

One property of this metric is that the higher the uniformity of the numbers of copies is, the greater the value of the RUF is. The RUF obtains its maximum value when the numbers of copies of all the messages are the same. In such a case, the value of the RUF is equal to one. When the number of copies of a message is zero, the value of the RUF is zero, its minimum possible value.

2) MEMORY UTILIZATION FACTOR

The *Memory Utilization Factor* (MUF) measures the occupancy of messages in the memory of all the vertices in the network. The MUF is equal to the sum of the total number of messages in the network divided by the maximum capacity of storage K in the network, as shown in (4).

$$\text{MUF} = \sum_{i=1}^{n} \frac{C_i}{K} \tag{4}$$

The value of the MUF varies between zero and one and is proportional to the percentage of the storage capacity of the network that has been used to store messages.

3) NUMBER OF DRAWS TO OBTAIN THE COLLECTION OF MESSAGES

This metric indicates the average number λ of vertices that are necessary to select from the network to obtain the set Ω_n. We call Ω_n the set of all the messages in the network. This metric assumes a random selection without replacement and with a uniform distribution. It also assumes that each selected vertex transmits all the messages in its memory to their destinations.

In the application scenario, the selected vertices represent those mobile devices that could exit the affected area and manage to contact a global communication network.

The parameter λ can take any value between $\{2, \ldots, n\}$ since we are assuming that $\kappa_v < n$ for all v. Notice that the lower the value of λ is, the better.

From this formulation, some interesting questions are the following: First, what is the average number of vertices that must be selected to obtain $\alpha \cdot |\Omega_n|$ different messages for some constant $0 < \alpha < 1$? Second, what is the value of λ if we assume an optimal uniform distribution of messages in the network? Notice that this is the best value that, on average, the best algorithm for the MUMD can obtain. Given the output of the MUMD M_ϕ, we experimentally measured λ.

Remark 2: The context of this metric is similar to a variation of the coupon collector problem with group drawings [30], [31].

Remark 3: The plots of Figs. 9, 12, 14, 15, 16, 17 and 19 use λ/n rather than λ since it is easier to compare the results of the experiments for various values of n. We call the ratio λ/n *Normalized Average Number of Draws* (NAND).

It is worth noting that the metrics RUF and MUF are good indicators of the copy uniformity of messages and the memory utilization, respectively; however, they are not correlated with the quality of the candidate solutions for the MUMD. The product of the RUF and MUF (denoted by RUF×MUF) combines the behavior of these two metrics into one single metric. This metric and λ are correlated with the quality of the candidate solutions for the MUMD. Moreover, for the practical scenarios described in Subsection III-B, between these two metrics, λ is probably the most suitable metric for the MUMD.

4) CONVERGENCE TIME

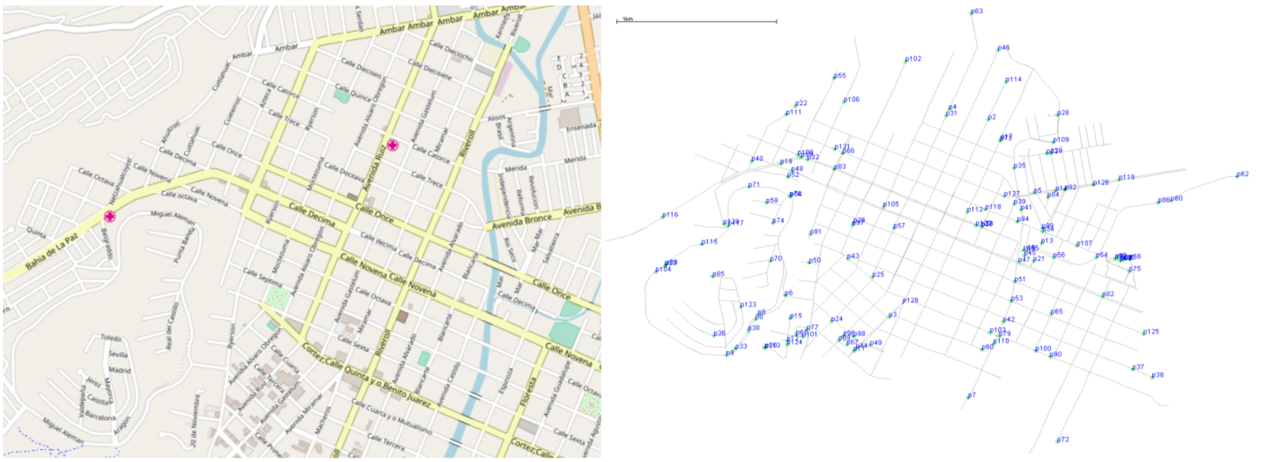
The *saturation time* is the time when the storage occupancy in the network is maximum, i.e., MUF =1. Similarly, the *convergence time of the network* is the time when the occupancy is maximum and, for each vertex $v \in V$, the content of b_v does not change.

B. NETWORK SIMULATOR

We performed the experiments using *the Opportunistic Network Environment* (ONE) simulator [32] version 1.5.1 RC2. We selected this simulator because it is open-source software, commonly used in DTNs research works, and have several implementations of well-known DTNs routing algorithms.

TABLE 1. **Simulation setup for the experiments of Scenarios *A*, *B*, *C*, *D*, and *E*.**

Parameter	**Scenario *A***	**Scenario *B***	**Scenario *C***	**Scenario *D***	**Scenario *E***
Number of vertices (n)	$16, 32, 64, 128$				
Transmission range	10 m				
Transmission velocity	250 kBps				
Speed of movement	0.5 - 1.5 m/s				
Random seeds	100				
Message size	1 kB				
Message creation	One message per node at $t = 0$				
Simulation time	200,000 seconds				
Distribution of buffers	Ub	Sb	Nb		
Mobility model	Random waypoint		Shortest path map-based		
Energy scheme	Unrestricted		Fully-charged	Random-charged	Unrestricted
Area of scenario	590 m × 532 m		4000 m × 3000 m		
Maximum arrival time	0				10k | 20k | 40k



(a) Map obtained from https://www.openstreetmap.org

(b) Map used in ONE simulator for Scenario D.

FIGURE 4. **Maps of the city of Ensenada, Baja California, Mexico. (a) The original map in OpenStreetMap. (b) The map generated for the simulation.**

The purpose of the computational experiments was to compare different routing protocols using the metrics RUF, MUF and NAND. The ONE simulator allows the representation of the scenario associated with the MUMD problem and the implementation of the proposed algorithms. The experiments described in this section used the parameter settings of Table 1. We implemented the following assumptions in the simulation scenario to accurately simulate the MUMD problem:

1) Each vertex generates only one message at the start of the simulation.
2) The final message destinations are not in the network.

C. PARAMETER SELECTION

We performed preliminary experiments to determine the appropriate parameter values of the simulator to emulate the MUMD. Table 1 lists the parameters setup, and the following subsections describe them.

1) NUMBER OF VERTICES

The number of vertices, n, in the simulation area takes values from the set {16, 32, 64, 128}.

2) AREA OF THE SIMULATION

The experimental simulations use two areas of rectangular shape. The first area does not have spatial constraints, and its size is 590 m × 532 m. The second area considers spatial constraints. It corresponds to a map of the city of Ensenada, Baja California, Mexico (see Fig. 4), and its size is 4000 m × 3000 m.

3) COMMUNICATION INTERFACE

Bluetooth and Wi-Fi are the most common communication interfaces used in opportunistic networks. We considered Bluetooth because it is more widely used than Wi-Fi in opportunistic communications with heterogeneous mobile devices. In practice, most Bluetooth devices easily achieve a transmission range of 10 m and a transmission velocity of 250 kBps, regardless of any obstacles or interference.

4) MOBILITY MODELS

We use the 'random waypoint' and 'shortest path map-based' models implemented in the ONE simulator. The velocity range for these models is between 0.5 and 1.5 m/s to emulate the average human walking pace.

The *shortest path map-based* model works similar to Random Waypoint. First, it chooses a random destination point on the map. Second, it computes the shortest path between the current vertex location and the destination point. Finally, it moves the vertex from its current destination to the destination through the computed path.

5) MESSAGE PARAMETERS

The message size is restricted to 1 kB. This size is useful for a wide variety of scenarios modeled by the MUMD problem. We assumed that each vertex generates only one message at the time it joins the network.

6) VERTEX STORAGE PARAMETERS

Our simulation scenarios assume $k_v < n$ and three different distributions of storage capacity for the vertices: 'Uniform buffers' (Ub), 'Semi-uniform buffers' (Sb), and 'Non-uniform buffers' (Nb).

- *Uniform buffers*. In this distribution, all the vertices have the same buffer size, $\kappa_v = n/2$, i.e., each vertex can store at most $n/2$ different messages.
- *Semi-uniform buffers*. This distribution equally divides the set of n vertices into four groups, and the groups have buffer sizes of $n/2$, $n/4$, $n/8$, and $n/16$, respectively.
- *Non-uniform buffers*. This distribution equally divides the set of n vertices into $n/2$ groups. The buffer size of each group is i, where $1 \leq i \leq n/2$.

We do not consider the scenarios where $k_v \geq n$, for any vertex. Because it can be solved straightforward with any routing algorithm that floods the network. A more challenging scenario for the MUMD is when $k_v < n$, for all the vertices. Table 2 shows the optimal value of l_v for each buffer distribution and different numbers of vertices.

TABLE 2. The optimal value of the estimator l_v (L') for different numbers of vertices and buffer distributions.

Number of vertices (n)	Value of L'		
	Ub	Sb	Nb
16	8	3.75	4.5
32	16	7.5	8.5
64	32	15	16.5
128	64	30	32.5

7) ENERGY MODEL AND PARAMETERS

Our simulation assumes the energy model and parameter values proposed by Rodrigues-Silva *et. al.* [33]. The authors of this model measured the average energy consumption of the Nokia E52 mobile phone using the Bluetooth interface. They obtained that the energy consumption for scanning, sending, and receiving files are 0.06 mW/s, 0.08 mW/s, and 0.08 mW/s, respectively. The battery capacity of this phone is 4800 mW/s.

This energy model is suitable for our application scenario. The ONE simulator already includes an implementation of this model, which has been used recently in some research works [34]–[36] with the same parameter values proposed in [33].

We considered three different energy schemes (unrestricted, fully-charged, and random-charged) for the capacity of the battery. The first scheme assumes that each vertex has unrestricted energy. The second scheme assumes that each vertex has a fully-charged battery at the beginning of the simulation. The third scheme assumes that the initial energy level of each battery is a random value with a uniform distribution between 50% and 100% of the maximum capacity. No scheme considers the possibility of recharging the battery.

8) VERTEX MAXIMUM ARRIVAL TIME

Our simulation assumes four different vertex arrival times schemes. These schemes assume that vertices join the network randomly with four uniform distributions in the intervals [0, 0], [0, 10k], [0, 20k], and [0, 40k] seconds. In particular, interval [0, 0] means that all vertices join the network at the beginning of the simulation.

9) SIMULATION SCENARIOS

This subsection presents five simulation scenarios. Each scenario has different parameters as shown in Table 1.

- *Scenario A*. Buffer distribution Ub and in an area without spatial constraints.
- *Scenario B*. Buffer distribution Sb and in an area without spatial constraints.
- *Scenario C*. Buffer distribution Nb and in an area without spatial constraints.
- *Scenario D*. Buffer distribution Nb in an area with spatial constraints, and with/without energy restrictions.
- *Scenario E*. Buffer distribution Nb in an area with spatial constraints, without energy restrictions, and three different vertex arrival times.

The objective of Scenarios *A*, *B*, and *C* is to analyze the effects of different buffer distributions. Meanwhile, Scenario *D* only considers the Nb buffer distribution, with energy restrictions (see Subsection V-C7) and using a map area (see Fig. 4) to represent a more realistic scenario. Finally, the objective of Scenario *E* is to measure the impact of the different vertices arrival times to the network (see Subsection V-C8) when using the same realistic scenario as Scenario *D* with unrestricted energy.

D. STATISTICAL ANALYSIS

The experiments consisted of 100 executions for each mobility model and each routing algorithm, changing the initial random seed at each execution. We performed a statistical test analysis of the results. First, we used the Lilliefors test [37] to determine whether or not the experimental data have a normal distribution. If the outcome of this test indicates that data was not normally distributed, then we applied the Wilcoxon rank sum test [38]; otherwise, we applied the student's t-test to

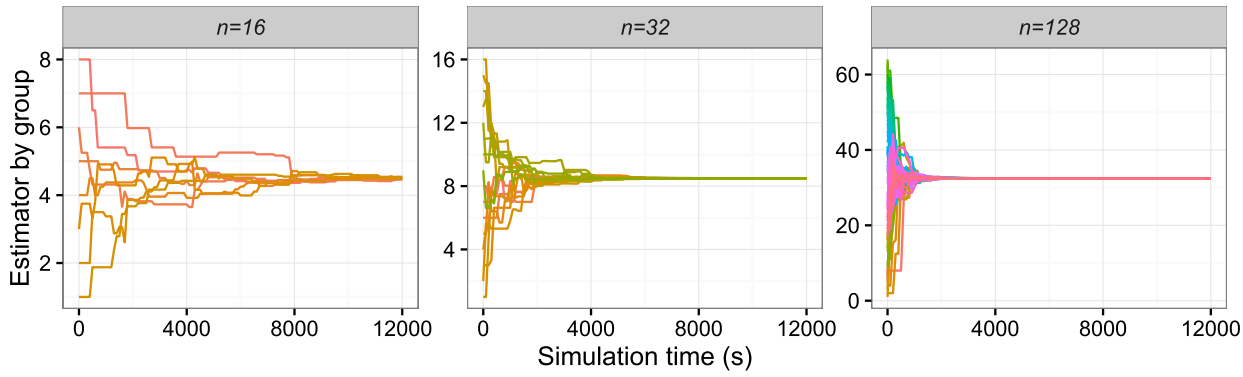


FIGURE 5. Convergence time of the estimator by using the random waypoint mobility model for Scenario *C* for various values of *n*.

determine if there exists a statistically significant difference among the outputs.

E. COMPUTING ENVIRONMENT

Our experiments run on a Dell PowerEdge R900 with 12 Intel Xeon processors running at 2.13 GHz, 32 GB of RAM, and the Linux CentOS 6 operating system. We coded our implementation in Java 1.8 and compiled it with JDK 8.

F. PARAMETERS OF THE ALGORITHMS

We compared EDEN algorithm against two well-known opportunistic routing algorithms, epidemic flooding [19] and Spray and Wait [20]. We did not modify any parameter of epidemic flooding as implemented in the ONE simulator. On the contrary, for the SnW algorithm, we set the initial number of copies, L, equal to 15% of n.

Additionally, to compare the performances of all the previous algorithms against an ideal algorithm, we generated the optimal number of copies ($\sum \frac{k_v}{n}$ for all v) of each message m_v and randomly distributed them in the vertices. We called this "algorithm" *Ideal*. Finally, the two versions of the EDEN algorithm used $\epsilon = 0.5$ and $historic.size = 50$ as initial parameters.

VI. RESULTS

This section presents the results of the simulation for all the scenarios. First, it shows the average convergence time of the estimator in EDEN for Scenario C, with different numbers of vertices, and the random waypoint mobility model. Then, it presents the results of the metrics RUF, MUF, λ, and convergence time of the network for all the algorithms in Scenarios A, B, and C. After that, it presents the results of λ and the energy consumption time of the vertices for all the algorithms in Scenario D. Finally, it shows the results of NAND and the convergence time of the network for Scenario E.

A. CONVERGENCE OF THE ESTIMATOR

Fig. 5 shows the average convergence time of the estimator for each group of buffer distribution for Scenario C and for different values of n. These values are always lower than 8,000 seconds (approximately 2 hours). The greater the number of vertices, the shorter the estimator's convergence time. The plots of Fig. 5 do not significantly change by using Scenarios A and B.

B. SIMULATION FOR SCENARIO A

This subsection presents the simulation results for Scenario A. Fig. 6 shows the RUF for various algorithms and values of n. The RUF for epidemic flooding is very low for all n compared to the other algorithms, in general, lower than 0.25. Despite the RUF of EDEN and SnW being close to one, SnW shows the best performance.

Remark 4: In the following figures, each notched boxplot represents the dispersion of the data obtained from the results of 100 executions. The bottom and top of the box are the first and third quartiles, respectively. The line inside the box is the second quartile (the median). The ends of the whiskers represent the lowest (highest) datum within 1.5 IQR of the third (first) quartile. Any data not included between the whiskers are considered outliers.

Fig. 7 shows the MUF for various algorithms and values of n. SnW has the worst performance for this metric, lower than 0.4, since this algorithm gives priority to the uniformity of messages rather than to memory utilization. In general, the MUF values for the epidemic flooding and EDEN algorithms are greater than 0.94. Particularly, epidemic flooding always obtains approximately a value of one, i.e., it practically fills the buffers in all executions for all n, obtaining the best performance for this metric.

Fig. 8 shows the RUF$\times$MUF for various algorithms and values of n. The values of this metric are similar to those of the RUF since most of the values of the MUF are very close to one for all algorithms, except for SnW. For the last algorithm, its product is always lower than 0.38. For simplicity, we replace the plots of MUF and RUF by RUF$\times$MUF in the remainder of this section.

Fig. 9 shows the NAND for the random waypoint mobility model and various algorithms and values of n. For this metric, the lower, the better. Because epidemic flooding replaces old

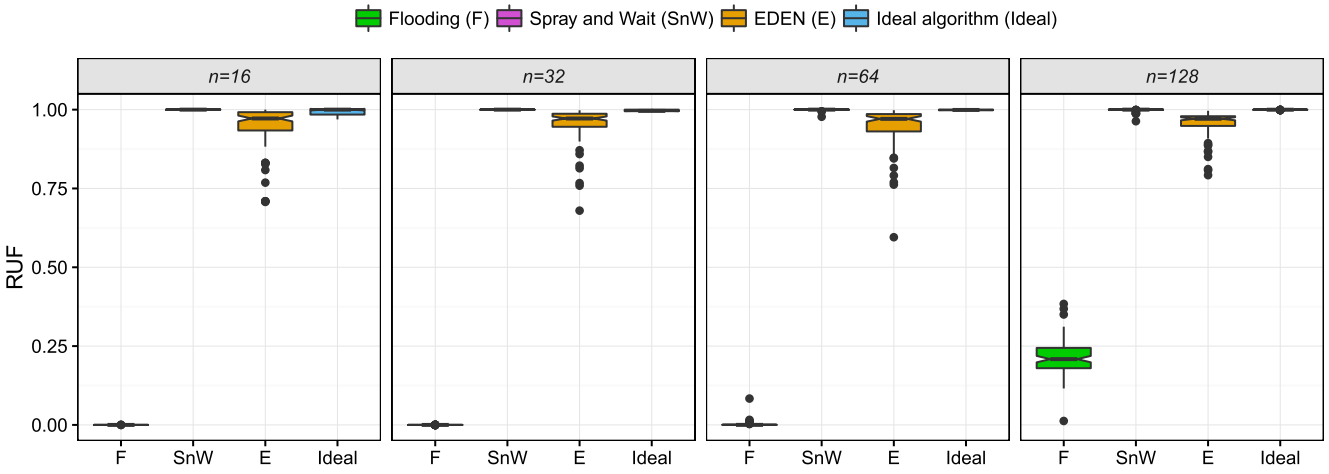


FIGURE 6. RUF for Scenario *A*, using the random waypoint mobility model, for all algorithms and values of *n*.

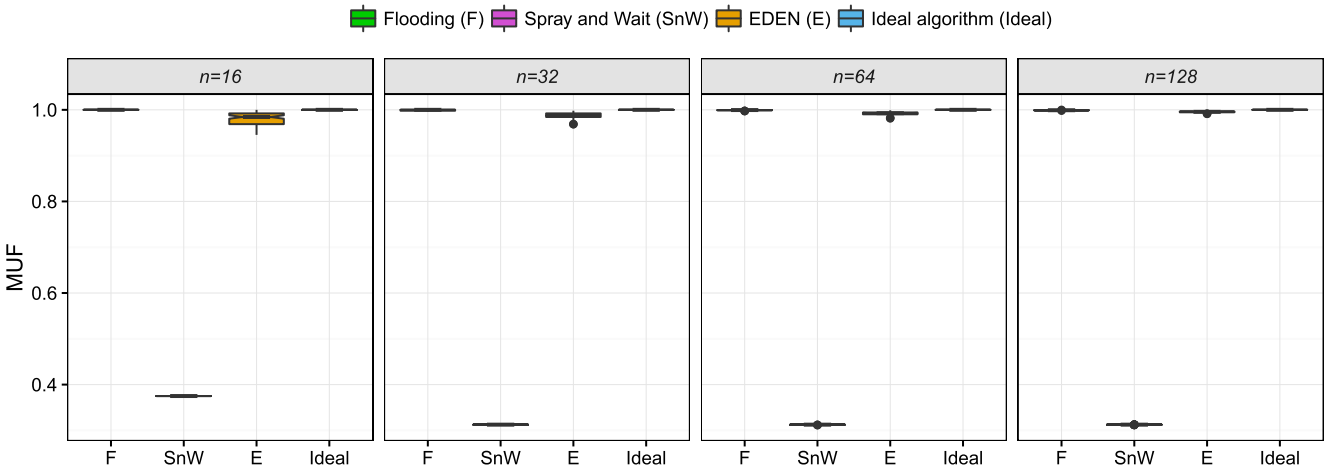


FIGURE 7. MUF for Scenario *A*, using the random waypoint mobility model, for all algorithms and values of *n*.

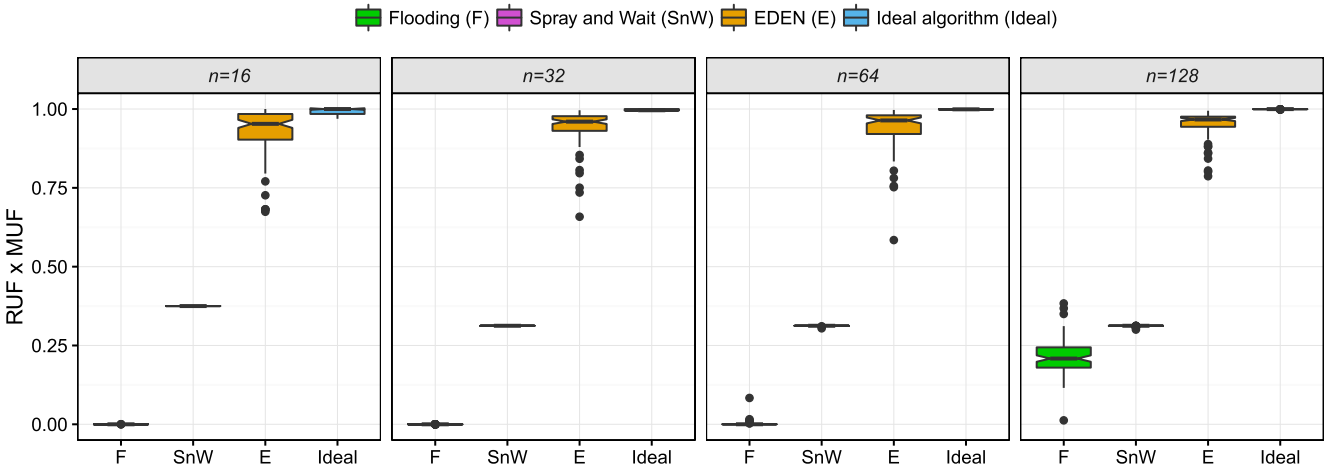


FIGURE 8. RUF×MUF for Scenario *A*, using the random waypoint mobility model, for all algorithms and values of *n*.

messages by new ones in each vertex, it is possible that even after drawing all the vertices from the network, this algorithm fails to obtain at least one copy of each message. Fig. 9 shows this behavior for epidemic flooding when n is equal to 16, 32, and 64, where, even after drawing all the vertices from the network, it cannot obtain Ω_n (i.e., $\lambda/n = 1$). The second

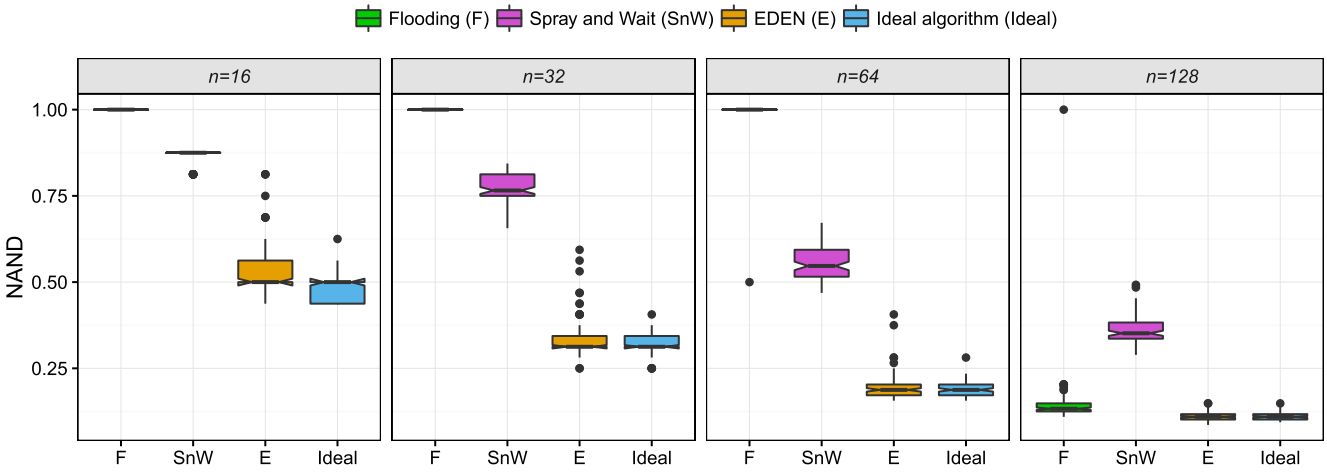


FIGURE 9. NAND for Scenario *A*, using the random waypoint mobility model, for all algorithms and values of *n*.

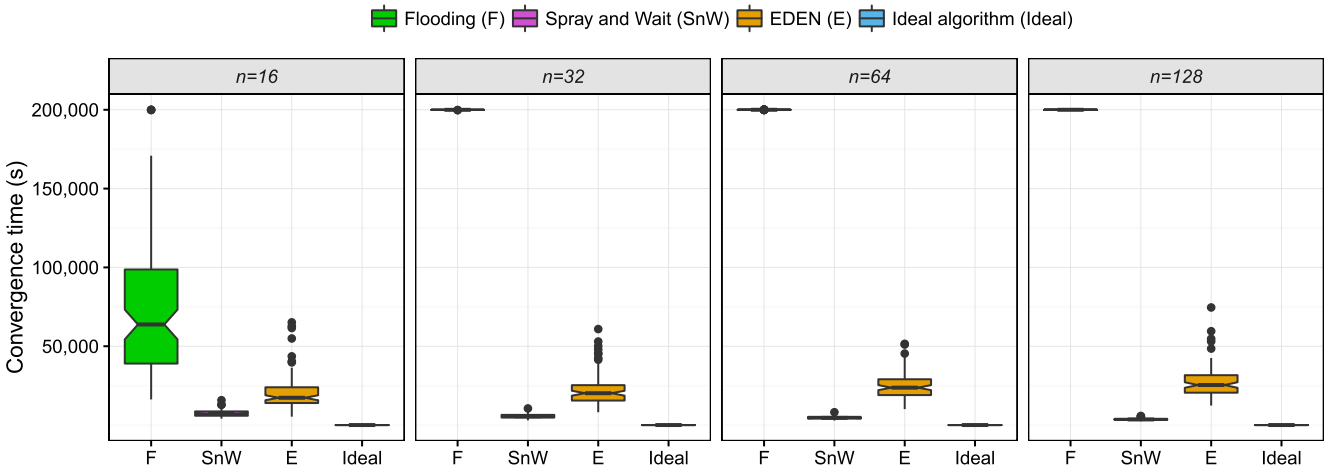


FIGURE 10. Convergence time of the network for Scenario *A*, using the random waypoint mobility model, for all algorithms and values of *n*.

worst algorithm is SnW for all n. The EDEN algorithm has the best performance among all analyzed algorithms. Moreover, there is no significant difference in the values between Ideal and EDEN for $n = 128$ vertices.

Fig. 10 shows the convergence time of the network for various algorithms and values of n. Epidemic flooding yields the worst results. For this algorithm, the network did not even converge at the final simulation time for $n > 16$. Because SnW transmits very few messages compared to the other algorithms, its convergence time is the smallest, on average, less than 7,500 seconds. For EDEN, its convergence time is 23,000 seconds, on average, for all n.

For the convergence time in Scenarios *B*, *C*, and *D* with unrestricted energy scheme, the algorithms keep their rankings which respect each other as in Scenario *A*. If this type of plot included EDEN_{wu}, its performance would be worse than EDEN. For this reason, we omit the corresponding plots for these scenarios in the following subsections.

C. SIMULATION FOR SCENARIO B

This subsection presents the simulation results for Scenario *B* for the random waypoint mobility model. Fig. 11 shows the RUF×MUF for various algorithms and values of n. The product for epidemic flooding is practically zero, the lowest for all algorithms and values of n. Then, SnW has the second worst performance. Its product is lower than 0.8, on average, for all n. Finally, EDEN outperforms EDEN_{wu} for all n, except for 16, in which the opposite occurs. For $n = 16$, this particular behavior is probably due to the effect of rounding on l_v, since its optimal value is 3.75 (see Table 2) and our protocol demands an integer number of copies. When executing EDEN_{wu}, not all of the vertices start the transmission of their messages at the same time. Each vertex starts its message transmissions immediately after its estimator converges. The first vertices that transmit do not have a problem to allocate all their messages; on the contrary, the last vertices that start their transmissions rarely can allocate all their messages. This behavior decreases the RUF×MUF.

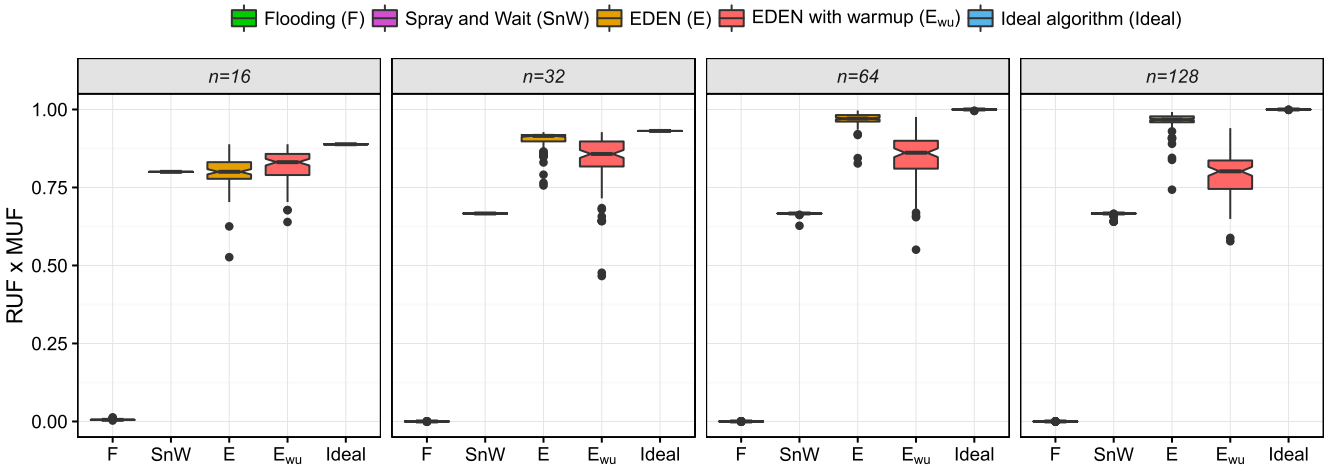


FIGURE 11. RUF×MUF for Scenario *B*, using the random waypoint mobility model, for all algorithms and values of *n*.

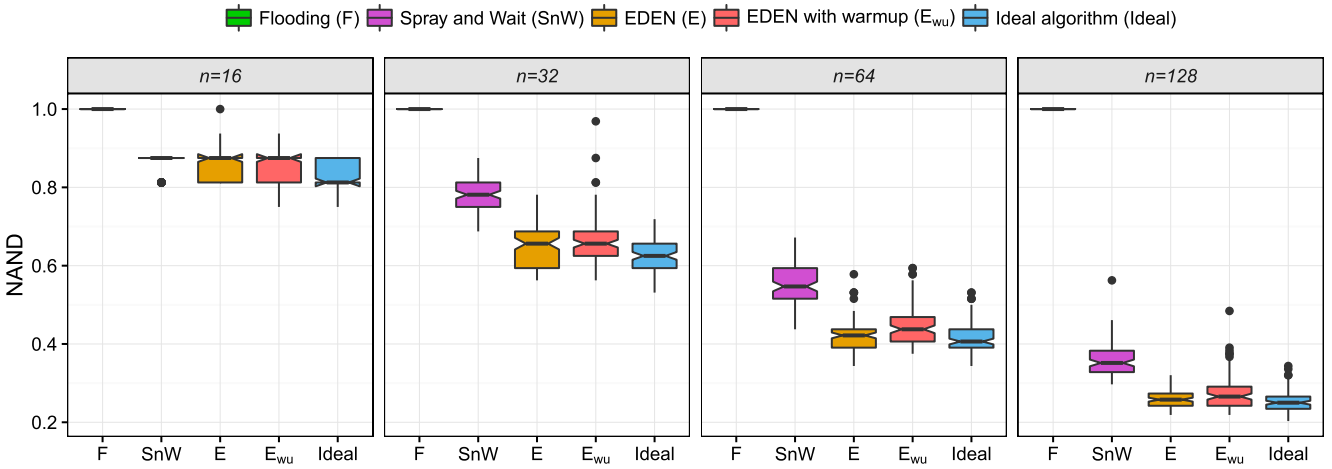


FIGURE 12. NAND for Scenario *B*, using the random waypoint mobility model, for all algorithms and values of *n*.

Fig. 12 shows the NAND for the random waypoint mobility model and various algorithms and values of n. Epidemic flooding obtains a value of one for this metric, being the worst for all algorithms and values of n. SnW obtains the second worst values for all n. Its values are 10% worse than those of EDEN and EDEN_{wu}, on average, for all n, except for $n = 16$, where they are similar. The performance of EDEN is 2% better than that of EDEN_{wu}, on average, for all n, except $n = 16$, in which case they are similar. There is no significant difference in this metric between Ideal and EDEN for $n = 64$ and $n = 128$.

D. SIMULATION FOR SCENARIO C

This subsection presents the simulations for Scenario *C* for the random waypoint mobility model. Fig. 13 shows the RUF×MUF for various algorithms and values of n. The product for epidemic flooding is practically zero, the lowest of all algorithms for all n. SnW obtains the second lowest product for all n. Its product is lower than 0.66, on average, for all n. EDEN is at least 5% better than EDEN_{wu}, on average, for all n.

Fig. 14 shows the NAND for the random waypoint mobility model and for all algorithms and values of n. EDEN is at least 2% better than EDEN_{wu}, on average, for all n. Furthermore, EDEN_{wu} is at least 10% better than SnW, on average, for all n. Finally, epidemic flooding, with a value of one for this metric, is the worst of all algorithms.

E. SIMULATION FOR SCENARIO D

This subsection shows the results for the NAND metric in Scenario *D*, which considers three different energy schemes (see Subsection V-C7).

Fig. 15 shows the NAND for all algorithms and values of n for the unrestricted energy scheme. Fig. 15 keeps practically the same behavior as Fig. 14 (Scenario *C*). Therefore, the descriptions of Figs. 15 and 14 are the same. In the remaining of this subsection, all the plot comparisons of Figs. 16 and 17 are against to those of Fig. 15. Note that SnW and epidemic flooding remain practically with the same values for all energy schemes.

Fig. 16 shows the NAND for all algorithms and values of n for the fully-charged battery scheme. All the plots of Fig. 16

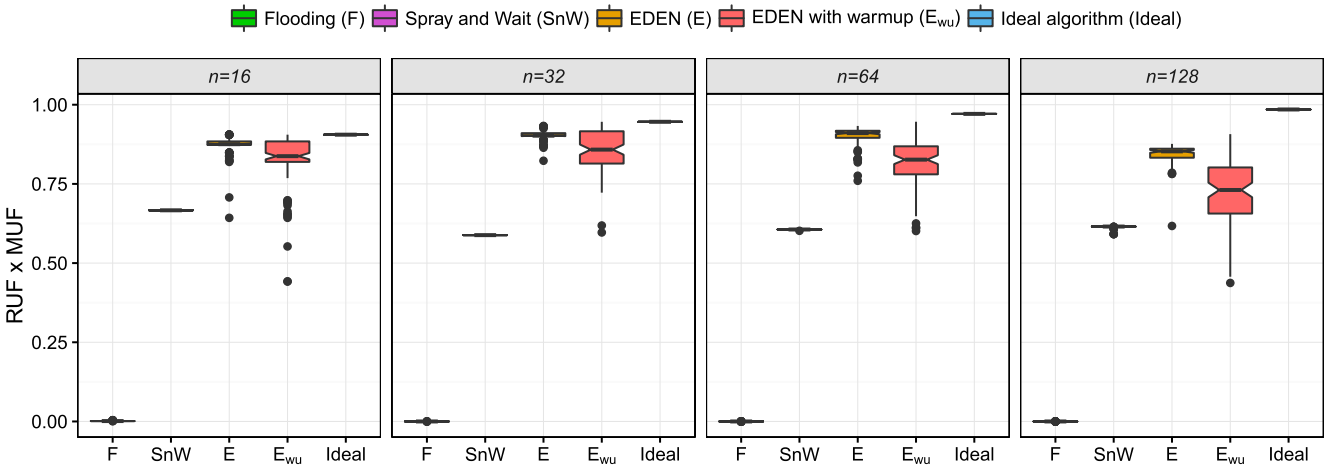


FIGURE 13. RUF×MUF for Scenario *C*, using the random waypoint mobility model, for all algorithms and values of *n*.

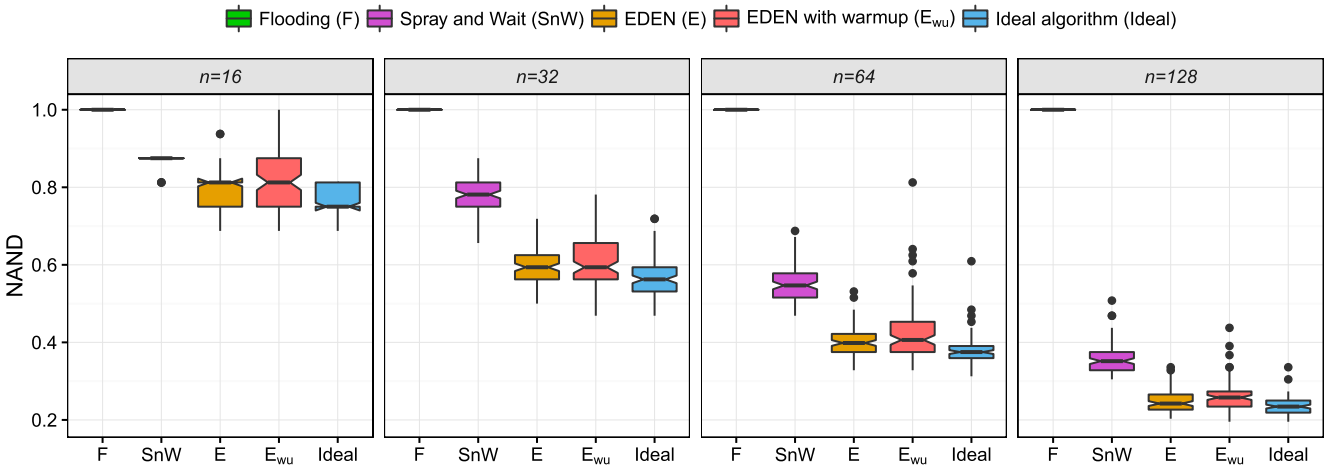


FIGURE 14. NAND for Scenario *C*, using the random waypoint mobility model, for all algorithms and values of *n*.

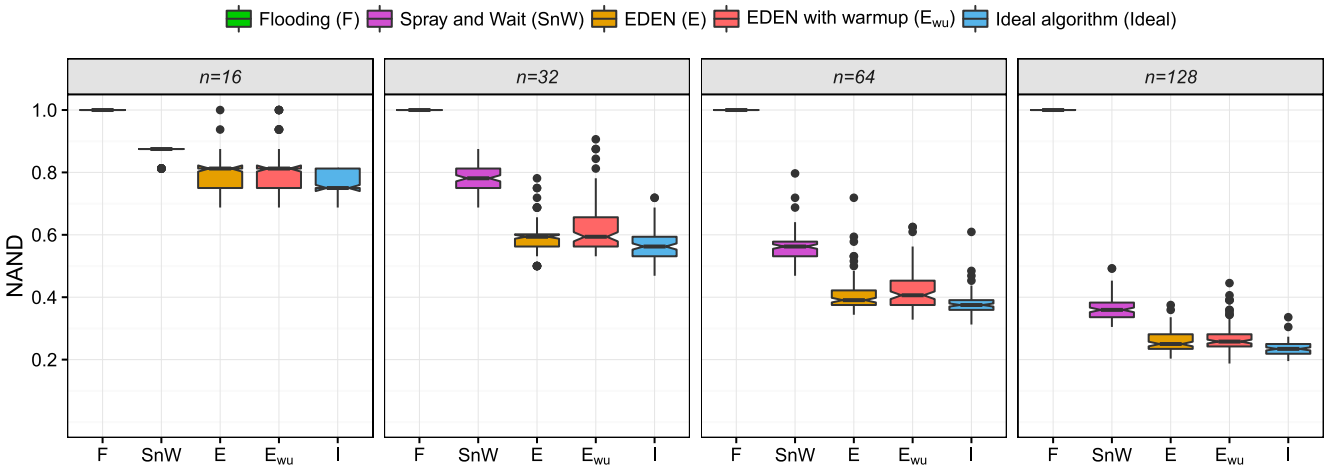


FIGURE 15. NAND for Scenario *D*, with unrestricted energy, for all algorithms and values of *n*.

maintain their values for all n, except for the following cases. First, the performance of EDEN_{wu} decreases 13%, on average, for $n = 32$. Second, the performances of EDEN and EDEN_{wu} decrease 19%, on average, for $n = 16$. Furthermore, note that the performances of these two algorithms are even worse than that of SnW for $n = 16$.

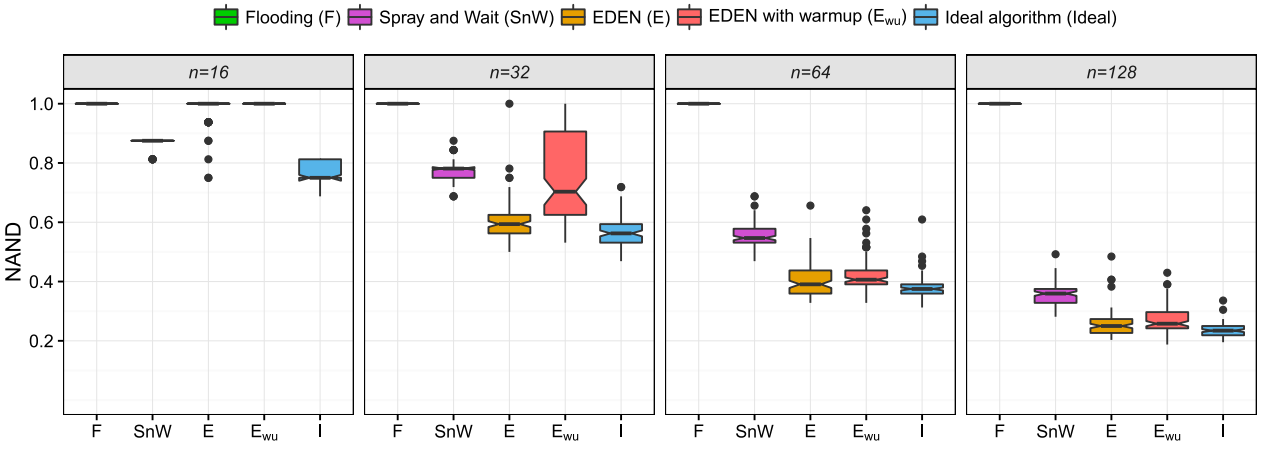


FIGURE 16. NAND for Scenario *D*, with a fully-charged battery, for all algorithms and values of *n*.

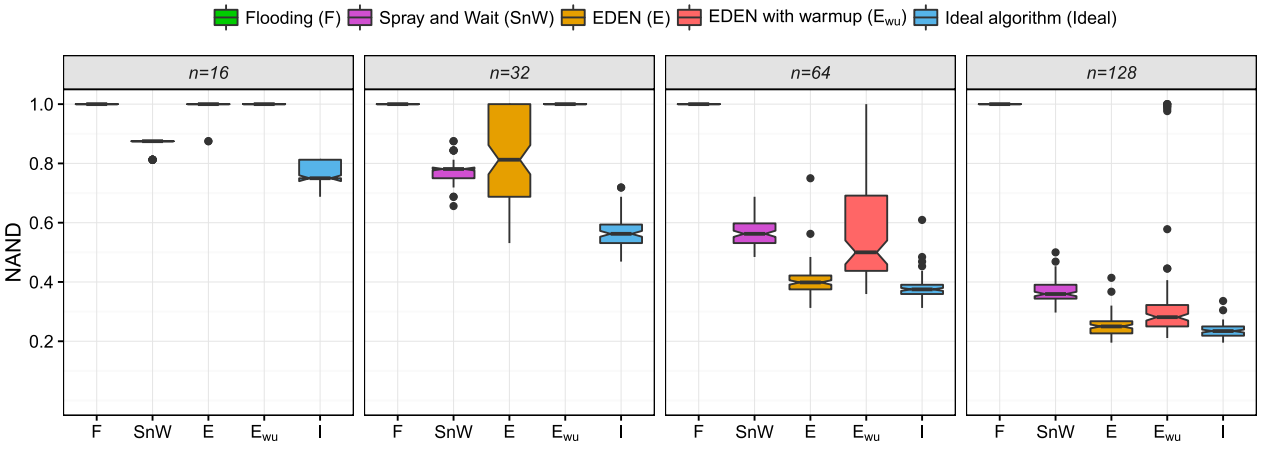


FIGURE 17. NAND for Scenario *D*, with a random-charged battery, for all algorithms and values of *n*.

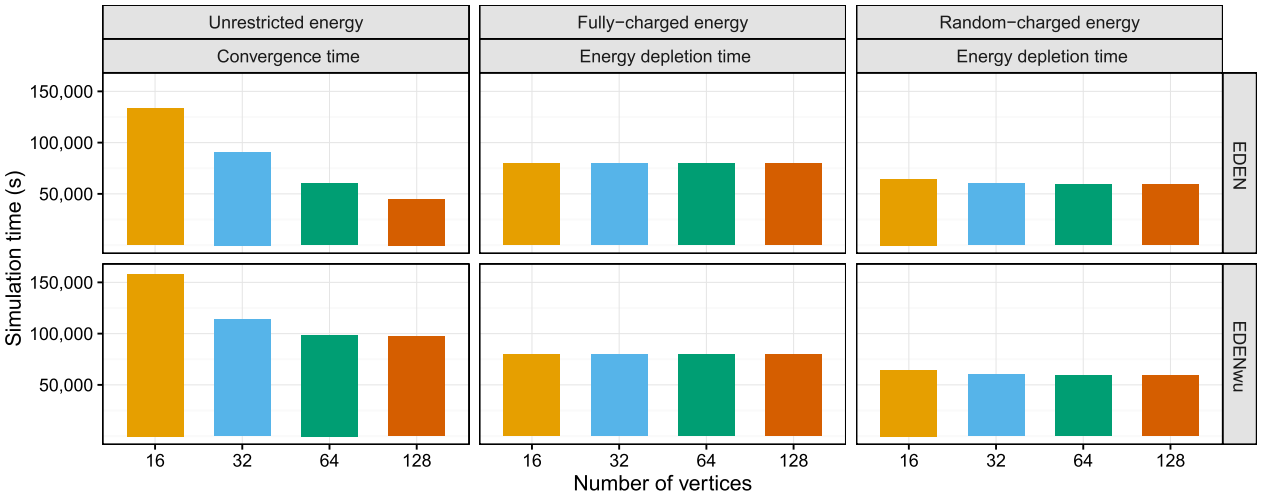


FIGURE 18. Convergence times for the unrestricted energy scheme. Energy depletion times for fully-charged energy, and random-charged energy schemes.

Finally, Fig. 17 shows the NAND for all algorithms and values of n for the third energy scheme. EDEN maintains the same values for $n = 128$ and $n = 64$; meanwhile, it decreases its performance 23% and 20%, on average, for $n = 32$ and $n = 16$, respectively. EDEN_{wu} maintains the same values only for $n = 128$; meanwhile, it decreases its performance

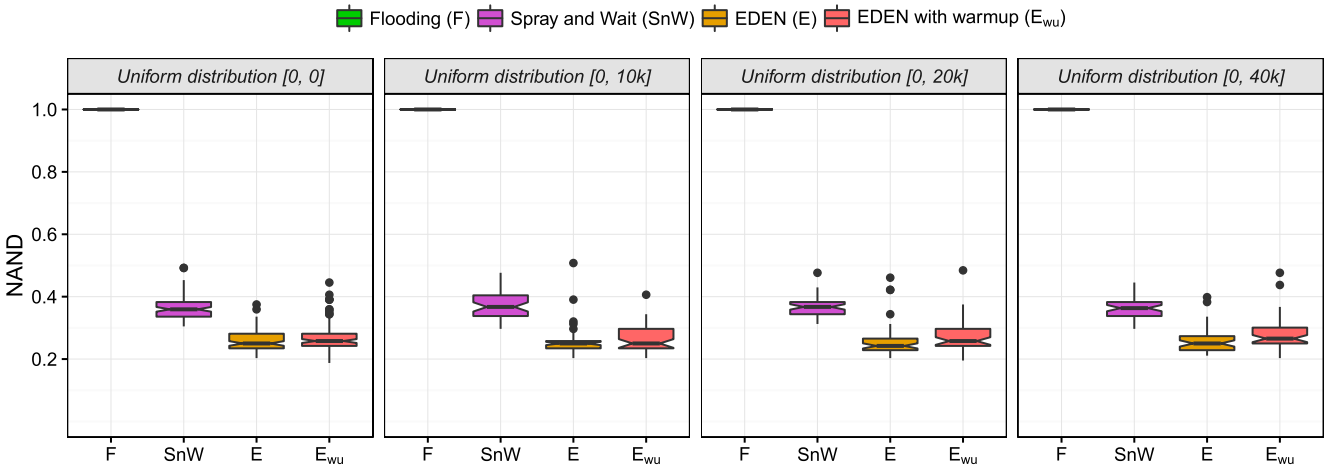


FIGURE 19. NAND for Scenario *E* with different vertex arrival time intervals and $n = 128$.

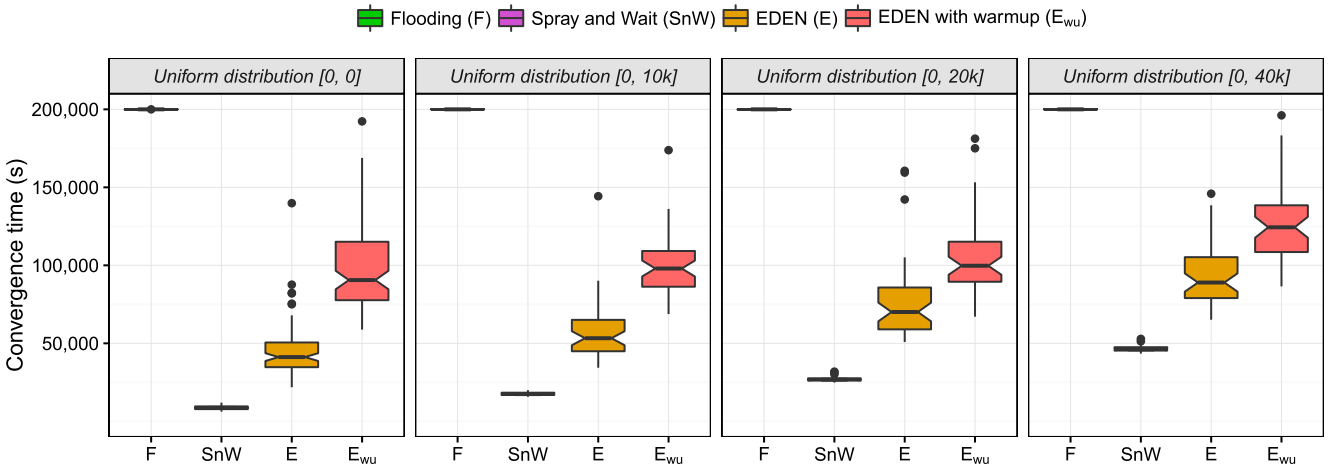


FIGURE 20. Convergence times for Scenario *E* with different vertex arrival time intervals and $n = 128$.

19%, 37%, and 16%, on average, for $n = 64$, $n = 32$, and $n = 16$, respectively.

The above cases show a notorious decrease for EDEN and EDEN_{wu}. The reason of this decrease is because the average convergence times of the network for the unrestricted energy scheme (left column of Fig. 18) are greater than those of the other two energy schemes (middle and right columns of Fig. 18, respectively). Note that the lower the number of vertices, the longer the convergence time. For this reason, the loss in performance for EDEN and EDEN_{wu} is more notorious for small values of n than for high values of n. In particular, EDEN_{wu} is the most affected algorithm because of the delay of its warming-up stage.

F. SIMULATION FOR SCENARIO E

This subsection presents the results for Scenario *E*, which uses the same parameters as Scenario *D* with the unrestricted energy scheme.

Fig. 19 shows the NAND for various algorithms, four uniform distributions, and $n = 128$. For comparison purposes, the first column of this figure includes the results when all the vertices join the network at the beginning of the simulation. Our experimental results indicate that the different arrival times practically do not affect the NAND for any algorithm. For the above reason, this subsection omits the plots for the other values of n.

Fig. 20 shows the convergence time for various algorithms and $n = 128$. In general, SnW, EDEN, and EDEN_{wu} require more time to converge, the wider the arrival range of the vertices, the longer the convergence time of the network. Particularly, EDEN_{wu} is the most affected. We conjecture that the delayed entrance of the vertices would have a more negative impact in scenarios that consider energy constraints.

VII. CONCLUDING REMARKS AND FUTURE WORK

This paper proposes a novel problem called the Maximum Uniform Message Distribution (MUMD). This problem arises from our attempt to mitigate a service failure of the global communication network. To deal with

this problem, we also proposed the EDEN algorithm and a variant of it, EDEN_{wu}, to generate approximate solutions for the MUMD. Additionally, we compared them against two well-known opportunistic routing algorithms, namely SnW and epidemic flooding, and against the performance of an ideal algorithm. We performed numerical simulations to evaluate these algorithms in scenarios with different characteristics.

Note that the SnW and epidemic flooding algorithms were not specifically designed to solve the MUMD. SnW gives priority to the uniformity of messages rather than to memory utilization; on the contrary, epidemic flooding does the opposite. Meanwhile, the objective of EDEN is to maximize both parameters; therefore, it obtains the best performance for metrics RUF×MUF and NAND for most scenarios.

The EDEN_{wu} algorithm is simpler than EDEN because it avoids the copy promotion and the copy depuration procedures. For this reason, it requires fewer message transmissions and a longer time to achieve the convergence time of the network than EDEN. However, in general, the performance of EDEN_{wu} is at most 2% worse than EDEN for NAND, in most scenarios. Furthermore, EDEN and EDEN_{wu} obtain values at most 2% and 4% worse than an Ideal algorithm, respectively, for NAND in Scenarios *A*, *B*, and *C*.

In general, for the two variants of EDEN, our experimental results show that convergence times of both the estimator and the network depend on the number of contacts among the vertices of the network: the greater the number of vertices, the shorter the convergence time.

In Scenario *D*, in general, the performances of EDEN and EDEN_{wu} deteriorate as the battery capacity decreases. The smaller the number of vertices, the higher the performance degradation of these two algorithms. In spite of this behavior, EDEN is the best, under the NAND criterion, for most simulation results for this scenario. SnW and epidemic flooding keep practically the same performance for all energy schemes.

Notice that Scenario *D* with random-charged energy scheme seems similar to a situation where all vertices gradually leave the network. The depletion of a battery results in the withdrawal of a vertex. The results in this Scenario imply that the earlier the removal of the vertices, the worse the performance of our proposed algorithms.

In Scenario *E*, the performances of the algorithms, under the NAND criterion, are practically the same as the ones in which all the vertices arrive at the beginning of the simulation; however, the different arrival times drastically affect the convergence times of the network.

For future work, it would be interesting to study mechanisms to allow EDEN to save energy, mainly because energy supply is a major concern in disaster scenarios [39]. Another interesting research direction is the evaluation of EDEN with a mobility model that realistically represents the movements in a disaster area scenario (e.g., [40]).

REFERENCES

[1] Google. *Google Crisis Response*. Accessed: Dec. 28, 2017. [Online]. Available: http://www.google.com/crisisresponse

[2] T. Hossmann, F. Legendre, P. Carta, P. Gunningberg, and C. Rohner, "Twitter in disaster mode: Opportunistic communication and distribution of sensor data in emergencies," in *Proc. ExtremeCom*. Manaus, Brazil, Sep. 2011=, pp. 1–6.

[3] OpenGarden. (2017). *Fire Chat*. Accessed: Dec. 28, 2017. [Online]. Available: https://www.opengarden.com/firechat.html

[4] S. Saha, Sushovan, A. Sheldekar, J. C. Rijo, A. Mukherjee, and S. Nandi, "Post disaster management using delay tolerant network," in *Recent Trends in Wireless and Mobile Networks* (Communications in Computer and Information Science), vol. 162. 2011, pp. 170–184, doi: 10.1007/978-3-642-21937-5_16.

[5] D. G. Reina, M. Askalani, S. L. Toral, F. Barrero, E. Asimakopoulou, and N. Bessis, "A survey on multihop ad hoc networks for disaster response scenarios," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 10, pp. 1–16, Jan. 2015, doi: 10.1155/2015/647037.

[6] K. Miranda, A. Molinaro, and T. Razafindralambo, "A survey on rapidly deployable solutions for post-disaster networks," *IEEE Commun. Mag.*, vol. 54, no. 4, pp. 117–123, Apr. 2016, doi: 10.1109/MCOM.2016.7452275.

[7] E. Rosas *et al.*, "Survey on simulation for mobile ad-hoc communication for disaster scenarios," *J. Comput. Sci. Technol.*, vol. 31, no. 2, pp. 326–349, 2016, doi: 10.1007/s11390-016-1630-x.

[8] H. Ghafghazi, A. Elmougy, H. T. Mouftah, and C. Adams, "Location-aware authorization scheme for emergency response," *IEEE Access*, vol. 4, pp. 4590–4608, 2016, doi: 10.1109/ACCESS.2016.2601442.

[9] L. Lilien, Z. H. Kamal, V. Bhuse, and A. Gupta, "Opportunistic networks: The concept and research challenges in privacy and security," in *Proc. WSPWN*, Miami, FL, USA, Mar. 2006, pp. 134–147.

[10] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks," *IEEE Commun. Mag.*, vol. 44, no. 11, pp. 134–141, Nov. 2006.

[11] L. Lilien, Z. H. Kamal, V. Bhuse, and A. Gupta, "The concept of opportunistic networks and their research challenges in privacy and security," in *Mobile and Wireless Network Security and Privacy*. Boston, MA, USA: Springer, 2007, ch. 5, pp. 85–117.

[12] C.-M. Huang, K.-C. Lan, and C.-Z. Tsai, "A survey of opportunistic networks," in *Proc. AINAW*, Ginowan, Japan, Mar. 2008, pp. 1672–1677.

[13] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wireless Netw.*, vol. 8, nos. 2–3, pp. 153–167, 2002, doi: 10.1023/A:1013763825347.

[14] O. M. Al-Kofahi and A. El Kamal, "Survivability strategies in multihop wireless networks," *IEEE Wireless Commun.*, vol. 17, no. 5, pp. 71–80, Oct. 2010, doi: 10.1109/MWC.2010.5601961.

[15] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," in *Service Assurance with Partial and Intermittent Resources* (Lecture Notes in Computer Science), vol. 3126, P. Dini, P. Lorenz, and J. N. de Souza, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 239–254. [Online]. Available: https://www.springer.com/us/book/9783540225676 and https://link.springer.com/chapter/10.1007/978-3-540-27767-5_24, doi: 10.1007/978-3-540-27767-5_24.

[16] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–11.

[17] J. LeBrun, C.-N. Chuah, D. Ghosal, and M. Zhang, "Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks," in *Proc. IEEE 61st Veh. Technol. Conf. (VTC-Spring)*, vol. 4. May/Jun. 2005, pp. 2289–2293.

[18] A. Martín-Campillo, J. Crowcroft, E. Yoneki, and R. Martí, "Evaluating opportunistic networks in disaster scenarios," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 870–880, 2013, doi: 10.1016/j.jnca.2012.11.001.

[19] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Duke Univ., Durham, NC, USA, Tech. Rep. CS-200006, 2000.

[20] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," in *Proc. WDTN*. Philadelphia, PA, USA, Aug. 2005, pp. 252–259.

[21] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility," in *Proc. PerCom*, White Plains, NY, USA, Mar. 2007, pp. 79–85.

[22] K. Wang, Y. Shao, L. Shu, Y. Sun, and L. He, "An improved spray and wait algorithm based on rvns in delay tolerant mobile sensor networks," in *Proc. ICC*. London, U.K., Jun. 2015, pp. 3552–3556.
[23] I. Caragiannis, C. Galdi, and C. Kaklamanis, "Basic computations in wireless networks," in *Proc. ISAAC*. Sanya, China, Dec. 2005, pp. 533–542.
[24] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient routing in intermittently connected mobile networks: The multiple-copy case," *IEEE/ACM Trans. Netw.*, vol. 16, no. 1, pp. 77–90, Feb. 2008, doi: 10.1109/TNET.2007.897964.
[25] A. Clementi, R. Silvestri, and L. Trevisan, "Information spreading in dynamic graphs," *Distrib. Comput.*, vol. 28, no. 1, pp. 55–73, Feb. 2015, doi: 10.1007/s00446-014-0219-2.
[26] F. Kuhn and R. Oshman, "Dynamic networks: Models and algorithms," *SIGACT News*, vol. 42, no. 1, pp. 82–96, Mar. 2011, doi: 10.1145/1959045.1959064.
[27] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Regular expressions and languages," in *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. 2000, ch. 3, pp. 83–85.
[28] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Syst. Control Lett.*, vol. 53, no. 1, pp. 65–78, 2004, doi: 10.1016/j.sysconle.2004.02.022.
[29] S. Rajagopalan and D. Shah, "Distributed averaging in dynamic networks," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 4, pp. 845–854, Aug. 2011, doi: 10.1109/JSTSP.2011.2114635.
[30] W. Stadje, "The collector's problem with group drawings," *Adv. Appl. Probab.*, vol. 22, no. 4, pp. 866–882, 1990, doi: 10.2307/1427566.
[31] M. Ferrante and M. Saltalamacchia. (2014). *The Coupon Collector's Problem*. Accessed: Dec. 28, 2017. [Online]. Available: http://www.mat.uab.cat/matmat/PDFv2014/v2014n02.pdf
[32] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," in *Proc. Simutools*, Rome, Italy, 2009, pp. 55:1–55:10.
[33] D. R. Silva, A. Costa, and J. Macedo, "Energy impact analysis on DTN routing protocols," in *Proc. ExtremeCom*, Zurich, Switzerland, Mar. 2012, pp. 1–6.
[34] M. Ababou, R. El Kouch, M. Bellafkih, and N. Ababou, "Energy efficient and effect of mobility on ACDTN routing protocol based on ant colony," in *Proc. ICEIT*. Marrakech, Morocco, Mar. 2015, pp. 335–340.
[35] C. O. Rolim *et al.*, "Situation awareness and computational intelligence in opportunistic networks to support the data transmission of urban sensing applications," *Comput. Netw.*, vol. 111, pp. 55–70, Dec. 2016, doi: 10.1016/j.comnet.2016.07.014.
[36] C. C. Sobin, V. Raychoudhury, and S. Saha, "An energy-efficient and buffer-aware routing protocol for opportunistic smart traffic management," in *Proc. ICDCN*. Hyderabad, India, 2017, pp. 25:1–25:8.
[37] H. W. Lilliefors, "On the kolmogorov-smirnov test for normality with mean and variance unknown," *J. Amer. Statist. Assoc.*, vol. 62, no. 318, pp. 399–402, 1967, doi: 10.1080/01621459.1967.10482916.
[38] W. Conover, *Practical Nonparametric Statistics* (Wiley Series in Probability and Statistics: Applied Probability and Statistics). Washington, DC, USA: Wiley, 1999.
[39] J. C. Araneda, H. Rudnick, S. Mocarquer, and P. Miquel, "Lessons from the 2010 chilean earthquake and its impact on electricity supply," in *Proc. POWERCON*. Hangzhou, China, Oct. 2010, pp. 1–7.
[40] N. Aschenbruck, E. Gerhards-Padilla, and P. Martini, "Modeling mobility in disaster area scenarios," *Perform. Eval.*, vol. 66, no. 12, pp. 773–790, 2009, doi: 10.1016/j.peva.2009.07.009.

HÉCTOR ZATARAIN-ACEVES received the B.E. degree in computer systems engineering from the Culiacan Institute of Technology, Culiacán, Mexico, in 2008, and the M.Sc. degree in computer science from CICESE Research Center, Ensenada, Mexico, in 2011, where he is currently pursuing the Ph.D. degree.

His research interests include algorithm design and analysis, opportunistic networks, game theory, and evolutionary computation.

JOSÉ ALBERTO FERNÁNDEZ-ZEPEDA received the B.E. and M.Sc. degrees from the National Autonomous University of Mexico in 1991 and 1994, respectively, and the Ph.D. degree from Louisiana State University in 1999. Since 2000, he has been an Associate Professor with the Department of Computer Science, CICESE Research Center, Mexico.

His research interests include the analysis and design of parallel and distributed algorithms and software process improvement in small organizations.

CARLOS A. BRIZUELA received the B.S. degree from the Tijuana Institute of Technology and the M.Sc. degree in electronics and telecommunications from CICESE in 1994, and the Ph.D. degree from the Kyoto Institute of Technology in Information and Production Sciences in 2001. He is currently an Associate Professor with the Algorithms and Biocomputing Laboratory, Computer Sciences Department, CICESE Research Center.

His research interests include the analysis and design of optimization and machine learning algorithms for applications in structural bioinformatics and combinatorial optimization in engineering.