RSock: A Resilient Routing Protocol for Mobile Fog/Edge Networks

Ala Altaweel*, Chen Yang*, Radu Stoleru, Suman Bhunia, Mohammad Sagor, Maxwell Maurice, Roger Blalock

Abstract

Fog/Edge networks (FENs) refer to wireless networks in which a "close-to-end-user cloud" is established amongst handheld devices to offer communication, computation, and storage services. In this paper, we focus on FENs that are formed by on-body devices of first responders and leveraged (as mission-critical deployable networks) in disaster-response scenarios. Designing an efficient routing protocol for such FENs is challenging since they exhibit dynamic connectivity characteristics and unreliable wireless links due to the mobility of devices, wireless obstacles, etc. Moreover, FENs applications in disaster-response scenarios distribute their processing/storage tasks to local nodes and accordingly have diverse requirements in terms of packet delivery's rate and delay (PDR and PDD). On the other hand, most of the state-of-the-art routing protocols are not suitable for FENs since their target networks have static connectivity characteristics. Furthermore, these protocols provide poor support for seamless mobility communication and for exploiting all devices' wireless interfaces as well as their initial/subsequent deployment(s) require pushing changes to the TCP/IP stack. In this paper, we present Resilient Socket (RSock), a limited-replication-based routing protocol that decides whether to use packet replication and by how much based on the mission-critical deployable FENs connectivity conditions.

*Ala Altaweel and Chen Yang contributed equally to this work.

**Ala Altaweel is with University of Sharjah. Chen Yang, Radu Stoleru, and Mohammad Sagor, are with Texas A&M University. Suman Bhunia is with Miami University of Ohio, Maxwell Maurice and Roger Blalock are with the National Institute of Standards and Technology.

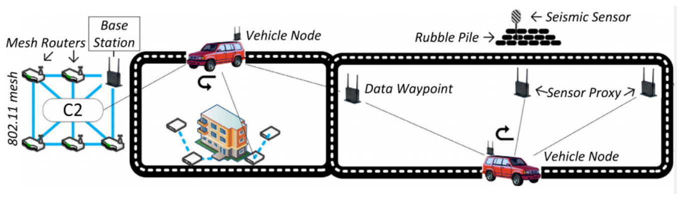
RSock is an identity-based routing protocol that exploits all wireless interfaces to route/deliver packets for IP-address/interface agnostic applications. We design RSock as an easy-to deploy-and-evolve protocol and we demonstrate its feasibility through a reliable full-system implementation for Linux and Android platforms. Real-world experiments show that RSock performs well in disaster-response scenarios.

1. Introduction

Fog/edge networks (FENs) aim to establish a wireless bubble (Wi-Fi and/or LTE) amongst end-users' handheld devices (i.e., at the edge of the network) to enhance their communication, computation and storage capabilities. FENs aim to reduce the number of tasks that are transferred to the cloud while supporting nodes mobility, location awareness, and efficient handling of real-time tasks. In this paper, we focus on a particular type of Fog/Edge networks (FENs). Specifically, the mission-critical deployable FENs that are formed by the on-body devices of first responders in disaster-response scenarios [1]. In such scenarios, a first responders team, who are equipped with mobile devices, aim to accomplish a wide-area search task after a disaster (e.g., earthquake, hurricanes) to look for victims, as shown in Figure 1a. The team might leverage a manpack that is equipped with an eNodeB and Wi-Fi hot-spots (as shown in Figure 1b) to establish communications amongst their handheld devices and to the cloud via the Command and Control center (C2), as shown in Figure 1c. Typically, C2s in such networks are reachable via the vehicular Delay Tolerant Networks (DTN)'s nodes.

Previous research [2] have shown that the connectivity amongst FENs' devices has diverse and dynamic characteristics. We also observed and validated that conclusion during real-world training scenarios for first responders [3]. Factors such as lack of infrastructure networks, wireless obstacles (e.g., metals, concrete blocks, etc.), mobility of devices (e.g., people, ambulances, etc.) results in unreliable wireless links. Accordingly, the first responders might

(a) (b)



(c)

Figure 1: a) First responders team during a disaster with an established mission-critical deployable FEN. b) A manpack with Wi-Fi and LTE hot-spots for first responders. c) A schematic of a disaster-response system deployment.

form networks with time-varying topology despite the fact that they might be in some cases totally disconnected. From another perspective, the applications of mission-critical deployable FENs like the Android Team Awareness Kit (ATAK) [4] [5] that is used by first responders for navigation, situational awareness, and data sharing, as well as other applications for storage file system and chatting [6, 7, 8, 9] have diverse requirements in terms of packet delivery rate and/or delay (PDR, PDD). However, most of the designed/implemented state-of-the-art routing protocols [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22] perform poorly in all FENs since they are designed with a specific type of connectivity in mind (well/intermittently connected). Furthermore, the aforementioned protocols do not support seamless endpoint mobility when devices' IP addresses change, do not exploit different devices' wireless interfaces, as well as they are only simulated or have a prototype-level (i.e., proof-of-concept) implementation. It's not even clear how to deploy these protocols without pushing changes to TCP/IP stacks that typically require OS level patches (i.e., rooted mobile devices). This coupling of the routing protocols implementation to the OS limits their deployment velocity and their iteration velocity of even simple changes. Hence, the design and full-system implementation of an easy-to-deploy and easy-to-evolve routing protocol that support seamless mobile-to-mobile communications, exploit all devices' wireless interfaces, and is able to adapt itself in uncertain mission-critical deployable FENs environments and perform well in the entire range of connectivity is still an open research problem.

In order to address the aforementioned research challenges, we propose the design of Resilient Socket (RSock), which is an enhanced version of the Hybrid Routing Protocol (HRP) [2]. In RSock, we investigated the benefit of packet replication in terms of packet delay reduction, as it is the key to decide when and how much replication should be used. The performance of HRP has been only evaluated through extensive simulations in the ONE simulator[2, 23]. However, HRP has not been integrated with other mission-critical deployable FENs systems, fully implemented for different platforms, deployed into real-world devices, nor rigorously evaluated during real-world scenarios. The contributions of this paper are as follows:

- We demonstrate the impact of path delay correlation on the benefit of packet replication in RSock, and propose a novel model for estimating inter-contact time without loosing the correlation information (Section 3.2). The proposed model allows for accurate analysis of packet replication and the design for efficient forwarding strategies.

- We propose a novel regret-minimization based algorithm which dynamically and adaptively decides the appropriate amount of packet replication given any mission-critical deployable FEN environment (Section 3.3).

- We design RSock which adopts a novel forwarding metric called Additional Contact Rate (ACR) and consists of four main modules (Section 3.4).

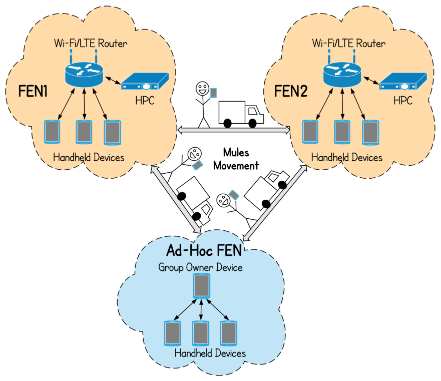
- We demonstrate the feasibility of RSock through a full-system implementation as a user-space service for both Linux and Android platforms to facilitate its deployment and future evolvements.

- We integrate RSock with EdgeKeeper [24, 25], a novel naming and coordination service for FENs, in order to enable identity-based routing and mobile-to-mobile communications via any available wireless interface.
- We integrate RSock with R-Drive [7, 8], a novel resilient data storage and sharing disaster-response application that is proposed for Android platforms to support chatting services for disaster-response and tactical scenarios.
- We extensively evaluate RSock on two real-world FENs systems in disaster-response events [3] and through extensive simulations (on a real-world wireless testbed) and demonstrate its efficacy by showing that it is able to route packets with high PDR, low PDD, and low overhead (i.e., a competitive performance with up to 3x in terms of delay improvement when compared to the state-of-the-art routing protocols).
- We share the lessons learned from our attempts to leverage the Optimized Link State Routing (OLSR) [17]. We believe that our findings are crucial to the research community.

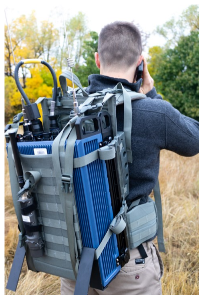
The organization of this paper is as follows. In Section 2, we present the background, our system models, and the motivation of our research. We present the design of RSock in Section 3. In Section 4, we present the full-system implementation and the lessons learned. In Section 5, we illustrate our real-world experiments and the evaluation results. We thoroughly survey the state-of-the-art routing protocols and the Software Defined Networks (SDN) approaches for FENs in Section 6 and finally conclude our paper in Section 7.

2. Background and Motivation

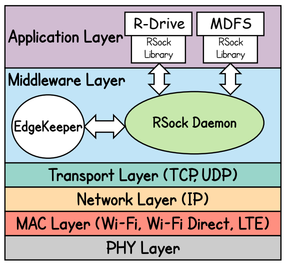
In this section we present the concept of FENs, our system models, and the motivations for our research.



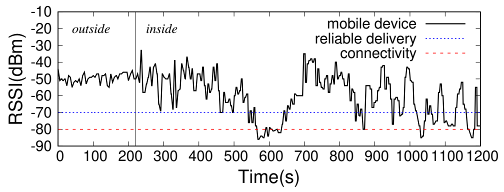
(a)



(b)



(c)



(d)

Figure 2: a) Different mission-critical deployable FENs scenarios. b) The PSCR System. c) DistressNet-NG Software Architecture. d) RSSI of a first responder personnel's device.

2.1. Fog/Edge Networks (FENs)

Traditionally, mobile devices, in typical cloud computing paradigms, offload their computation and storage tasks to remote servers in the cloud. Indeed, performing the aforementioned tasks on the cloud has been efficient due to the assumption that the computing power of the cloud servers outperform the devices' capabilities at the edge of the networks (i.e., the end users' devices). However, nowadays, due to the growing quantity and quality of the generated data at the edge from IoT devices like smartphones, wireless sensors, cameras, etc., the bandwidth and reliability of the communication to the cloud becomes the bottleneck for the cloud-based computing paradigm [26]. Accordingly, many researchers proposed the idea of Fog/edge paradigms [27] [28]. As it often happens with new technologies, a consensus definition of the “Fog/edge” needs some time to be agreed on by the community to mitigate confusion. However, most of the first definitions tend to focus on bringing the computation and storage capabilities close to their data sources. By reducing the distance (i.e., in terms of network topology) and accordingly the time that is required to send data to cloud servers, the performance of data storage, retrieval, and computation operations can be enhanced. In this paper, the term Fog/edge networks (FENs) refer to any group of handheld devices (i.e., smartphones, notebooks, etc.) that are locally connected via their Wi-Fi and/or LTE interfaces. Furthermore, these devices might be able to connect to a local high-performance computing (HPC) node, as shown in Figure 2a. Notice that the environments for the mission-critical deployable FENs in Figure 2a can be in indoor (homes, offices, etc.) or outdoor (in a manpack) with first responder personnel during a disaster, as shown in Figures 1a and 1b.

In such disaster-response scenarios, a FEN is established and utilized by a team of first responders, who are equipped with mobile devices and assigned a wide-area search task in a post-disaster area to discover dangerous zones and search for victims. The first responders can collect a large amount of data, using on body video sensors and cameras in order to analyze it in real time. For such scenario, the data analysis requires significant computational resources and typically it is offloaded to the cloud. However, the communication infrastructure is usually destroyed or debilitated during disasters, which makes leveraging the computation powers of the nearby mobile devices a promising option. Another type of mission-critical deployable FENs is the ad-hoc FEN in which handheld devices like smartphones establish a Peer-to-Peer (P2P) group for communication [29]. That is, one device with the Group Owner roles (AP-like functionalities) establishes a Wi-Fi Direct hot-spot to enable other smartphones to connect to it as clients and communicate amongst each other in an ad-hoc manner, as shown in Figure 2a. Furthermore, there might be mules that physically move between FENs (disconnect from one and join another). These mules might be users with their handheld devices or vehicular nodes, as shown in Figure 2a.

2.2. System Model

In this paper, we have two mission-critical deployable FENs systems with the following details:

1. **The Public Safety Communications Research (PSCR) system [30]**, which consists of LTE and Wi-Fi networks, as presented in Figure 2b. The LTE network provides broadband connectivity for voice, video, and text services. The system also contains an application server with Linux OS that can host our software services that we will discuss below. The system can be powered by a portable generator and has its antennas mounted on top of the cabin of a pickup truck or a table in the field.

2. **Our manpack system**, as shown in Figure 1b. This system consists of LTE and Wi-Fi networks and has an application server that is running on Intel Next Unit of Computing Kit (NUK). We leveraged the NextEPC project [31], an open source implementation of the 4G/5G 3GPP core network that includes the Mobility Management Entity (MME), Serving Gateway (SGW), Packet Data Network Gateway (PGW), Home Subscriber Server (HSS), and Policy and Charging Rules Functions (PCRF) to provide the LTE functionalities. The NUC, Wi-Fi router, and eNodeB are directly powered by inboard batteries inside the manpack that can be carried by public-safety personnel during a disaster.

We illustrate in Figure 2c the software architecture of our project, the DistressNet-NG [1], that we deployed onto the manpack and the PSCR systems. DistressNet-NG aims to enhance the resilience of both public safety mission-critical systems and services in the face of connectivity challenges. In the following paragraph, we describe our software components in more detail.

MDFS [6] [9] is a crucial disaster-response application. That is, a network-disconnection resilient-and-fault-tolerant mobile distributed file storage and processing system. The reason behind proposing MDFS is that most modern applications that off-load data to a remote server have an implicit assumption that the network is always available. However, in FENs, nodes are mobile and might be disconnected from a remote server. Hence, there is a crucial need for a distributed data sharing and processing mechanism that is resilient to network failures and delay tolerant. MDFS is proposed and implemented to store and process data in multiple mobile devices in a distributed manner, incorporating a delay tolerant scenario in FENs. MDFS followed a design structure so that it can be portable for both Linux and android platforms. Moreover, it provides the traditional file system functionalities (e.g., create, retrieve, remove) with standard Unix-like access control list and data-level encryption.

R-Drive [7, 8] is a novel resilient data storage and sharing disaster-response application that is proposed for Android platforms to support chatting services for disaster-response and tactical scenarios. R-Drive provides uni-cast chatting services for text, image, video, and voice messages. R-Drive employs erasure coding and data encryption, ensuring resilient and secure data storage against device failure and adaptively chooses erasure coding parameters to ensure highest data availability with minimal storage cost. For each user, R-Drive lists all other team members such that the user can select the destination(s) for the messages

MDFS requires a naming and coordination service for distributing its storage tasks as well as a fault-tolerant directory-service where the directory structure of the file system is maintained. Similarly, R-Drive requires a naming service for its ID-based communications. That's why we proposed EdgeKeeper [24, 24], a resilient and distributed coordination service for FENs. By following the Apache Hadoop system, EdgeKeeper provides Zookeeper-like services for FENs applications. EdgeKeeper is implemented for Linux and Android platforms and runs as a background service to provide resilient coordination services like network management, application-coordination, etc. An important component of EdgeKeeper is its topology manager, which runs a discovery service where each device periodically pings other devices in the same FEN to determine all device-to-device link qualities. If there are multiple links available between a pair of devices (e.g., LTE and Wi-Fi), the topology manager maintains separate link qualities for these links.

2.3. Motivation: Why RSock?

Diverse Connectivity Characteristics in mission-critical deployable FENs. We observed that the devices, in real-world disaster-response scenarios, often exhibit diverse connectivity characteristics. Factors such as mobility of devices and obstacles cause unreliable wireless links, which lead to a network with rather dynamic connectivity. Moreover, the mobility of first responders makes the network connectivity range from well-connected to almost disconnected. For example, Figure 2d shows the Wi-Fi Received Signal Strength Indicator (RSSI) value of a mobile phone carried by a first responder during a wide area search exercise while connected to the Wi-Fi router in our manpack system that is shown in Figure 1b. Due to node mobility, signal shielding and attenuation, the RSSI value changes and frequently goes below the minimum threshold value that is required for the reliable connection. Furthermore, with the proliferation of wireless capable devices, we believe that mission-critical deployable FENs' users have increasing opportunity to encounter real-world disaster-response scenarios with diverse connectivity in the future. Hence, there is an urgent need for a routing protocol that is able to fulfill and guarantee the diverse requirements of different applications like ATAK, MDFS, R-Drive [4, 6, 7, 8, 9] in terms of PDR and PDD. Hence we design and implement in Section 3.4.2 (Replication Factor Decision) the Replication Factor Decision module of RSock as shown in Figure 6 to decide whether to use packet replication and by how much based on the current connectivity conditions.

Seamless Mobility and Multipath Communication. Mobile devices (smartphones, tablets, notebooks, etc.) and their applications have experienced a huge growth in recent years with higher number nowadays compared to tethered hosts. The overall mobile data traffic is expected to grow to 77 exabytes per month by 2022 [32]. Even though the Internet's TCP/IP stack has accommodated this phenomenal transformation so far, it still provides poor support for seamless endpoint mobility, multipath, and mobile-to-mobile communication. Hence, mobile application developers nowadays have to adopt redundant and fragile application-layer workarounds to address the aforementioned issues. Accordingly, the design and implementation of *location-independent* communication service is of utmost importance to facilitate the development of mission-critical deployable FENs mobile applications. In order to enable the location-independence communication service, we focus on the following goals when we design RSock. First, seamless mobility that aims to allow handheld devices to freely move across network addresses of different mission-critical deployable FENs while relieving the application developer from keeping track of them. Second, since mission-critical deployable FENs mobile devices are equipped with multiple network interfaces such as LTE, Wi-Fi, and Wi-Fi Direct. We aim to leverage these interfaces and seamlessly switch amongst them while relieving the application developer from handling this. Hence, we design RSock and its user-level socket library (i.e., *RSock library*) to enable the location-independent communication service. *RSock library*, as we will present in Section 3.4.1, can be leveraged by any mission-critical deployable FENs mobile application with minimal code changes.

Deploy-ability and Evolve-ability. Many routing protocols [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22] have been proposed in recent years to meet the evolving applications demands in wireless networks. However, they have not seen wide deployment. Middleboxes and firewalls have accidentally become key crucial points in today's wireless networks architecture. Firewalls used to block anything unfamiliar for security reasons and Network Address Translators (NATs) rewrite the transport layer packets' headers, making it hard to allow traffic from new routing protocols without adding explicit support for them. Apart from that, even modifying current routing protocols remains challenging due to its stiffening by middle boxes. As a result, deploying changes to routing protocols are now expected to take long a time (i.e., years). Moreover, as the wireless networks nowadays continue to evolve, there is a crucial need to be able to deploy changes to routing protocols rapidly. For example, the latest releases of OLSR and Prophet protocols [17, 18, 33, 34] require root privileges and OS upgrades to be able to run/update. In order to resolve the aforementioned issues, we design RSock as a user-space protocol that leverages UDP for its data and control packets to facilitate its deployment and enable iterative modifications to occur at application update timescales.

3. Resilient Socket

In this section, we present our preliminaries, basic idea, and how RSock makes routing decisions and its design details.

3.1. Preliminaries

We first introduce three concepts that we use in our design.

Definition 1. *Inter-contact time (ICT): the time duration between two contact events between a pair of nodes in mission-critical deployable FENs.*

Definition 2. *Replication factor r: the total number of data copies created at the source for a given packet.*

If D_r is the random variable for routing delay when the replication factor is r, then:

Definition 3. *Replication gain $\gamma_r = E[D_1]/E[D_r]$.*

Basically, r limits the total number of copies for a packet and γ_r captures the benefit of replication in terms of delay improvement. Notice that γ_r depends on the forwarding strategy of a routing protocol. In particular, if the packet copies are routed along a set of paths P, where each path has delay X_i, then the replication gain is $\gamma_r^P = \min_{i \in P}\{E[X_i]\}/E[\min_{i \in P}\{X_i\}]$. In the following sections, we propose a novel model for ICT estimation that incorporates the correlation among a group of nodes. We will show, by a simple example, that delay correlations among different nodes can significantly affect the benefit of replication. Motivated by this observation, we propose a novel Joint ICT model that quantitatively captures the correlation among different mission-critical deployable FENs nodes.

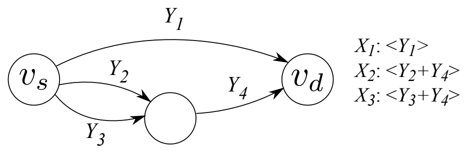


Figure 3: A simple network model where $\{Y_j\}_{j=1}^4$ denotes link delays and $\{X_i\}_{i=1}^3$ denotes path delays.

3.2. Basic Idea

In this section, we present the basic idea that is behind our RSock routing and replication policy. That is, how delay correlation amongst mission-critical deployable FENs nodes adversely affects the benefit of replication. Let's illustrate this via a simple example as shown in Figure 3. v_s sends packets to v_d with replication factor $r = 2$ and it can uses paths $\{X_i\}_{i=1}^3$, consisting of links $\{Y_j\}_{j=1}^4$, which are independently and exponentially distributed. Then, the replication gain from using paths $\{X_i, X_j\}$ is $\gamma_2^{i,j} = \frac{\min\{E[X_1],E[X_2],E[X_3]\}}{E[\min\{X_i,X_j\}]}$ (i.e., when $r = 1$ the shortest path is used). Notice that the delays of X_2 and X_3 are positively correlated due to the common link Y_4, and the correlation becomes stronger when $E[Y_4]$ increases. X_1 is independent of both X_2 and X_3.

When all paths have identical expected delay, correlated paths always have smaller replication gain, and the replication gain decreases when the correlation becomes stronger. We fix $E[X_i] = 60$ for $i = 1, 2, 3$ and vary $E[Y_4]$ while examine the replication gain $\gamma_2^{1,2}$ and $\gamma_2^{2,3}$. As shown in Figure 4a, $\gamma_2^{1,2}$ remains relatively high due to the independence of delay. But $\gamma_2^{2,3}$ quickly approaches 1 when $E[Y_4]$ increases, indicating a drastic decrease of benefit from replication. If v_s assumes independence of path delays, it cannot accurately estimate the replication gain and may make sub-optimal decisions. Indeed, redundancy helps improve system performance, but we should utilize resources as diverse as possible to maximize the benefit [35].

However, if independent paths have inferior performance, it might still be wise to utilize correlated paths. To see this, we fix $E[X_2] = E[X_3] = 60$ and let $E[X_1] = 60 + x$ where $x > 0$. By varying both x and $E[Y_4]$, we observe that there exist threshold x^*, above which $\gamma_2^{2,3}$ becomes larger than $\gamma_2^{1,2}$, as shown in Figure 4. This indicates that better performance is achieved when correlated paths are used. Notice that x^* increases with $E[Y_4]$ (i.e., it is related to the correlation between X_2 and X_3). Hence, in order to accurately estimate the replication gain and make better routing decision, it is important to capture the delay correlation. $\{X_i\}_{i=1}^3$ may correspond to the ICTs between three mobile nodes and the destination. Therefore, it is paramount for v_s to be aware of the potential correlations between the ICTs in order to accurately estimate the replication gain.

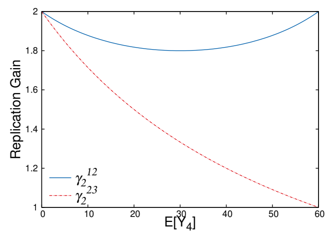
Table 1: Symbols used in this paper

r	Replication factor
γ_r	Replication gain for r
$\boldsymbol{\pi}$, π_r	Probability distribution for replication factors
$\boldsymbol{g}$, g_i	Vector in $\{0,1\}^n$, representing a group of mission-critical deployable FENs nodes
$\lambda_{\boldsymbol{g}}$	Contact rates for a group of mission-critical deployable FENs nodes $\boldsymbol{g}$

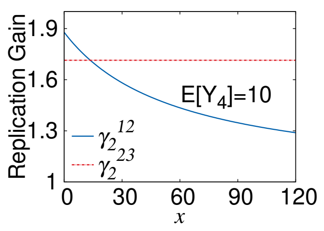
Given the above insights, we propose to jointly capture the ICTs between mission-critical deployable FENs nodes such that the correlation information is preserved, which enables more accurate estimation of the replication gain. Intuitively, the ICTs of two nodes are correlated if the observer often meets both of them at the same time. Consider the contact processes between node v, and nodes v_1 and v_2, as shown in Figure 5 (the right hand side of which shows the contact processes). Notice that node v has met v_1 and v_2 for 7 and 6 times, respectively. Out of these, 4 times it meets them at the same time. In this case, ICTs Y_1 and Y_2 can no longer be assumed as independent.

We use the Marshall and Olkin's Bivariate Exponential Distribution (BVE) [36] to model the 2-D joint ICT distribution for a pair of nodes.

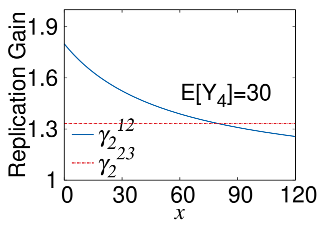
Definition 4. *Marshall and Olkin's Bivariate Exponential Distribution: random variables Y_1 and Y_2 follow*
BVE($\lambda_1, \lambda_2, \lambda_{12}$), if the joint distribution satisfies $Pr(Y_1 > y_1, Y_2 > y_2) =$



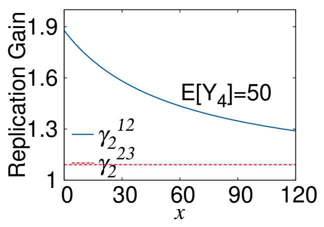
(a)



(b)



(c)



(d)

Figure 4: a) Fix $E[X_i] = 60$ min, $i = 1, 2, 3$, and varying $E[Y_4]$. b, c, d) Fixing $E[X_2] = E[X_3] = 60$ min, and varying x.

$e^{-(\lambda_1 y_1 + \lambda_2 y_2 + \lambda_{12} \max(y_1, y_2))}$

BVE was derived from a "fatal shock" model of a two component system, which is similar to the contact process shown in Figure 5. In the "fatal shock" model, shocks arrive at the system according to three independent Poisson processes with λ_1, λ_2 and λ_{12} parameters, and are applied to component 1, component 2, and both, respectively. The lifetimes of Y_1 and Y_2 components follow BVE. Similarly, the contact process between v, v_1, and v_2, can be seen as three independent Poisson processes with λ_1, λ_2 and λ_{12} parameters, where two processes govern v_1 and v_2 individually, while the third process characterizes the events when both v_1 and v_2 meet v. In this case, Y_1 and Y_2 are BVE jointly distributed with exponential marginals and correlation coefficient $= \lambda_{12}/(\lambda_1 + \lambda_2 + \lambda_{12})$.

Although BVE can be used to model correlated ICT for a pair of nodes, in real-world scenarios, a node may also meet with a group of nodes on a regular

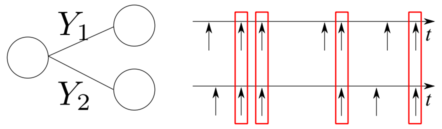


Figure 5: Contact process between (v, v_1) and (v, v_2).

basis. It is therefore necessary to characterize the joint distribution of a group of nodes. BVE is a special case of a Multivariate Exponential Distribution (MVE) [36], which can be used to model multi-dimensional joint ICT. MVE is defined as follows: Let G denote the set of vectors $\boldsymbol{g} = (g_1, \ldots, g_n)$ where each $g_i = 0$ or 1, but $(g_1, \ldots, g_n) \neq (0, \ldots, 0)$. For a vector $\boldsymbol{g} \in G$, let $\max(y_i g_i)$ denote the maximum of the y_i's for which $g_i = 1$.

Definition 5. *$Y_1, \ldots, Y_n$ random variables follow Multivariate Exponential Distribution, if their joint distribution satisfies:*
$Pr(Y_1 > y_1, \ldots, Y_n > y_n) = \exp\left[-\Sigma_{\boldsymbol{g}\in G}\lambda_{\boldsymbol{g}} \max(y_i g_i)\right]$

As an example, consider $n = 3$: $\lambda_{(100)} = \lambda_1$, $\lambda_{(010)} = \lambda_2$, $\lambda_{(001)} = \lambda_3$ are parameters for Poisson processes that govern each individual node; $\lambda_{(110)} = \lambda_{12}$, $\lambda_{(101)} = \lambda_{13}$, $\lambda_{(011)} = \lambda_{23}$, are parameters that govern each pair of nodes; and $\lambda_{(111)} = \lambda_{123}$ is the parameter that governs all three nodes. Marginal distribution of Y_1 and Y_2 follows a BVE$(\lambda'_1, \lambda'_2, \lambda'_{12})$, where $\lambda'_1 = \lambda_1 + \lambda_{13}$, $\lambda'_2 = \lambda_2 + \lambda_{23}$, $\lambda'_{12} = \lambda_{12} + \lambda_{123}$. Throughout this paper, we use $\boldsymbol{g}$ to denote a group of nodes, i.e., the node set $\{v_i : g_i = 1\}$.

In a real-world scenario, a node experiences a sequence of contact events, from which it estimates the parameters of the MVE distribution. We adopt a simple *consistent* and *unbiased* estimator derived in [37]. Consider that a node v has N encounter events within time T. At each event node v may meet one or more nodes. Let $E_{\boldsymbol{g},j} = 1$, if at the j-th event node v encounters a set of nodes $\{v_i : g_i = 1\}$; and $E_{\boldsymbol{g},j} = 0$ otherwise. Then the estimation of $\lambda_{\boldsymbol{g}}$ given these N observations is given by:

$$\hat{\lambda}_{\boldsymbol{g},N} = \frac{\frac{1}{N}\Sigma_{j=1}^{N} E_{\boldsymbol{g},j}}{\frac{T}{N-1}} \tag{1}$$

Algorithm 1 MVE Parameter Estimation For Node v

```
1: Initialize $EncounterSet = \emptyset$, $T = 0$, $N = 0$
2: $nextTimer \leftarrow \delta$
3: **upon connection up event**
4: Add $newneighbor$ to $EncounterSet$
5: **upon** $nextTimer$ **fired event**
6: Set $nextTimer \leftarrow nextTimer + \delta$
7: $T = T + \delta$
8: **if** $EncounterSet \neq \emptyset$ **then**
9:     $N = N + 1$
10:    **if** no $counter$ exist for $EncounterSet$ **then**
11:        Create new $counter$ for $EncounterSet$
12:        $counter = 0$
13:    Increment $counter$ for $EncounterSet$
14:    $EncounterSet \leftarrow \emptyset$
15: Update parameters for all $EncounterSet$ according to Equation 1
```

In the example shown in Figure 5, $N = 9$ contact events for node v. Assuming that the last event is at time T, according to Equation 1, we have $\hat{\lambda}_{(10),9} = \frac{8}{3T}$, $\hat{\lambda}_{(01),9} = \frac{16}{9T}$, $\hat{\lambda}_{(11),9} = \frac{32}{9T}$. In real-world scenarios, contact events with multiple nodes do not happen at the exact same time. We therefore aggregate multiple contact events that happen within an *estimation time window* as a single contact event with multiple nodes.

We present the MVE parameter estimation algorithm for a node v in Algorithm 1. We first initialize T, N and $EncounterSet$ (used to aggregate nodes contacted in the estimation time window δ). The $nextTimer$ is scheduled to fire at time t=δ. Upon each contact event, the newly encountered node is added to $EncounterSet$ (lines 3 to 4). When the $nextTimer$ is fired, the $nextTimer$ is set up (line 6) and T is incremented by one estimation time window. If the $EncounterSet$ is not empty, node v has met some nodes during the time window. The total contact event counter, N, is updated (line 9). Next, if no *counter* exists for the encountered set of nodes, a new counter is created (lines 10 to 12). Then, the *counter* for the encountered set of nodes is incremented and the *EncounterSet* is reset to empty. Finally, all the parameters are updated using Equation 1.

3.3. How to decide replication factors?

We have shown how delay correlation can affect replication gain and how to capture potential ICT correlations, however, we still need a mechanism to decide appropriate replication factors. Hence, we propose a novel algorithm based on regret-minimization [38] to dynamically decide replication factor. The objective of our algorithm is to obtain the best replication factor while we gain more information about the network and minimize the total cost (we define later). Our basic idea is to use a probability distribution to represent the preference of choosing a particular replication factor. Based on the probability distribution, RSock draws the value of the replication factor such that it is able to seamlessly switch between a single-copy to a multi-copy protocol based on current mission-critical deployable FENs connectivity conditions.

In order to demonstrate our idea, let us consider that each node in mission-critical deployable FENs as a player, who needs to repeatedly take actions (choosing a replication factor for a packet) under an *uncertain* environment (unknown network condition). Each time the node takes an action, the network gives a feedback (e.g., delay, resource utilization). The algorithm aims to learn the best policy, which is a *probability distribution* over a set of actions, for such an environment. A powerful technique for analyzing this problem is the Regret Analysis [38]. We aim for an algorithm which performs as good as the best action we could have used if we knew the network condition *a priori.* The performance degradation between our algorithm and the best action is defined as *regret.*

Let us assume a network with a set of nodes V and a pair of source and destination nodes (v_s, v_d). At time slot t, v_s chooses a probability distribution over a set of replication factors, i.e., $\boldsymbol{\pi}^t = \{\pi_r^t\}_{r=1}^{r_{max}}$. It then calculates a *loss vector*

Algorithm 2 Polynomial Weighted Algorithm

```
1: Initialize $w_r = 1$ for $r = 1, \cdots, r_{max}$; $\alpha, \eta \in (0, 1)$
2: at time slot $t$:
3: for $r = 1, \cdots, r_{max}$ do
4:     Estimate delay $D_r$ for $r$, set $\gamma_r = D_1/D_r$
5:     Update $l_r^t$
6:     Set $w_r = w_r \cdot (1 - \eta \cdot l_r^t)$
7: for $r = 1, \cdots, r_{max}$ do
8:     Set $\pi_r^t = w_r / \sum_{i=1}^{r_{max}} w_i$
```

$l^t = \{l_r^t\}_{r=1}^{r_{max}}$ (we define later) and experiences an expected loss of $\sum_{r=1}^{r_{max}} l_r^t \pi_r^t$. After T time steps, node v's total loss is $\sum_{t=1}^{T} \sum_{r=1}^{r_{max}} l_r^t \pi_r^t$. v_s draws replication factors using the same distribution and experiences the same expected loss. In retrospect, however, v_s could have chosen the same action i that results in the minimum total loss, i.e., with loss $L_i^T = \min_r \sum_{t=1}^{T} l_r^t$. The performance degradation of $\sum_{t=1}^{T} \sum_{r=1}^{r_{max}} l_r^t \pi_r^t - L_i^T$ is then defined as *regret*, which is to be minimized.

We define the loss function to reflect the trade-off of two objectives, i.e., minimizing delay and resource utilization.

Definition 6. *The loss function for r:*

$$l_r^t = \alpha \cdot \frac{1}{\gamma_r} + (1 - \alpha) \cdot \frac{r - 1}{r_{max} - 1}, s.t.\ \alpha \in (0, 1), r_{max} > 2.$$

$\frac{1}{\gamma_r} \in (0, 1]$ is the inverse of replication gain, which approaches 1 when there is no benefit of replication. $\frac{r-1}{r_{max}-1}$ represents the loss of resource utilization. $\alpha \in (0, 1)$ is a parameter that reflects the preference between optimizing performance and reducing overhead.

To decide the probability distribution of replication factors, we adopt the *Polynomial Weighted* (PW) algorithm, as presented in Algorithm 2. A weight w_r is assigned to each r, and is initialized to 1 at $t = 0$. At each time slot (lines 3 to 8), new estimations of the delays are obtained. l_r^t, w_r and π_r are then updated accordingly. At the end of each time slot, we have a new replication factor distribution. In order to run the algorithm, we need to estimate the delay of packets when different replication factors are used. In our design, RSock in each mission-critical deployable FENs device keeps sending probe packets to neighbors periodically using different replication factors to obtain the delay estimation. The PW algorithm is guaranteed to converge to zero regret after sufficient amount of time, and achieves *near optimal* performance in terms of regret (i.e., total cost) minimization [38].

3.4. RSock Design

Our RSock mainly consists of four modules, as shown in Figure 6. As we illustrated in Section 2.2, each mission-critical deployable FENs node runs EdgeKeeper, which has the topology manager service. The *MVE Estimation* module is responsible for estimation MVE parameters using Algorithm 1, which uses node events from EdgeKeeper. Each node also periodically exchanges the MVE parameters with their neighbors, as we presented in Section 3.3. In the following sections, we introduce the details for *Applications Packets Mux/Demux*, *Replication Factor Decision*, and *Packet Forwarding* modules.

3.4.1. Application-layer Packets Mux/Demux Module

The communications between any application and RSock daemon in any device is accomplished via *RSock Library*, as shown in Figure 6 for R-Drive, MDFS, or any future application. RSock Library employs a TCP socket to handle the following Application Programming Interface (API) calls:

1. Registration API call, in which the application registers itself with RSock. The registration is accomplished with three tuples $< Local_{ID}, App_{ID}, QoS_{val} >$. $Local_{ID}$ represents the local device ID (40-character string) that is obtained from EdgeKeeper and used in the sender field of RSock data packets. App_{ID} is a random string that the application generates for itself. App_{ID} is used by RSock for multiplexing and de-multiplexing purpose when sending and receiving packets from different applications, respectively, (i.e., as is the case of TCP/UDP port numbers). Notice that in

our current design, RSock guarantees the uniqueness of each application's App_{ID}. That is, the daemon rejects any application that tries to register with an already used App_{ID}. QoS_{val} is used to define any Quality-of-Service parameter for the packets. RSock provides a Time-to-Live (TTL) QoS parameter. That is, the maximum delay, in seconds, that the sender can tolerate before its packets get delivered to the final destination. If any packet's TTL expired before it reaches the final destination, it is simply dropped.

2. Transmission API call, in which the application sends a data with five tuples: $< Hash, Bytes, Size, Type, Dest_{ID} >$. The $Hash$ field contains the generated hash value by the sender for the data, which is used by RSock for fragmentation and de-fragmentation purposes in case the data size exceeds the IPv4 packet size (64KB). $Bytes$ and $Size$ fields contain the bytes of the data and its corresponding size, respectively. $Type$ field specifies whether this is a registration packet or not (it can be leveraged for future enhancement). $Dest_{ID}$ is the destination device's ID (i.e., obtained from EdgeKeeper).

3. Receiving API call. A zero-parameter blocking call in which the application waits for receiving any data. The returned data contains the $Hash$ value that the sender generated, data size, and the data itself as a byte array.

4. Termination API call, in which the application terminates its connection (i.e., un-register) with the RSock daemon. RSock drops any received packets for any terminated application.

3.4.2. Replication Factor Decision

This module maintains the replication factor probability distribution $\boldsymbol{\pi}$. In order to calculate $\boldsymbol{\pi}$, we need accurate information of packet delays for any given replication factors to derive corresponding replication gains. One way to achieve

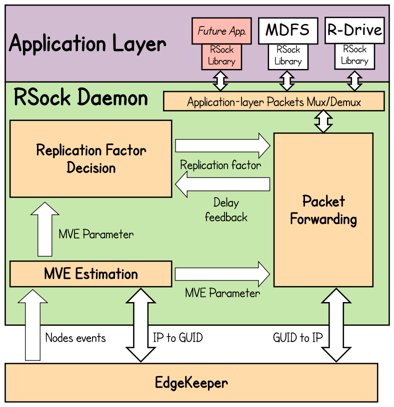


Figure 6: RSock architecture.

this goal is through direct-feedback from destination nodes. For example, destination nodes send *ack* packets once they receive the earliest delivered packet and we estimate delays as half of the Round Trip Time (RTT). This module runs an instance of Algorithm 2 and produces $\boldsymbol{\pi}$. That is, RSock daemons exchange MVE information once they discover each other and send probes to each discovered neighbor every t_{probe} seconds in order to collect delay information for different replication factors. The probes are sent along r paths specified by the Replication Factor Decision module. We leveraged Yen's algorithm [39] for finding multiple shortest paths to the destination. Once RSock receives a probe, it immediately sends back a reply. Upon the reception of the reply, the sender then feeds the round-trip time (RTT) to the Replication Factor Decision module to calculate replication factor distribution. In our implantation as we will present is Section 5, t_{probe} value is adjustable from the configuration file (its default value is set to 2 seconds). Once the replication factor distribution, $\boldsymbol{\pi}_m$, converge, no additional probes are sent. In order to save bandwidth, we adopt a *slow reduce, fast recover* approach to adjust the t_{probe}. That is, t_{probe} is set to $2 \cdot t_{probe}$ each time the absolute difference between $\pi_{m,r}$ and its last value is within 1%, for all r. Otherwise, t_{probe} is set back to its default value.

3.4.3. Packet Forwarding Module

This module implements the forwarding strategy. It uses a novel metric called *additional contact rate* (ACR) for measuring delivery capability, which is built using MVE parameters to incorporate the potential ICT correlation between different mission-critical deployable FENs nodes. It draws replication factors using $\boldsymbol{\pi}$ produced by the *Replication Factor Decision* module that we described above for new packets.

The packet copies specified by the Replication Factor Decision module need to be forwarded to appropriate packet carriers. We therefore use the MVE parameters and define a new forwarding metric called *additional contact rate* (ACR) that depends on the existing set of packet carriers. Let $\boldsymbol{c}$ denote the current set of packet carriers, and $C = \{j : c_j = 1\}$.

Definition 7. *ACR for node i to destination dst given packet carrier set $\boldsymbol{c}$:*

$$ACR_{v|\boldsymbol{c}} = \sum_{\boldsymbol{g}:g_i=1,g_j=0,j\in C} \lambda_{\boldsymbol{g}}^{dst}$$

$ACR_{v|\boldsymbol{c}}$ accounts for the correlation of ICTs to the destination, between v and the existing set of carriers. If v is highly correlated, $ACR_{v|\boldsymbol{c}}$ is likely to be small since the *additional* contribution made by v is little. To facilitate the forwarding strategy for RSock, we augment the packet header with four new fields.

1. $nrofCopies$ is the remaining number of data copies.
2. $nextCarrier$ specifies the intermediate destination for the packet. The current carrier sets this field when it decides to forward the packet to that node.
3. $carriers$ contains a list of packet carriers that currently hold this packet. This field facilitates the forwarding strategy to evaluate the ACR metric of a node.
4. $path$ contains a list of nodes that specifies a shortest path leading to the $nextCarrier$ in the current network.

Algorithm 3 RSock Forwarding Algorithm For Node v

```
1: $Neighbors \leftarrow$ TopologyManagement.$Neighbors$
2: for all Packet $m$ do
3:     if $m.dst$ is in $Neighbors$ then
4:         $m.nextCarrier \leftarrow m.dst$
5:         Find $m.nrofCopies$ shortest paths to $m.dst$
6:         Send $m$ along each path
7:     else if $m.nrofCopies > 1$ then
8:         $\boldsymbol{c} \leftarrow m.carriers$, $\boldsymbol{c}' \leftarrow m.carriers \backslash \{v\}$
9:         $u \leftarrow \arg\max_{u \in Neighbors} ACR_{u|\boldsymbol{c}}$
10:        if $ACR_{u|\boldsymbol{c}} > ACR_{v|\boldsymbol{c}'}$ then
11:            Duplicate $m' \leftarrow m$
12:            $m'.nextCarrier \leftarrow u$
13:            $m'.path \leftarrow ShortestPath(u)$
14:            $m'.nrofCopies = m'.nrofCopies - 1$
15:            Send $m'$ along $m'.path$
16:            Update $m.nrofCopies = 1$
```

We present the forwarding strategy in Algorithm 3. If the destination is in the same network with v, we send the packet along $nrofCopies$ paths (lines 3-6). This allows us to take advantage of the replication to deal with the potential lossy mesh network which v is connected to. If the network is well-connected, the $nrofCopies$ output by the learning algorithm has high probability to be 1, and thus reduces to a single shortest path forwarding. If the destination is not available and $nrofCopies > 1$, we send $nrofCopies - 1$ copies to the best possible carrier according to the ACR metric (lines 7- 16). Notice that only the carrier with more than 1 copy can replicate the packet and update the *carriers*, while other carriers only forward to the destination. Also, at any given time, there is at most one node with more than 1 copy. This allows RSock to keep track of the current packet carriers and to evaluate the ACR metric.

When a node receives a packet, it first checks if it is the destination. If it

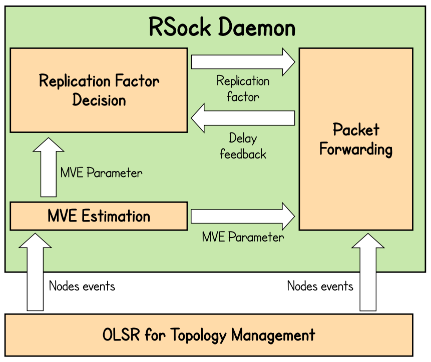


Figure 7: Initial RSock architecture.

is the destination, the node sends back an ACK containing the delay of the packet. Otherwise, it adds itself to the *carriers* if it is the *nextCarrier* or simply forwards the packet to the next hop according the *path* field if it is not the *nextCarrier*.

4. Implementation and Lessons Learned

Our first release of RSock was implemented using C++ as a user-space daemon service [40]. The design of this release is presented in Figure 7. The daemon handles both control traffic and the data packet forwarding. We leverage Optimized Link State Routing (OLSR) [41] for topology maintenance. That is, RSock invokes *olsrd* [17], an OLSR implementation, continuously to fetch the topology information (the graph view of the current connected network) from it. OLSR is a table-driven, proactive routing protocol developed for mobile ad-hoc networks. OLSR optimizes the pure link state protocol by reducing the size and number of the control packets that have to be transmitted between the nodes in the network. That is, by only sending packets to the Multipoint Relays (MPR), which is the minimum set of any node's one-hop neighbors that are able to reach all two-hop neighbors, instead of flooding the whole network. The network topology information in OLSR is periodically maintained by exchanging link state information. OLSR is implemented as a user-space daemon

(a) (b)

Phone #	Type	Android Version
P_1	Essential	7.1
P_2	Essential	7.1
P_3	Essential	7.1
P_4	Essential	9.0
P_5	Samsung Galaxy S8	7.1
P_6	Samsung Galaxy S8	7.1
P_7	Samsung Galaxy S8	7.1
P_8	Sonim XP8	7.1
P_9	Sonim XP8	7.1
P_{10}	Sonim XP8	7.1

(c)

Figure 8: a) A view of Disaster City. b) First responders team during a search-and-rescue mission with our manpack system during 2020 Winter Institute exercise in Disaster City. c) Smartphones used in our experiments.

service [17] and we cross-compiled for Android platforms. Our initial release of RSock has been evaluated during 2018 and 2019 Wireless institute exercises. In the following paragraphs, we will discuss the design and implementation's limitations of this release and how we addressed them.

Problems with IP-layer broadcast messages. In OLSR, each node broadcasts (to 255.255.255.255 IP address) hello message to discover other nodes and to calculate the link qualities and shortest paths in the network. However, we found that the LTE links in the NextEPC project [31] (that we leveraged in our manpack system) do not support broadcast messages. As a result, OLSR daemons were not able to discover each other over the LTE links. As a workaround and in order to enable the neighbor discovery over LTE links, we modified the OLSR daemons code for both smartphones and NUC (i.e., Android and Linux platforms). That is, the OLSR daemons on the smartphones have to uni-cast their hello messages to the OLSR daemon running at the NUC, which replies by uni-casting it's hello messages to the sender IP address. We faced a similar problem when we run OLSR daemons on the PSCR system. In which, the IP-layer broadcast messages are, by the default configuration, disabled for both Wi-Fi and LTE links for security purposes.

Seamless switching amongst interfaces. One major goal of our design, is to leverage the Wi-Fi and LTE links that are established between the smartphones and the manpack/PSCR systems. And that's why we decided to rely on OLSR, which in its RFC [41] claims that it is able to work on multiple interfaces and finding the shortest path (i.e., in terms of link qualities) amongst the nodes. RSock leverages the Jsoninfo plugin of OLSR for topology maintenance by invoking it periodically and fetching topology information from it. However, we found that the current OLSR and Jsoninfo plugin implementations [17] do not handle the scenarios when a smartphone switches amongst interfaces or changes it IP address on any interface. That is, the Jsoninfo plugin keeps announcing the stale IP address (i.e., the initial IP address of the interface in which the OLSR daemon was initiated) and never updates it. We fixed this bug by modifying the OLSR code [40] in order to keep Jsoninfo plugin aware of the updated/new IP address of the same/new interface of the device.

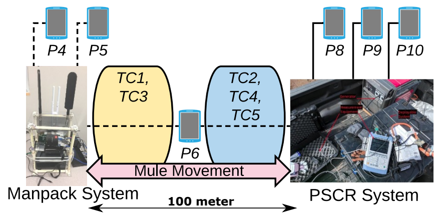
Running OLSR on unrooted Android smartphones. The latest release of OLSR [17] is only run-able on rooted Android smartphones. That is because OLSR updates the routing table in Android OS and accordingly requires root privileges for that. However, PSCR first responders' smartphones in the public safety department [30] are unrooted. Indeed, rooting first responders' smartphones is not recommended from security perspectives and it voids the smartphone warranty. Hence, in order to enable OLSR to run on unrooted Android smartphones, we provided a modified version of OLSR [40] that can be used for neighbor discovery and to calculate the link qualities amongst the devices (i.e., it doesn't update the phones' routing tables).

Development and deployment of OLSR and RSock. We cross-compiled our C++ RSock implementation and the modified version of OLSR for Android platforms using the ARM toolchain [42]. The deployment of these daemons was accomplished using the Android Debug Bridge (adb) [43] tool that pushes the daemons to the smartphones' file-systems via USB cables. On rooted smartphones, OLSR and RSock daemons can be initially pushed to any directory under the `sdcard` path and then (with root privilege) moved to any directory under the `data` path, in which they are able to run. On unrooted smartphones, we had to rigorously search (via trial and error) for the `data/local/tmp` path in which we could run both daemons. Indeed, the C++ implementations of these daemons hamper their development, debugging, and deployment tasks. That is, any code change requires the tedious cross-compilation and deployment process. The debugging was accomplished using the log files.

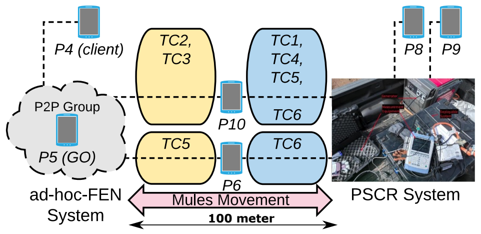
Due to the above issues/limitations, we enhanced RSock design, as we presented in Section 3, with the below goals.

Enable identity-based routing. The routing in the initial design of RSock was accomplished using nodes' IP addresses that are obtained from OLSR. That is, OLSR represents the network by an undirected graph $G = (V, E)$, where V is a set of nodes' IPs and E is a set of links. However, due to mobility, smartphones' IP addresses might change (i.e., in case of re-association or switching among different interfaces). Accordingly, the destination IP address that any application uses when sending data might be stale. In order to resolve the aforementioned issue, we integrated RSock with EdgeKeepr, as shown in Figure 6, in order to use its ID-based topology information. That is, each vertex in G is represented by node's ID instead of IP.

Easy deployment and running of RSock. We implemented RSock as an Android service [40], which is an application component that can perform long-running operations in the background for both Android and Linux platforms. RSock service can be installed as a regular Android App and it provides a simple user-friendly interface for starting and stopping purposes. Moreover, it is able to continuously run in the background even if the user switches to another application (e.g., R-Drive or MDFS). MDFS, R-Drive, and any future



(a)



(b)

Figure 9: A schematic diagram of the manpack system, PSCR system, and ad-hoc-FEN system. Mules connectivity during: a) TC_1 - TC_5 of Experiment-F. b) TC_1 - TC_6 of Experiment-G. Solid lines represent simultaneous Wi-Fi and LTE links. Dashed lines represent Wi-Fi or Wi-Fi Direct links.

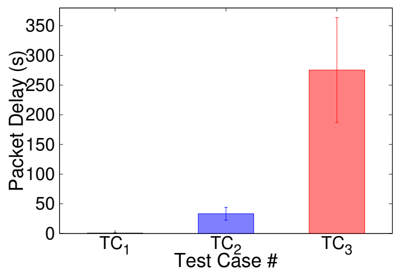
application of RSock can bind and interact with it by performing inter-process communication (IPC) calls via *Rsock Library*, as we illustrated in Section 3.4.1.

5. RSock Evaluation

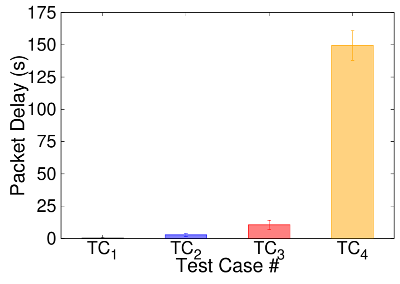
In this section, we present our experimental setup and the evaluation results of RSock in real-world scenarios.

5.1. Evaluation during the Winter Institute Exercise

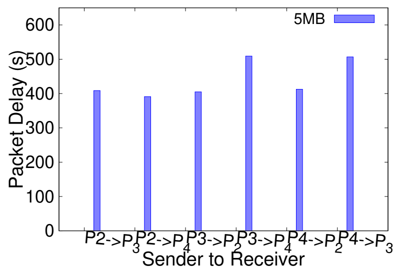
Our experiments were conducted on our manpack and the PSCR systems (presented in Section 2.2) during the Winter Institute exercise event in Disaster city [3] on January 2020. Disaster city is a comprehensive 52-acre training facility for emergency responders with extremely realistic wrecks, including several rubble piles of wood and concrete, as shown in Figure 8a. During the event, a team of first responders performed a wide-area search-and-rescue missions that mimicked looking for victims after a disaster, as shown in Figure 8b. We deployed RSock, R-Drive, and EdgeKeeper into the two systems and ten smartphones (their specifications are shown in Figure 8c). In the following paragraphs, we present our experiments:



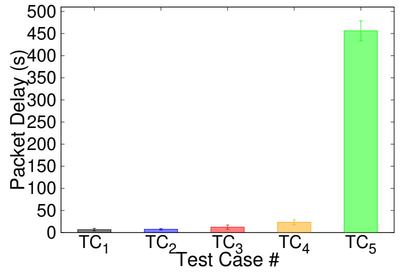
(a)



(b)



(c)



(d)

Figure 10: Packet Delay for all test cases for: a) Experiment-A. b) Experiment-B. c) Experiment-C. d) Experiment-D.

1. **Experiment-A:** This experiment was conducted with P_1, P_2, and P_3 smartphones that were connected using LTE to the manpack system with three test cases: TC_1-TC_3. In which, each phone sent 10KB, 1MB, and 5MB files (images or videos) to two other phones.
2. **Experiment-B:** This experiment was conducted with P_2, P_3, and P_4 smartphones that were connected using Wi-Fi to the manpack system with four test cases: TC_1-TC_4. In which, each phone sent 10KB, 200KB, 1MB, and 21MB files to two other phones, respectively.
3. **Experiment-C:** This experiment was conducted with P_2, P_3, and P_4 smartphones that were simultaneously connected using Wi-Fi and LTE to the manpack system. Each phone sent a 5MB file to two other phones. Then, after 10 seconds (i.e., of start sending the file) all phones were manually entered into airplane mode (disconnected from LTE and Wi-Fi). Then, after 5 minutes,

airplane mode was disabled (the phones received LTE and Wi-Fi connections again).

4. **Experiment-D:** This experiment was conducted with seven smartphones that were connected to the manpack system as follow: P_1-P_3 via Wi-Fi and P_4-P_7 via LTE. In TC_1, P_1 sent a "hi" text message to all other phones. Then, in TC_2, P_1 sent a 10KB file to all other phones. In TC_3 and TC_4, P_1 sent a 200KB and a 1MB file to all other phones. In TC_5, all phones except P_1 were manually entered into airplane mode for 5 minutes. Meanwhile, P_1 sent a 1MB file to all phones. Finally (after 5 minutes), airplane mode was disabled on P_2-P_7 phones.

5. **Experiment-E:** This experiment was conducted with P_6, P_7, and P_8 smartphones (that were connected via LTE to the PSCR system) with six test cases: TC_1-TC_6. In each test case, each phone sent a "hi" text message, 10KB, 200KB, 1MB, 1MB, and 10MB files to two other phones, respectively.

6. **Experiment-F:** This experiment aims to illustrate the interoperability of our softwares into two different mission-critical deployable FENs and the delay-tolerant routing capabilities of RSock. The two systems were physically separated by $\sim$100 m (i.e., the manpack system was indoor and the PSCR system was outdoor). We used P_8, P_9, and P_{10} phones that were simultaneously connected via LTE and Wi-Fi to the PSCR system as well as P_4, P_5, and P_6 phones that were connected via Wi-Fi to the manpack system. This experiment had five test cases, as presented in Figure 9a. In TC_1, while P_6 is connected to the manpack system, P_4 sent "hi" text message to P_8, P_9, and P_{10} phones. After 4 minutes, P_6 moved to the PSCR system and stayed there for $\sim$100 seconds. In TC_2 (while P_6 was connected to PSCR system), P_8, P_9, and P_{10} sent "hi" text messages to P_4, P_5, and P_6, respectively. In TC_3, P_6 moved to the manpack system and stayed there for 10 minutes. During that, P_4 sent "hi" text messages and a 10KB file to P_8, P_9, and P_{10}, respectively. In TC_4 and TC_5, (after 10 minutes), P_6 moved to the PSCR system.

7. **Experiment-G:** This experiment aims to illustrate the delay-tolerant routing capabilities of RSock when it runs between the PSCR system and an ad-hoc

edge network (ad-hoc-FEN), which is formed using Wi-Fi Direct of P_5 and P_4 phones. That is, P_5 was assigned the GO role and established a P2P group. P_4 was connected to P_5 as a client. For the PSCR system, we used P_8, P_9, and P_{10} phones that were connected via Wi-Fi. The two systems were physically separated by ∼100 m from each other (i.e., the ad-hoc-FEN system was indoor and the PSCR system was outdoor). This experiment had six test cases as presented in Figure 9b.

In TC_1, P_8 sent a 10KB file to P_9 and P_{10}, respectively. In TC_2, P_8 sent a 50KB file to P_9, P_{10}, P_4, and P_5, respectively. After ∼7 minutes, P_{10} moved to the ad-hoc-FEN system. In TC_3, P_4 sent a 10KB file to P_5, P_{10}, P_8, and P_9, respectively. In TC_4 (i.e., after ∼6 minutes), P_{10} moved to the PSCR system. Then (i.e., after P_{10} is connected), P_8 sent a 200KB file to P_9, P_{10}, P_5, and P_4, respectively. After that, a new smartphone, P_6, joined as a mule and moved between the ad-hoc-FEN, PSCR, and then back to the ad-hoc-FEN within ∼10 minutes. In TC_5, P_4 sent a 1MB file to P_8, P_9, and P_{10}. Then, after ∼4 minutes, P_6 moved back to the PSCR system. In TC_6, P_{10} switched to LTE only connectivity to the PSCR system (i.e., instead of Wi-Fi) and sent a 1 MB file to P_6, P_5, and P_4, respectively. Finally. after ∼5 minutes P_6 disconnected from the PSCR system and connected back to the ad-hoc-FEN system.

5.1.1. Evaluation Results

In this section, we present the performance evaluations of RSock in terms of PDR, PDD, and overhead. PDR measures the amount of useful information delivered to destinations within the deadlines, which indicates the routing protocols' performance of finding efficient packet carriers within a time constraint. The PDD of a packet is the time the packet has spent in the network before it reaches its destination. The overhead is defined as:

$$\frac{\#relays - \#delivered}{\#delivered}.$$

#relays is the total number of transmissions (i.e., the total number of message copies). *#delivered* is the total number of delivered messages (i.e., only

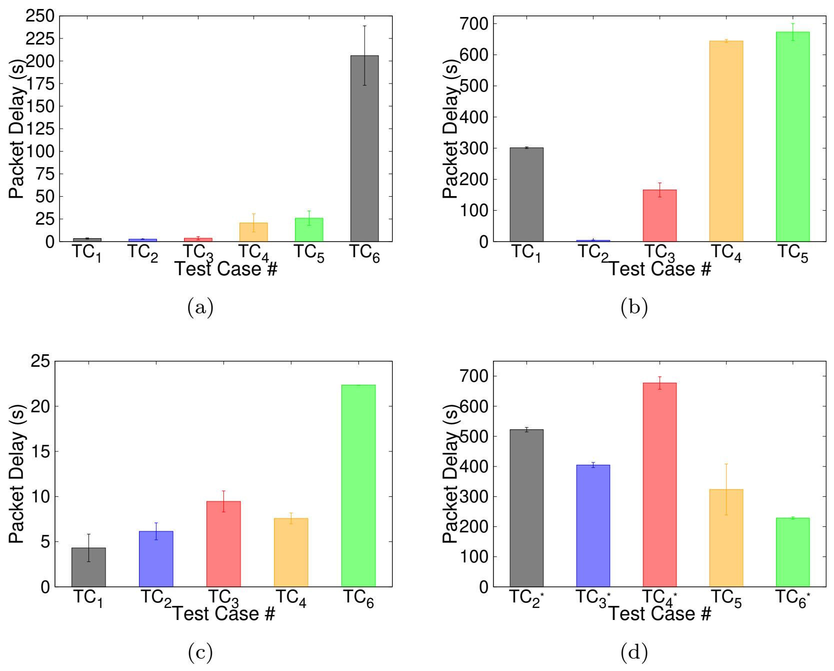


Figure 11: Packet Delay for all test cases for: a) Experiment-E. b) Experiment-F. c & d) Experiment-G.

the first copy that reaches the destination counts as delivered and the rest are discarded).

We also measure the *control traffic ratio*, defined as:

$$1 - \frac{size\ of\ payload}{size\ of\ data\ packets + size\ of\ control\ packets}.$$

Payload refers to application data; each *data packet* includes a header and a payload; *data packets* may contain packet replicas; and *control packets* include packets for exchanging MVE parameters and probes. In the following paragraphs, we will discuss the evaluation of our experiments for the aforementioned metrics. The results in Figures 10 - 12 are averaged per each test case with error bars showing standard deviation.

Evaluation in Fully Connected Mesh Network. Figures 10a and 10b show the PDDs for all test cases of Experiment-A and Experiment-B in our

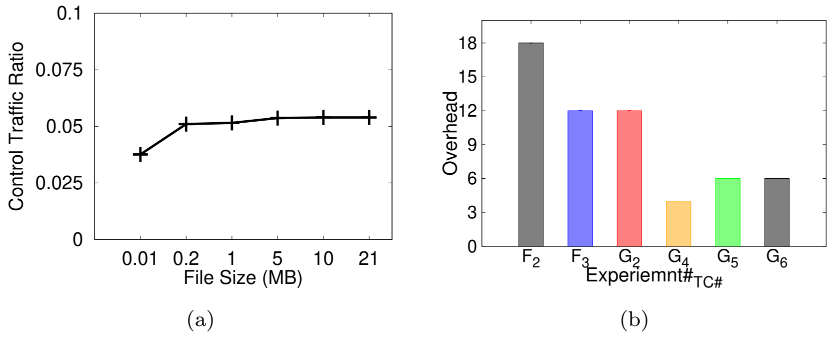


Figure 12: a) Control Traffic Ratio for all files. b) Overhead of RSock for DTN Experiments.

manpack system, respectively. Moreover, Figure 11a shows the PDDs for all test cases for Experiment-E in the PSCR system. Notice that the networks during these experiments are fully connected. That is, there is no disconnection or re-association of the smartphones' Wi-Fi/LTE links. In these experiments, we intend to show the routing capabilities of RSock in the manpack and PSCR systems for various smartphones and file sizes when the network is fully connected. As is clear in Figures 10a and 10b, RSock is able to deliver all files with different sizes (10KB to 21MB) over Wi-Fi and LTE links. The PDD increases while the file size increases (e.g., over the Wi-Fi network of the manpack system, the PDD ranges from ∼1 second up to ∼150 seconds for the files with 10KB to 21MB sizes, respectively). The PDRs for all test cases of these experiments is 100%.

Evaluation in Diverse Network Conditions. Figure 10c shows the PDD for all test cases of Experiment-C. This experiment demonstrates the resiliency of RSock when the smartphones lose their connections during packet transmissions. That is, when smartphones re-associate their Wi-Fi and LTE connections (might be with new IP address for the LTE connection). Notice that this kind of connection disruption commonly occurs due to the mobility of the smartphones' carriers in mission-critical deployable FENs. The PDD for all test cases is > 300 seconds (i.e., the time-period that the connections were lost). In addition to the 300 seconds, the topology discovery manager of EdgeKeeper [24] requires some time to learn the updated graph view of the network (i.e., once the smartphones either connect/disconnect to the manpack system). The PDR for all test-cases is 100%.

In Experiment-D, all test cases of Experiment-A and Experiment-C are combined with additional smartphones that use either Wi-Fi or LTE connections. That is, seven smartphones were fully connected in TC_1-TC_4 and then disconnected (except P_1), for 5 minutes in TC_5. As shown in Figure 10d, RSock is able to deliver all packets with 100% PDR and comparable PDDs (i.e., compared to Experiment-A and Experiment-C in which we used three smartphones).

Evaluation in DTN between the manpack and PSCR systems. TC_1 in Figure 11b shows the PDD for the "hi" text message via the mule between the manpack and PSCR systems in Experiment-F. Notice that, once the mule (i.e., P_6 in our case) becomes physically close to a system, it automatically discovers its Wi-Fi or LTE networks and re-associate with them. That is, P_6 pre-stored their credentials, e.g., Wi-Fi passphrase of both systems and LTE configurations of the PSCR system. In TC_2 (while P_6 is connected to the PSCR system) it instantly received the "hi" text message that is sent from P_8, P_9, and P_{10}. Then P_6 carried and delivered (i.e., after 100 seconds) P_8, P_9, and P_{10}'s "hi" text messages to $P4$ and $P5$ as presented in TC_3 in Figure 11b. TC_4 and TC_5 in Figure 11b show the PDD of the "hi" text messages and the 10KB files, respectively, that are sent from P_4 in the manpack systems and then (i.e., after 600 seconds) carried and delivered to P_8, P_9, and P_{10} via P_6. The PDR of all test cases of this experiment is 100%.

Evaluation in DTN between the ad-hoc-FEN and PSCR systems. TC_1 and TC_2 in Figure 11c present the PDDs for the 10KB and 50KB files that were sent from P_8 to both P_9 and P_{10} in the PSCR system, respectively. TC_{2^*} in Figure 11d shows the PDD of the 50KB files that were sent from P_8 in the PSCR system and carried and delivered after (420 seconds) to P_4 and P_5 in the ad-hoc-FEN system. Once the mule (i.e., P_{10}) is connected to the ad-hoc-FEN system during TC_3, it received along with P_5 a 10KB file with the PDD that is shown in TC_3 in Figure 11c from P_4. Then after $\sim$6 minutes, $P10$ carried and delivered the 10KB file to P_8 and P_9 in the PSCR system, as shown in TC_{3^*} in Figure 11d. The PDD of the 200KB file that is sent from P_8 and delivered to P_9 and P_{10} within $\sim$5 seconds is shown in TC_4 in Figure 11c. The aforementioned file is then carried via the new mule, P_6, to both P_5 and P_4 in the ad-hoc-FEN system, as presented in TC_{4^*} in Figure 11d. TC_{5^*} in Figure 11d shows the PDD of three 1MB files (this demonstrates the capability of RSock to deliver relatively large files in DTN scenarios via the mule, P_6). TC_{6^*} in Figure 11d shows the PDD when another phone, P_{10}, connected to the PSCR system sent a 1MB file, however; via LTE to P_5 and P_4 in the ad-hoc-FEN system. The file is also carried and delivered via P_6.

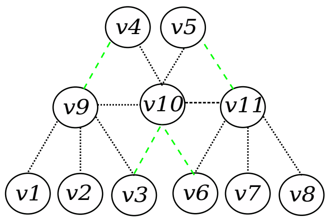
As for control traffic overhead, RSock has a ratio that ranges from 2.8% up to 6% for 0.01-21 MB files, as shown in Figure 12a. The overhead is resulted from the additional header that *RSock Library* adds to the packets' payloads to handle the fragmentation and the QoS parameters, as we presented in Section 3.4. Similarly, the overhead of the DTN experiments that is presented in Figure 12b is decreasing from TC_2 to TC_3 for Experiment-F. That is, after RSock daemons in the manpack system's smartphones discover the path (i.e., via P_6) to the PSCR system. The same decrease in the overhead is also observed from TC_2 to TC_4 as well as from TC_5 to TC_6 for Experiment-G for the two mules, P_{10} and P_6, respectively, as presented in Figure 12b.

5.2. Evaluation using a Wireless Testbed

We deployed RSock and EdgeKeeper daemons on 11 Asus Eee notebooks that run Ubuntu 14.04 LTS. The wireless cards of the notebooks are set to operate in 2.4 GHz (channel 3) and in ad-hoc mode. Due to space limitation and for the ease of testing, we place all notebooks together and emulate a fully connected mission-critical deployable FENs by manipulating the firewall configurations. To this end, we connected all the notebooks to a server through Ethernet cables and issued *iptable* commands to enable/disable the links, as shown in Figure 13a.

We conducted two sets of experiments, i.e., in a fully connected FENs and

(a)

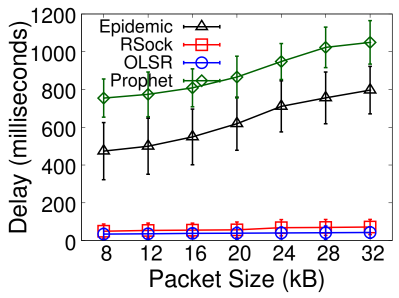


(b)

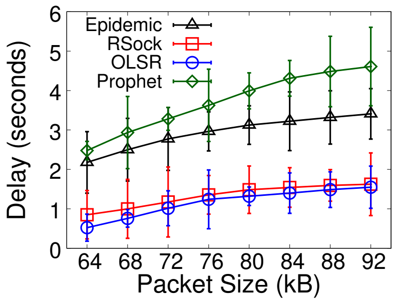
Figure 13: (a) Testbed for our experiments. (b) Network topology for the Wireless Testbed (dashed green lines are for on-off experiments only).

a dynamic on-off FENs. For the fully connected FENs, we created three FENs (with v_9, v_{10}, and v_{11} running as the EdgeKeeper master), as shown in Figure 13b. For each experiment, we generated a data flow with different data packets sizes (8 kB to 92 kB) within each FEN and we set the deadline of each packet to 300 seconds. For on-off FENs, we used the same baseline topology as shown in Figure 13b but turned each link on and off randomly with different duty cycles (in order to create contact events between the nodes as it is the case in mission-critical deployable FENs). The duration for one on-off cycle is set to 60 seconds. For the on-off FENs, we generated a data flow for different data packet sizes amongst the three FENs, as shown in Figure 13b. In addition to RSock, we use OLSR [41], Epidemic [10], and Prophet [18] routing protocols for comparison. We intend to show that: 1) OLSR, Epidemic, and Prophet protocols perform poorly when they used outside their designed scenario; and 2) that RSock performs closely to the best protocol for each specific scenario. For OLSR, Epidemic, and Prophet protocols, we leveraged *olsrd* and IBR-DTN [17, 18, 33, 34] implementations. We repeated each experiment 30 times and we averaged these 30 runs per each experiment. The error bars in the Figures 14, 15, and 16 show the standard deviations.

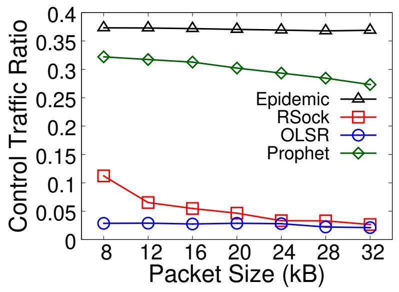
For the fully connected FENs, the PDR for all packets is 100% for all protocols (i.e., RSock, OLSR, Epidemic, and Prophet). The packets delays and control traffic ratios for the fully connected FENs are shown in Figure 14. As



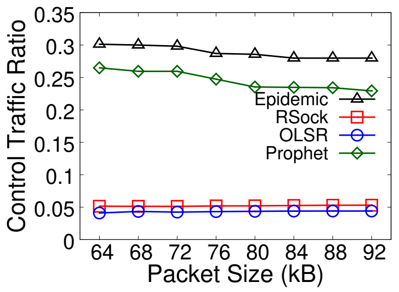
(a)



(b)



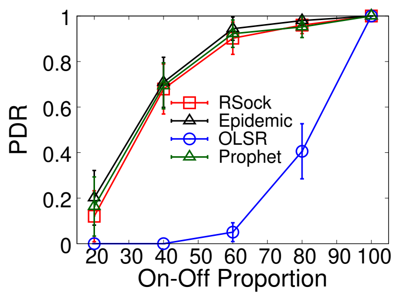
(c)



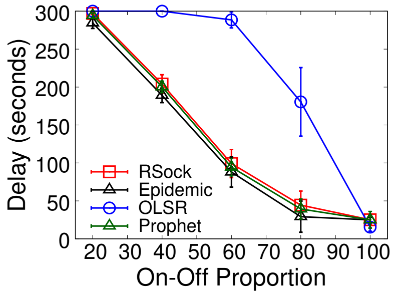
(d)

Figure 14: Packet Delay for the experiments on the fully connected FENs for: a) 8 kB to 32 kB packets sizes. b) 64 kB to 92 kB packets sizes. Control Traffic Ratio for the experiments on the fully connected FENs for: c) 8 kB to 32 kB packets sizes. d) 64 kB to 92 kB packets sizes.

shown in Figures 14a and 14b, RSock achieves comparable packets delays with OLSR. The delays for RSock range from 49.42 ms up to 1623.58 ms. On the other hand, OLSR's packets delivery delay is between 34.34 ms and 1547.80 ms. For control traffic overhead, as shown in Figures 14c and 14d, RSock is higher than OLSR and significantly smaller than Epidemic and Prophet. RSock has control traffic ratios that range from 11% down to 2.6%, whereas OLSR ranges from 4.40% down to 2.87%. The additional overhead for RSock is required in order to adapt to the network condition. However, as we will see later for the on-off FENs, RSock ability to adapt to network condition leads to superior performance in FENs with dynamic connectivity.



(a)



(b)

Figure 15: a) Packet Delivery Ratio for the on-off experiment. b) Packet Delay for the on-off experiments.

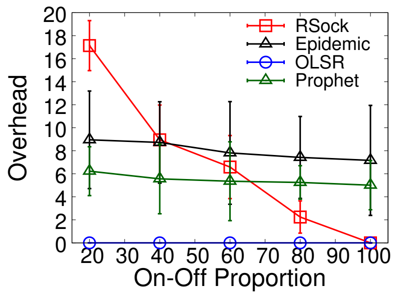


Figure 16: Overhead for the on-off experiments

Figure 15a shows the PDRs for the on-off FENs. RSock performs very close to Epidemic and Prophet in terms of PDR under different on-proportions. When the on-proportion is 20%, Epidemic achieves PDR with 20.15%, Prophet achieves PDR with 16.33%, whereas RSock's PDR is 12.05%. That's because in extremely sparse connectivity FENs, Epidemic's flooding strategy is efficient. Moreover, both Epidemic and Prophet are specifically designed for sparsely connected FENs. However, when the on-proportion ranges from 40% to 100%, RSock's PDRs are within 4.17% difference compared to both Epidemic and Prophet's PDRs. Similar observations were obtained for the packet delays, as presented in Figure 15b. When the on-proportion is 20%, Epidemic achieves PDD with 284.45 seconds, Prophet achieves PDD with 294.23 seconds, whereas RSock's PDD is 297.01 seconds. When the on-proportion ranges from 40%

to 100%, RSock's PDDs are within 15.23 seconds difference compared to both Epidemic and Prophet's PDDs. However, RSock achieves the aforementioned results with less overhead compared to Epidemic and Prophet when the on-proportion is $> 40\%$ and $> 60\%$, respectively, as is shown in Figure 16. On the other hand, as OLSR is designed for the fully-connected networks (i.e., when the on-proportion is 100%), it achieves significantly lower PDR and significantly higher PDD when the on-proportion is $< 100\%$ compared to RSock, Epidemic, and Prophet, as shown in Figures 15a and 15b. Moreover, OLSR has zero overhead (as shown in Figure 16) as it does not generate redundant traffic under all on-off FENs scenarios (and hence it does not efficiently deliver the packets when the on-proportion is $< 100\%$ either).

6. State of the art

Routing protocols for networks with diverse connectivity. Many routing protocols [10, 11, 12, 13, 14, 15, 16, 44, 45, 46, 47, 48, 49, 50] have been proposed for DTN recently. In general, packet replication is adopted to improve routing performance [10] [11] [13]. In order to estimate the delivery capability of individual nodes, social network analysis is accomplished to develop social-based metrics [12] [14] [15] that can be leveraged to improve routing efficiency. On the other hand, previous research [19] [51, 52, 53, 54, 55, 56, 57] have proposed to integrate the connected networks and sparsely connected networks' routing protocols via different approaches. The store-and-forward mechanism that is leveraged by DTN' routing protocols is the major difference compared to the routing protocols that are designed for the well-connected mobile ad-hoc or mesh networks. One simple approach is to integrate the store-and-forward mechanism to mobile ad-hoc or mesh routing protocols [53] [54] [56]. That is to store the packet when the next-hop/destination is unavailable, and forward it when it is available. The aforementioned approach might be suitable for specified network conditions. However, due to its inability to select appropriate packet carriers in disconnected networks and the limitation of using the single copy routing degrade its performance in many scenarios.

Other approaches [51] [52] [55] [57] aim to switch between mobile ad-hoc and mesh routing protocols. That is, to enable the source node to pick a specific routing protocol based on some conditions. In [51], the network is partitioned into multiple diameter-constrained sub-networks in which mobile ad-hoc routing protocols are used for intra-sub-network routing and DTN routing protocols are used for inter-sub-network routing. In [55], the source node decides which protocol to use based on some collected information (e.g., packet size, node density, etc.). For example, if the nodes are moving fast, the network is sparse, and/or the message size is large, a DTN routing protocol is used. Else, a mobile ad-hoc routing protocol is used. The aforementioned approaches have many limitations. For example, once the decision is made at the source [55], the intermediate nodes cannot change it even though they might have more up-to-date information. In [51], the border line that distinguishes DTN and ad-hoc/mesh networks is not defined, which makes it difficult to select an appropriate switching point.

In R3 [19], the relationship between replication benefit and the path delay predictability was observed. Based on the analytical model for replication, R3 protocol aims to dynamically replicate packets according to the network condition. That is, the nodes keep probing the network to collect path delay distributions and then the shortest path (in terms of delay) is selected first. The source node adds additional paths (i.e., to the packet header for source routing) if the replication gain is greater than a certain threshold. Inspired by R3, our work delves deeper into understanding replication. We find that the correlation of path delays can significantly reduce the benefit gained from packet replication and propose a novel model for capturing the potential correlation of ICT

Software Defined Networks (SDN)-based FENs enabled IoT services. In many recent research publications, SDN approaches have been employed in FENs in order to enhance their network performance [58, 59, 60, 61]. In [58], an architecture that combines SDN and Fog computing is proposed. That is, an SDN controller that is responsible for resource management and orchestration is deployed between the fog and cloud layers. Gupta et al. illustrated the effectiveness of SDN and Network Function Virtualization technologies for FENs over a health smart-home use case scenario and proposed SDFog [59], an SDN-based fog computing system to perform QoS-aware management amongst different FENs services' flows. Sun et al. proposed EdgeIoT [60], a fog-to-cloud hierarchical structure to effectively handle the data produced by IoT devices. In their design, the SDN-based cellular core is located at top of fog servers and is responsible for data forwarding amongst fog servers. Zeng et al. proposed an SDN load balancing scheme [61] for FENs in order to effectively leverage network resources. They proposed an effective task scheduling and resource management system to satisfy user experiences. They formulated the problem as mixed-integer nonlinear programming problem with three constraints: transmission time, I/O time, and computation time. Their evaluation results show the efficiency of their approach in minimizing FENs task completion time. Even though the aforementioned SDN approaches are able to enhance the network performance, they require a network administrator that is always available to apply different rules/configurations according to different scenarios, which might be tedious in disaster-response events. Moreover, the aforementioned approaches have been mainly evaluated via simulations without a reliable full-system implementation that can be deployed into real-world systems.

Jin et al. [62] proposed an optimization formulation for the Virtual Network Function (VNF) chain deployment problem that minimizes the resource consumption of both edge servers and physical links under the latency limitations. That is, by taking the queuing delay at the APs into account for meeting the latency requirements. They formulated the VNF chain deployment problem as a mixed integer linear programming (MILP) to minimize the total resource consumption and evaluated it via Extensive simulations based on real-world topologies. Similarly, Gao et al. [63] studied the problem of jointly optimizing the access network selection and service placement for Mobile Edge Cloud (MEC), towards the goal of improving the QoS by balancing the access, switching and communication delay. Specifically, they first design an efficient online framework to decompose the long-term optimization problem into a series of one-shot problems. They proposed an iteration-based algorithm to derive a computation efficient solution and validated the efficacy of their proposed solution by both rigorous theoretical analysis and extensive trace-driven simulations.

7. Conclusions

We present Resilient Socket (RSock), a novel identity-based routing protocol for mission-critical deployable FENs that exhibits diverse connectivity characteristics. RSock is able to adjust its routing and replication decisions to deal with any dynamic mission-critical deployable FENs environments. That is, by leveraging EdgeKeeper [24], a novel coordination and naming service for FENs, to accomplish the identity-based routing and to exploit all wireless interfaces of FENs devices. We carefully design RSock to guarantee the ease of its deployment in Android and Linux platforms and to be leveraged and adopted (i.e., with minimal code changes) by modern/future mission-critical deployable FENs' applications. Furthermore, to facilitate its future enhancement for new requirements. We provide a reliable full-system implantation of RSock and extensively evaluate it in two real-world systems in diverse real-world disaster-response scenarios to illustrate its routing capabilities.

Acknowledgment

This material is based upon a work supported by National Institute of Standards and Technology (NIST) under grant NO. (#70NANB17H190). We appreciate the helpful comments and suggestions from the reviewers.

References

[1] Distressnet-ng project. http://distressnet.net/.

[2] C. Yang, R. Stoleru, Hybrid routing in wireless networks with diverse connectivity, in: Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2016, pp. 71–80.

[3] Winter institute workshops. https://itec.tamu.edu/winter-institute-2018/.

[4] The android team awareness kit. https://takmaps.com.

[5] K. Usbeck, M. Gillen, J. Loyall, A. Gronosky, J. Sterling, R. Kohler, K. Hanlon, A. Scally, R. Newkirk, D. Canestrare, Improving situation awareness with the Android Team Awareness Kit (ATAK), in: Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security, Defense, and Law Enforcement, Vol. 9456, 2015, pp. 172 – 193.

[6] C. Chen, M. Won, R. Stoleru, G. G. Xie, Energy-efficient fault-tolerant data storage and processing in mobile cloud, IEEE Transactions on Cloud Computing 3 (1) (2015) 28–41.

[7] M. Sagor, R. Stoleru, S. Bhunia, M. Chao, A. Haroon, A. Altaweel, M. Maurice, R. Blalock, R-drive: Resilient data storage and sharing for mobile edge computing systems, arXiv (2022). `doi:10.48550/ARXIV.2204.10823`.

[8] R-drive for distressnet-ng. https://github.tamu.edu/lenss/rsockimageshare.

[9] Mdfs: Mobile distributed file system for fog/edge networks. https://github.tamu.edu/lenss/mdfs_ng.

[10] M. J. F. Alenazi, Y. Cheng, D. Zhang, J. P. G. Sterbenz, Epidemic routing protocol implementation in ns-3, in: Proceedings of the 2015 Workshop on Ns-3, 2015, p. 83–90.

[11] T. Spyropoulos, K. Psounis, C. S. Raghavendra, Spray and wait: An efficient routing scheme for intermittently connected mobile networks, in: Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking, WDTN '05, 2005, pp. 252–259.

[12] E. M. Daly, M. Haahr, Social network analysis for routing in disconnected delay-tolerant manets, in: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing, 2007, pp. 32–40.

[13] A. Balasubramanian, B. Levine, A. Venkataramani, Dtn routing as a resource allocation problem, in: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, 2007, pp. 373–384.

[14] P. Hui, J. Crowcroft, E. Yoneki, Bubble rap: Social-based forwarding in delay-tolerant networks, IEEE transactions on mobile computing 10 (11) (2010) 1576–1589.

[15] W. Gao, Q. Li, B. Zhao, G. Cao, Multicasting in delay tolerant networks: A social network perspective, in: Proceedings of the Tenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '09, 2009, pp. 299–308.

[16] E. Bulut, S. C. Geyik, B. K. Szymanski, Utilizing correlated node mobility for efficient dtn routing, Pervasive Mob. Comput. 13 (2014) 150–163.

[17] Implementation of optimized link state routing protocols for mobile ad-hoc networks. https://github.com/olsr/olsrd.

[18] A. Lindgren, A. Doria, E. Davies, S. Grasic, Probabilistic routing protocol for intermittently connected networks. irtf rfc 6693 (2012).

[19] X. Tie, A. Venkataramani, A. Balasubramanian, R3: Robust replication routing in wireless networks with diverse connectivity characteristics, in: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, MobiCom '11, 2011, pp. 181–192.

[20] Z. Li, H. Shen, A qos-oriented distributed routing protocol for hybrid wireless networks, IEEE Transactions on Mobile Computing 13 (3) (2014) 693–708.

[21] W. A. Jabbar, W. K. Saad, M. Ismail, Meqsa-olsrv2: A multicriteria-based hybrid multipath protocol for energy-efficient and qos-aware data routing in manet-wsn convergence scenarios of iot, IEEE Access 6 (2018) 76546–76572.

[22] I. Kacem, B. Sait, S. Mekhilef, N. Sabeur, A new routing approach for mobile ad hoc systems based on fuzzy petri nets and ant system, IEEE Access 6 (2018) 65705–65720.

[23] A. Keränen, J. Ott, T. Kärkkäinen, The one simulator for dtn protocol evaluation, in: Proceedings of the 2nd international conference on simulation tools and techniques, 2009, pp. 1–10.

[24] S. Bhunia, R. Stoleru, M. Sagor, A. Haroon, A. Altaweel, M. Chao, M. Maurice, R. Blalock, Edgekeeper: Resilient and lightweight coordination for mobile edge computing systems, arXiv (2022). `doi:10.48550/ARXIV.2204.11095`.

[25] Edgekeeper for distressnet-ng. https://github.tamu.edu/lenss/gns.

[26] M. Chiang, T. Zhang, Fog and iot: An overview of research opportunities, IEEE Internet of Things Journal 3 (6) (2016) 854–864.

[27] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 2012, pp. 13–16.

[28] L. M. Vaquero, L. Rodero-Merino, Finding your way in the fog: Towards a comprehensive definition of fog computing, ACM SIGCOMM computer communication Review 44 (5) (2014) 27–32.

[29] A. Altaweel, R. Stoleru, G. Gu, Evildirect: A new wi-fi direct hijacking attack and countermeasures, in: 26th International Conference on Computer Communication and Networks (ICCCN), 2017, pp. 1–11.

[30] Public safety communications research division. https://www.nist.gov/ctl/pscr.

[31] Nextepc project. https://nextepc.org/.

[32] Cisco visual networking index: Global mobile data traffic forecast update, 2017-2020. https://www.cisco.com/ (2019).

[33] S. Schildt, J. Morgenroth, W.-B. Pöttner, L. Wolf, Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation, ECEASST 37 (01 2011).

[34] Ibr-dtn - a modular and lightweight implementation of the bundle protocol. https://github.com/ibrdtn/ibrdtn.

[35] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, S. Shenker, Low latency via redundancy, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, 2013, pp. 283–294.

[36] A. W. Marshall, I. Olkin, A multivariate exponential distribution, Journal of the American Statistical Association 62 (317) (1967) pp. 30–44.

[37] B. C. Arnold, Parameter estimation for a multivariate exponential distribution, Journal of the American Statistical Association 63 (323) (1968) pp. 848–852.

[38] N. Nisan, T. Roughgarden, E. Tardos, V. V. Vazirani, Algorithmic Game Theory, 2007.

[39] J. Y. Yen, Finding the k shortest loopless paths in a network, Management Science 17 (11) (1971) 712–716.

[40] Rsock implementation. https://github.tamu.edu/lenss/.

[41] T. H. Clausen, C. Dearlove, P. Jacquet, U. Herberg, The optimized link state routing protocol version 2. ietf rfc 7181 (2014).

[42] Android ndk toolchain. https://developer.android.com/ndk.

[43] Android debug bridge (adb). https://developer.android.com/studio /command-line/adb.

[44] N. Srinidhi, C. Sagar, J. Shreyas, D. K. SM, An improved prophet-random forest based optimized multi-copy routing for opportunistic iot networks, Internet of Things 11 (2020).

[45] M. W. Kang, D. Y. Seo, Y. W. Chung, An efficient opportunistic routing protocol for icn, ICN'19.

[46] Y. Mao, C. Zhou, Y. Ling, J. Lloret, An optimized probabilistic delay tolerant network (dtn) routing protocol based on scheduling mechanism for internet of things (iot), Sensors 19 (2) (2019) 243.

[47] K. M. Baek, D. Y. Seo, Y. W. Chung, An improved opportunistic routing protocol based on context information of mobile nodes, Applied Sciences 8 (8) (2018).

[48] D. K. Sharma, S. K. Dhurandher, I. Woungang, R. K. Srivastava, A. Mohananey, J. J. P. C. Rodrigues, A machine learning-based protocol for efficient routing in opportunistic networks, IEEE Systems Journal 12 (3) (2018) 2207–2213.

[49] S. Basu, A. Biswas, S. Roy, S. DasBit, Wise-prophet: a watchdog supervised prophet for reliable dissemination of post disaster situational information over smartphone based dtn, Journal of Network and Computer Applications 109 (2018) 11–23.

[50] S. J. Borah, S. K. Dhurandher, S. Tibarewala, I. Woungang, M. S. Obaidat, Energy-efficient prophet-prowait-edr protocols for opportunistic networks, GLOBECOM'17.

[51] J. Whitbeck, V. Conan, HYMAD: Hybrid DTN-MANET routing for dense and highly dynamic wireless networks, Comput. Commun. 33 (13) (2010) 1483–1492.

[52] Z. Lu, G. Cao, T. L. Porta, Networking smartphones for disaster recovery, in: 2016 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2016, pp. 1–9.

[53] M. Demmer, K. Fall, Dtlsr: Delay tolerant routing for developing regions, in: Proceedings of the 2007 Workshop on Networked Systems for Developing Regions, NSDR '07, 2007, pp. 5:1–5:6.

[54] C. Raffelsberger, H. Hellwagner, A hybrid manet-dtn routing scheme for emergency response scenarios, 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops) (2013) 505–510.

[55] J. Lakkakorpi, M. Pitkänen, J. Ott, Adaptive routing in mobile opportunistic networks, in: Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, MSWIM '10, 2010, pp. 101–109.

[56] L. Delosières, S. Nadjm-Tehrani, Batman store-and-forward: The best of the two worlds, in: 2012 IEEE International Conference on Pervasive Computing and Communications Workshops, 2012, pp. 721–727.

[57] C. Kretschmer, S. Ruhrup, C. Schindelhauer, Dt-dymo: Delay-tolerant dynamic manet on-demand routing, in: 2009 29th IEEE International Conference on Distributed Computing Systems Workshops, 2009, pp. 493–498.

[58] N. B. Truong, G. M. Lee, Y. Ghamri-Doudane, Software defined networking-based vehicular adhoc network with fog computing, in: IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015.

[59] H. Gupta, S. B. Nath, S. Chakraborty, S. K. Ghosh, Sdfog: A software defined computing architecture for qos aware service orchestration over edge devices, arXiv preprint arXiv:1609.01190 (2016).

[60] X. Sun, N. Ansari, Edgeiot: Mobile edge computing for the internet of things, IEEE Communications Magazine 54 (12) (2016) 22–29.

[61] D. Zeng, L. Gu, S. Guo, Z. Cheng, S. Yu, Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system, IEEE Transactions on Computers 65 (12) (2016) 3702–3712.

[62] P. Jin, X. Fei, Q. Zhang, F. Liu, B. Li, Latency-aware vnf chain deployment with efficient resource reuse at network edge, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, 2020, pp. 267–276.

[63] B. Gao, Z. Zhou, F. Liu, F. Xu, Winning at the starting line: Joint network selection and service placement for mobile edge computing, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, 2019, pp. 1459–1467.