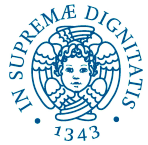


Università di Pisa
FACOLTÀ DI INGEGNERIA
Corso di Laurea Specialistica in Ingegneria Informatica

A reputation-based mechanism to mitigate host misbehaviors in DTNs

Relatori:
Prof. Gianluca Dini
Prof. Giuseppe Anastasi

Candidato:
Gabriele Zannerini

ANNO ACCADEMICO 2009/2010

Abstract

Delay Tolerant Networking (DTN) is a network paradigm designed for disconnected networks. Message delivery in DTNs relies on the mobility of *carriers*, hosts that carry messages from a network partition to another. Context-Aware Adaptive Routing (CAR) is a routing protocol for DTNs with the aim to select the carrier with the highest chance of successful message delivery. CAR relies on the assumption that all hosts in the network are collaborative, i.e. that cooperate in the message forwarding process. In real-life environments hosts can not cooperate in such process and endanger communication among partitions. We propose RCAR, a decentralized approach based on reputation aimed to detect and exclude misbehaving hosts from the network. Simulation tests made on a human mobility model show that RCAR increases the message delivery probability of CAR in presence of misbehaving carriers.

Riassunto Analitico

Delay Tolerant Networking (DTN) è un paradigma di rete progettato per reti caratterizzate da elevati ritardi e frequenti disconnessioni, che si prefigge di permettere la comunicazione fra dispositivi posizionati in partizioni di rete differenti. A tal fine viene sfruttata la mobilità dei cosiddetti *carriers*, dispositivi che trasportano fisicamente un messaggio da una partizione di rete all'altra per conto di un altro dispositivo. Context-Aware Adaptive Routing (CAR) è un protocollo di rete per DTN che seleziona il carrier avente la più alta probabilità di successo nella spedizione di un messaggio verso la destinazione. Il corretto funzionamento di CAR fa affidamento sulla collaborazione di tutti i carriers nel processo di consegna di un messaggio. In sistemi reali i carriers possono però non partecipare a tale processo, mettendo a rischio il processo di consegna dei messaggi. Questo documento presenta RCAR, un meccanismo di sicurezza basato sul concetto di reputazione volto ad individuare ed escludere i carriers non collaborativi dal processo di consegna dei messaggi. I risultati di simulazioni effettuate utilizzando un modello di mobilità realistico dei dispositivi mostrano come RCAR aumenti la percentuale di messaggi consegnati correttamente rispetto a CAR in presenza di carriers non collaborativi.

Acknowledgements

Non vi è spazio per ringraziare tutte le persone che hanno contribuito al raggiungimento di questo inaspettato obiettivo, dovrei riempire almeno tante pagine quante sono quelle di questa tesi e forse mi scorderei qualcuno... quindi, prima di tutto, volevo ringraziare tutti coloro che mi hanno dato la forza di non mollare ed arrivare fino in fondo.

Vorrei partire con il ringraziare il prof. Gianluca Dini per avermi dato la possibilità di svolgere questo lavoro di tesi.

Non vi sono invece parole per ringraziare in modo appropriato il mio tutor, Angelica Lo Duca, per l'aiuto, l'infinita pazienza e la disponibilità durante lo svolgimento di questa tesi: grazie di cuore!

Grazie anche a tutti i miei familiari per aver sempre creduto in me. In particolare devo ringraziare mia madre, che durante questi anni ha sempre sopportato i miei sfoghi ed ha fatto per me tanti sacrifici, senza che io le abbia mai fatto capire quanto è stata e quanto sarà importante per me.

Un ringraziamento a tutti i miei colleghi per la compagnia e l'aiuto ricevuto nel superamento di tanti esami: fra tutti vorrei ringraziare sentitamente Marco Tiloca, Valentino Stopponi ed Alessio Rossi. Se sono arrivato fino in fondo è anche grazie a voi.

Infine vorrei ringraziare Sabrina, per essermi stata vicino ed avermi aiutato negli ultimi 6 mesi, ma soprattutto per avermi fatto capire che nella vita ci sono cose più importanti del lavoro. Ora che mi hai trovato, vorrei non mi lasciassi mai più.

Contents

List of Figures

List of Tables

Chapter 1

Delay Tolerant Networks

1.1 Introduction

Ubiquitous computing is a human-machine interaction model, first envisioned by Mark Weiser in several papers [44] [43] in early 1990's, in which one of the objectives is communication *everytime* and *everywhere*. The last decade saw impressive enhancements in microelectronics industry, making devices cheaper, portable and lightweight, so that they became accessible to a wide range of people and not only military companies or computer researchers. With little steps, we are going towards Weiser's dream.

Unfortunately, the Internet infrastructure did not grow up so quickly as the devices industry, especially in environments away from big cities or developed areas, in which the lack of coverage makes possible communication only to devices in the near proximity, typically using wireless channels.

Mobile ad-hoc networks [20] (MANETs), first developed for military applications in 1970's, allow communication on a hop-by-hop basis. Every host acts as a router for the other hosts, in a totally decentralized way. Ad-hoc networks allow to construct disconnected networks, each of them having local connectivity but no access to the "global" Internet. Starting from the work in Interplanetary Internet [9], which envisioned a *network of Internets* (the basic idea was to enable communication between planets having Internet-like connectivity in within), a new networking paradigm has been designed, named

Delay-Tolerant Networking (DTN) [7] [14]. DTN is an approach to a computer network architecture capable to allow communication among heterogeneous networks that may lack of continuous connectivity and suffer possibly long delays and high bit error rates. This is the case of many environments such as war battlefield, wildlife monitoring in unmanned areas, communication between isolated villages in remote areas, intelligent highways, etc. To enable communication among possibly heterogeneous networks, the DTN architecture defines an *overlay* to allow transparent communication in presence of different transport (or other) layers of the networks it interconnects.

DTN relaxes the traditional end-to-end constraints, the need of a permanent link between sender and receiver of a message, and the need of a common protocol stack among devices in order to be able to communicate.

Consider Figure 1.1, representing an example of Delay Tolerant Network with the aim to provide communication between isolated villages and a big city. Assume that communication is possible using a telephone line, a satellite connection and an autobus equipped with a wireless device, periodically travelling across the villages. The telephone line could be active only for a limited number of hours per day, the satellite could appear at pre-defined intervals (it rotates around the world globe) and the autobus crosses a village a few times per day.

It is obvious that in this scenario the same connection between two entities can experiment different performance, depending on the available channel: satellite connection permits to reach the highest speed and the lowest delay, but it is available only in limited periods of time, same as the telephone line, with a lower connection speed; the autobus represents instead the slowest channel, and delivers messages in a *store-and-forward* fashion: it collects messages from the villages, store them in a local buffer until it reaches the final destinations and deliver such messages to them.

The satellite, the telephone line and the autobus are also called *contact opportunities* because of their capability to provide sporadic communication opportunities among networks that would have been otherwise disconnected.

A message can be delivered from sender to receiver synchronously or asynchronously. We say that a message is delivered *synchronously* when a

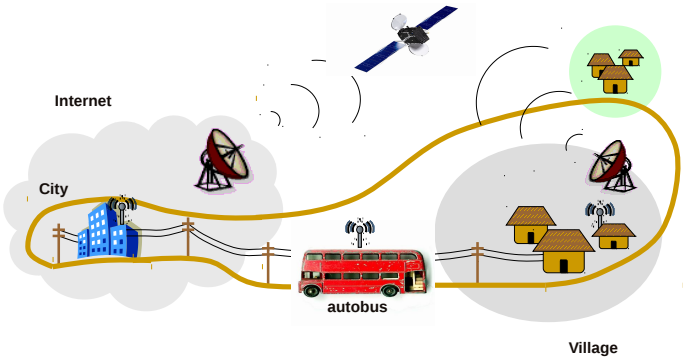


Figure 1.1: Example of Delay Tolerant Network, reproduced from [15].

connected path exists between sender and receiver, because they are placed in the same network partition or because the current connectivity graph permits to immediately reach the receiver on a hop-by-hop basis. When a connected path between sender and receiver does not exists, a message is forwarded *asynchronously*.

Device mobility is exploited to allow communication between partitions that otherwise would have been disconnected.

A wide range of traditional Internet applications can be used in DTNs also, for example e-mailing, virtual bulletin boards or newsgroups. It is obvious that some of the most common Internet applications, such as audio and video streaming, are not conceivable because of the intrinsic bandwidth and delay constraints typical of DTNs.

1.2 Classification of DTN routing protocols

Synchronous communication, i.e. communication between hosts belonging to the same partition, occurs using a standard routing protocol typical of MANETs. Synchronous routing protocols can be divided in *reactive* and *proactive* routing protocols.

Reactive routing protocols, such as AODV [33] and DSR [21], search for a path between sender and receiver of a message on a *per-need basis*, i.e. only when there is the need to send a message. Basically, a source host sends a *request* message to its neighbors for the receiver, and the receiver itself or one of the intermediate hosts answer with a *response* message.

In proactive routing protocols, such as DSDV [32], each host keeps a routing table for each known destination. Such routing table is updated periodically depending on the value of one or more given *metrics*, typically the number of hops.

It is worth to note that in DTNs the concepts of *routing* and *forwarding* are strictly linked together, since routes are actually built in the same time messages are forwared. So, in the following we will use the terms routing and forwarding interchangeably.

With respect to asynchronous routing, Pelusi et al. in [31] defined a taxonomy of routing/forwarding techniques for DTNs, graphically represented in Figure 1.2. They first divided protocols designed for completely flat ad hoc networks from protocols designed for ad hoc networks with some form of infrastructure that hosts exploit to opportunistically forward messages.

The infrastructure-based family of protocols is based on hosts with the only aim to provide some sort of infrastructure for the other hosts in the network. This family of protocols can be further divided in protocols based on a fixed infrastructure and protocols based on a mobile infrastructure depending on the mobility of the hosts forming the infrastructure.

The family of delay tolerant routing protocols without infrastructure can be further divided in dissemination-based protocols and context-based protocols. Context-based protocols exploit some information about the network in terms of host meetings' history, user's characteristics, network topology, etc. Dissemination-based approaches do not rely on any kind of information and they only spread a certain number of copies of the same message in the network in the hope to find a host that will eventually carry it until the final destination.

Following subsections analyze such protocol categories, providing some approaches for each of them.

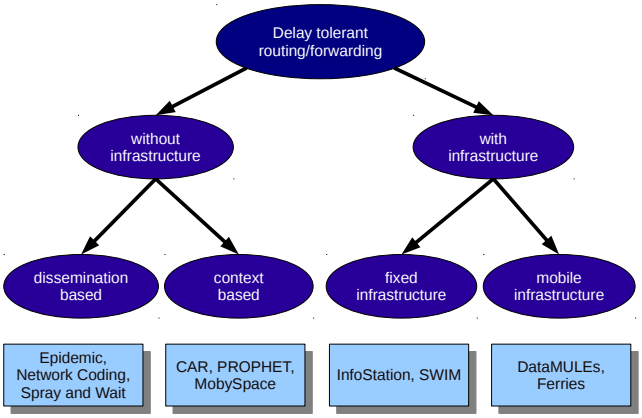


Figure 1.2: Taxonomy of routing/forwarding techniques for Delay Tolerant Networks, reproduced from [31].

1.2.1 Routing with infrastructure

In routing protocols based on a fixed infrastructure the territory is supposed to be scattered of base stations providing access to the Internet or to a high-speed LAN. Hosts whishing to deliver a message on the Internet or to another host that is not reachable directly or quickly move toward the nearest of those base stations and deliver the message to them. Examples of such family of routing protocols are the Infostation model [16] and the Shared Wireless Infostation Model (SWIM) [38].

- In the *Infostation model*, infostations are base stations with limited wireless coverage, and users in proximity can exploit their resources (typically Internet access). The communication paradigm allow only host-Infostations data exchange. This model may cause temporary disconnections in case of network outages, and users have to reach another Infostations to continue communication.

- *SWIM* represents and extension of the Infostation model, since the communication paradigm allows also data exchanges between hosts, not only between hosts and infostations.

A mobile infrastructure is composed by some hosts, typically high-performance devices, moving around the territory following pre-defined or arbitrary routes. They act as data-gatherers, collecting messages from the hosts they pass by. Examples of such family of routing protocols are the DATA-MULE system [37] and the Message Ferrying approach [49].

- The *DATA-Mule system* defines a hierarchy of hosts, according to the role they have in the routing/forwarding process: at the lower level there are sensors which periodically perform data sampling, whose data are collected by *MULEs*, mobile data-gatherers moving around the territory and considered at the intermediate level of the hierarchy; at the upper level there are the wired access points (APs), retrieving data from MULEs and storing/processing them.

- In *Message Ferrying* some hosts, called *ferries*, move around the territory collecting messages from source hosts. Source hosts can either schedule ferries' periodical movement patterns and send messages whenever they come within their transmission range, or explicitly requesting the ferries to change their trajectory and move toward the requester source host itself.

1.2.2 Routing without infrastructure

This family of protocols does not rely on any existing infrastructure, basing message delivery on hosts' cooperation. In dissemination-based approaches, selection of the best path towards the message receiver is made spreading multiple copies of the same message among the network, while in context-based routing it is made using some piece of information about the network or hosts' environment.

Dissemination approaches are based on the simple idea to broadcast a message to a high number of hosts in the network, in a way to increase the chance of successful delivery. The choice of such hosts is purely random. This approach well performs in terms of both delivery rate and delay, especially in highly mobile environments. This form of uncontrolled flooding has obvious

drawbacks, e.g. medium contention and network congestion. Limits to messages, in terms of maximum number of hops it can traverse or total number of copies present at the same time in the network are typically imposed. Examples of dissemination approaches are the epidemic routing protocol, the network-coding-based routing and spray-and-wait.

- The *epidemic routing* [40] follows the principle of the spreading of a disease, by means of pair-wise contats (*infections*) between hosts. Each host keeps a list (in form of hash table) of the messages stored in its local buffer. Every time a host gets in touch with another host, an *anti-entropy session* is initiated: the two hosts exchange first their respective message lists (by means of a *summary vector*), then the messages that are each other's missing in order to have equal local buffers. For example, if hosts A and B start an anti-entropy session, A sends the messages present in its local buffer that B has not (A can check it via the summary vector received from B), and B does the same, sending messages present in its local buffer that A has not.

- *Network-coding based routing* [45] is an epidemic-like approach, in which packets generated from the source hosts are combined together (*coded*) at the intermediate hosts; receiver hosts *decode* the packet and extract the original message. To give an example, let A, B, C be the only three hosts of a small network. Let host A generate the information "a" and the host C generate the information "c". Suppose the information produced needs to be known by all the hosts of the network. Hence, host A and host C send their information to host B, then host B rather than sending two different packets "a" and "c" respectively, broadcasts a single packet containing "a" *xor* "c". Once received "a" *xor* "c", both hosts A and C can finally infer the missing information, i.e. host A can infer "c" and host C can infer "a".

- *Spray and Wait* [39] exploits no information about past host meetings or network topology, it only spreads a certain number of copies of a message. The protocol works as follows. Two phases characterize messages delivery: *spray phase* and *wait phase*. During the spray phase L

copies of the same messages are spread over the network both by the source host and the hosts that first received a message copy from the source itself. Then, in the wait phase, each host holding a copy of the message does nothing but simply storing its copy in the local buffer until it eventually comes within reach of the receiver and delivers the message to it. Spray can be performed in several ways, and the value of L can be tuned in order to achieve the desired performance in a specific scenario.

Context-based routing protocols exploit information about the context in which hosts are operating (e.g. workplace, institution, habits) the network topology or the history of past encounters. The common idea behind such family of protocols is to limit message spreading by carefully selecting the next hop(s) with the highest chance of successful delivery. Hence, what basically changes among these approaches is the method used to calculate delivery probabilities. Examples of context-based protocols are CAR, PRoPHET and MobySpace routing.

- In *Context-Aware Routing* (CAR) [30] each host computes its *delivery probability* (i.e. the probability to reach a given host) for each known destination and exchange such values with the other hosts belonging to the same network partition. Delivery probabilities are calculated using multi-attribute utility theory over a set of context information (change rate of connectivity, colocation with other hosts, residual battery level, etc.). Forecasting techniques based on Kalman filter theory [**?**] are used to predict future values. The predicted values are actually distributed to the other hosts than the current ones, for a better estimation of such probabilities.

- In the *Probabilistic Routing Protocol using History of Encounters and Transitivity* (PRoPHET) [25] hosts keep a table of delivery probabilities for each known destination. Delivery probability is increased every time a host encounters a given destination and it is decreased according to an aging function. Transitivity is defined as the capability of a host to

act as carrier for a message on behalf of another host, and it is also used to update (increase) the delivery predictability. Message delivery works as follows. Every time two hosts meet, they exchange their summary vectors, together with their delivery probability for the destinations hosts of that messages. If a host finds that it has a higher delivery probability with respect to the other host, requests the message to it. For example, assume two hosts A, B get in touch, and assume host B has to send a message to a third host, say C. Upon exchanging their summary vectors, if host A founds that it has a higher delivery probability to reach host C tha host B, then host A requests the message to host B.

- In *MobySpace routing* [24], each host builds a high dimensional Euclidean space in which each axis represents a contact opportunity between two hosts and the distance measuring the probability that the two hosts meet. So, two hosts are close in the *virtual space* when the contact probability is high. This approach requires full knowledge of all the hosts traversing the network and tracking of their movements, unless restrictions on mobility can be applied (e.g. hosts have to follow pre-defined routes instead of arbitrary ones).

Hereafter we focus on CAR, because of its compromise between achieved throughput and resource consumption. Furthermore, it is based on a human mobility model and it represents a framework approach for the evaluation of delivery probabilities, with the possibility to be implemented in real systems.

1.3 Security issues in DTNs

We can envision ubiquitous computing as the realization of a worldwide MANET. In this sense, DTN inherits the architectural model from MANETs but, unfortunately, it also inherits all the security issues related to MANETs. This Section will refer to DTNs whilst analyzing MANET's security issues.

Before discussing such security issues, we first give a review of the requirements that a network, be it wired or wireless, should accomplish to be

defined *secure*. These requirements are general for network communication, and can be summarized in five categories:

availability A given service, in this context the successful routing/forwarding of messages, should be always operative despite the presence of attackers.

authentication Users should be sure of each other's identity, avoiding an attacker to impersonate a legitimate user.

data confidentiality Only sender and receiver of a communication have to know the content of messages exchanged, while intermediate hosts should not.

integrity Message content should not be altered during its path from sender to receiver.

non-repudiation Users can not deny to be the originator of a message previously issued.

Attacks on DTNs aim to break one or more of the security requirements listed above. Such attacks can be further cathegorized according to their origin or to their nature [13]; according to the origin of the attack, threats can be classified in:

- *external* attacks, in which the attacker is an entity that does not belong to the logical network, or is not allowed to access to it.
- *internal* attacks, in which the attacker is assumed to be a compromised or malicious host, acting as a host belonging to the network under attack. They constitute the most common and dangerous category of attacks.

According to the nature of the attack, threats can be classified in:

- *active* attacks, in which an attacker actively participates in disrupting the normal operation of the network services. A malicious host can modify packets or introduce false information in the network. It confuses routing procedures and degrades network performance.

- *passive* attacks, in which an attacker snoops the data exchanged without altering it. The attacker mainly eavesdrops the data packets in the network without doing any active operations, or just refuses to execute the requested function. The goal of the attacker is to obtain information that is being transmitted, thus violating the data confidentiality. Since the activity of the network is not disrupted, these attackers are difficult to detect. Passive attacks are usually preliminar of an active attack.

Hereafter we consider internal attacks, unless it is explicitly said.

Attacks can be performed at different layers of the protocol stack: physical, MAC, routing. Attacks on the physical layer are targeted to make the hardware devices or the wireless channel useless. They can be summarized as follows:

attacks on the physical device They may lead to simply *destroy* the device itself or damage it in order to make it useless, e.g. using *signal jamming* to cause interference. They aim to disrupt service availability. More subtle attacks are *stealing and/or cloning* the devices in order to retrieve condifential data or impersonate the user that possesses such device. In this case the security requirements to guarantee are authentication and data confidentiality.

eavesdropping Is a passive attack, in which the attacker switches its radio interface in promiscuous mode with the aim to eavesdrop packets in transit. Such packets can be used for immediate attacks, such as packet dropping, or more sophisticated attacks based on retriveing confidential data. It is still practically impossible to detect this attack. User-level authentication and data confidentiality have to be taken into consideration to constrast such attack.

Attacks at the MAC layer can be summarized as follows:

address spoofing It is a passive attack usually mounted in preparation of other more dangerous attacks. It aims to retrieve MAC addresses of

honest hosts from data packets and clone them to mount for example an *impersonation* attack. Such attacks puts on risk host authentication.

unfairness Kyasanur and Vaidya in [23] described a possible attack at the MAC layer leading to an unfair use of the medium contention alghoritm. The attack is based on a malicious modification of the backoff distribution of the CSMA/CA protocol [11] aimed to increase the probability to obtain the channel access with respect to the other hosts. An unfair use of the wireless channel may lead to disrupt service availability.

Attacks at the routing layer is the category most widely analyzed in the litetature. The routing process and packet forwarding are functionalities very connected in DTNs, especially when talking about security. In fact, before mounting an attack on the message forwarding process, an attacker has to become member of the network, i.e. it has to somehow disrupt the routing process.

Threats deriving from the routing/forwarding process can be summarized as follows:

impersonation An attacker may assume the identity of another user of the network. This attack hides the attacker from the other hosts of the network, allowing him to mount one of the attack listed above. Impersonation attacks pose on risk non-repudiation, and user-level authentication techniques should be adopted to contrast it.

route fabrication An attacker may inject false route information, e.g. routing messages with zero hop distance to a given host, in order to alter routing tables of the hosts in the network. The other hosts think that the attacker itself is the next hop of the shortest path toward the receiver of a message: the attacker will start receiving a high number of messages. This attack can be thought to provide only misfunctioning of the routing process (i.e. service unavailability) or it may be considered preparatory for a packet modification/dropping attack.

sleep deprivation torture The attacker floods the network with false information, that can be routing messages or data packets, with the only

aim to cause continuous information processing on the devices and consuming their resources. This attack puts on risk service availability.

packet modification/dropping The attacker may modify message contents or simply destroy them. Availability, integrity and data confidentiality have to be taken into consideration to contrast such attack.

selfishness it is a passive attack, in which a host refuses to participate in the routing process or the packet forwarding. A host could not have interest to forward a packet on behalf another host because for example it may want to preserve its resources (especially battery power). This is not an intentional attack but a selfish misbehavior. However, selfishness poses on serious risk service availability, that is one of the most important security requirements.

We focus our attention on the detection and exclusion from the network of malicious carriers with particular reference to the CAR protocol, described in Chapter 3. We will consider two different types of misbehaviors. We will call them *blackhole carriers* or *selfish carriers* depending on the misbehavior acted:

- Blackhole carriers distribute to their neighbors a large utility function value (close or equal to 1), so that the malicious carrier increases the probability to be chosen for message forwarding. A blackhole carrier can put on risk the security of not only the partition in which it is currently located, but also of the entire network.

- Selfish carriers instead distribute to their neighbors a low utility function value (close or equal to 0), so that they are never chosen (or chosen with a very low probability) as carriers for messages. They exploit instead network capabilities, using well-behaving carriers for the delivery of their own messages while saving their own resources.

A message that reaches a malicious host can be at least read, but also compromised via modification or dropping. Threats presented above are

very dangerous for both synchronous and asynchronous communication, especially in CAR, a single-copy DTN routing protocol. In fact, if a message is dropped, it is irremediably lost; this is instead not true for multi-copy routing protocols, e.g. Epidemic Routing. Chapter 5 discuss advantages and disadvantages of the different approaches.

We modeled blackhole carriers dropping all the packets for which they are not final receivers. Selfish carriers do not drop packets, they only make very unlikely the probability to be selected as message carriers.

Chapter 2

Mobility models

When analyzing the performance of a protocol for mobile networks the protocol should be tested under realistic conditions. Such conditions include, but are not limited to, devices' transmission range, limited buffer space for the storage of messages, representative data traffic models, and realistic movements of the mobile hosts, i.e. a *mobility model*.

A mobility model aims to produce movements for the mobile hosts that somehow reflect the reality. Changes in speed and direction must occur and they must occur in reasonable time slots. For example, we would not want a mobile host to travel in straight lines at constant speeds throughout the course of the entire simulation because they actually would not frequently travel in such a restricted manner in a real-life scenario.

There are two types of mobility models used in the simulation of networks: trace-based and synthetic models.

Trace-based models are those mobility patterns that are observed in real life systems. Traces provide accurate information, especially when they involve a large number of participants and an appropriately long observation period. An example of trace-based model can be found in [10], describing the results of the measurement performed by the Intel Research Laboratory in Cambridge and results coming from other publicly available data sets ([19] and [28]). The basic idea is to log movements of a group of volunteers for a certain period of time. Such logs are possible by assigning to each vol-

unteer a sensor equipped with a radio interface (Bluetooth) [10], asking to always keep it whenever they are. The devices log inter-contacts among the volunteers and their approximate duration. Another technique is to collect data coming from laptops or PDAs connected to access points in the area of interest [19], [28]. In this case is made the assumption that a device can communicate with all the other devices connected to the same access point. In [10] the authors compare the results of such experiments, showing evident similarities between the pattern movements collected by the different groups.

However, network environments such as DTNs and ad hoc networks are not easily modeled if traces have not yet been created. In this type of situation it is necessary to use synthetic models. Synthetic models attempt to realistically represent the behaviors of the mobile hosts without the use of traces. Here we present two synthetic models: the Random Waypoint model and the Community-based mobility model.

2.1 Random waypoint model

Random Waypoint model (RWP) [21] is a commonly used synthetic model for host mobility. It is an elementary model which describes the movement pattern of independent hosts by using a few input parameters:

1. pause time (t_{pause})
2. minimum speed (*minspeed*)
3. maximum speed (*maxspeed*)

In RWP a host is allowed to move inside a convex area of given size, in which it moves independently on the others. The area can be thought as divided in *regions* of a given size. The algorithm for the generation of movement patterns for each host can be summarized as follows:

0. initially, each host is somehow placed inside a region of the area.
1. the host remains in its location for a time t_{pause}.

2. after the pause time, the host is assigned to move to a new location randomly selected. The host starts moving along a straight line toward the given target with a speed randomly chosen in the interval $[minspeed, maxspeed]$.

3. upon reaching the target location, the algorithm is repeated.

This model has the advantage to be very easy to implement, and it has been widely used in the scientific community in the past to test communication protocol for MANETs.

The drawback of such simplicity is the failure in providing movements that reflect reality. We can resume the reasons of such failure in the following points:

unnatural movements It is easy to argue about the paths being unnatural, because hosts tend to make *zig-zag movements* in the given area. In [29] it has been shown that RWP presents properties very distant from those extracted from real scenarios, such as duration of the contacts between hosts and the inter-contact times.

speed distribution The most common problem with simulation studies using RWP is a poor choice of speed distribution [35], e.g. a uniform distribution $U(0, V_{max})$. Such distribution leads to a steady-state level in which the average speed is lower than the initial average speed [47]. To give an example, using an area of $1500m \times 500m$ and speed range of $(0, 20]m/s$, if a destination is chosen 1000m away and the speed is chosen to be 0.1 m/s, then the travel time would be 10000 seconds. If hosts do reach the destination they will be assigned another possibly higher random speed, but hosts like this can be "trapped" to these slow journeys for significant amount of time and therefore dominate the average speed.

In the last years the research community made efforts in providing mobility models that overcome the limititations of RWP. One of such mobility models is the Community-based mobility model, presented in Section 2.2.

2.2 Community-based mobility model

The Community-based mobility model [29] relies on the assumption that, since devices are usually carried by humans, *mobile networks are social networks after all.*

A *social network* [36] can be defined as a network in which entities (e.g. individuals) are tied (connected) by one or more types of interdependency, such as friendship, kinship, beliefs, knowledge, etc. Social networks view social relationships in terms of network theory. In particular, a social network can be represented using weighted graphs, in which nodes represent people and weighted edges model the strenght of the social interaction, the value of which is defined in the range [0,1]; 0 means no interaction, 1 means strong social interaction.

Social networks can also be represented by *interaction matrices*, in which an element $m_{i,j}$, called *interaction indicator*, represents the *social attractivity* between individuals i and j. The matrix is considered symmetric, i.e. both individuals give the same weight to the relationship. This clearly constitutes a semplification, since in many real-life situations there is a strong asymmetry in the weight each part assigns to the relationship, e.g. professor-student, boss-exmployee, actor-fan, etc.

Figure 2.1 shows an example of social network, while Figure 2.2 the interaction matrix representing that social network.

Social networks have been investigated in detail, both in sociology and in other areas, most notably mathematics and physics. They can be described by a set of metrics, measuring the relationships between entities forming the network. Metrics of our interest are *average path length* and *clustering coefficient*. Average path length is defined as the average distance (in terms of hops) between two nodes in the network. Clustering is a measure of the number of interconnections locally to each host; in other words, it measures the "cliqueness" of a node's neighborhood.

Theoretical models have been developed to reproduce the properties of these networks, such as the so called *small worlds model*, proposed by Watts and Strogatz in [42]. A social network that presents small worlds character-

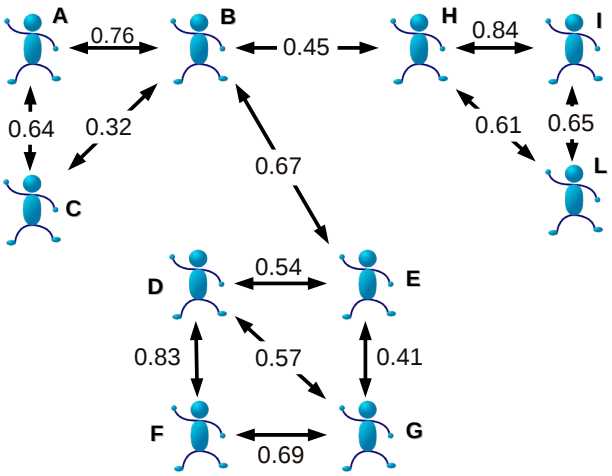


Figure 2.1: Example social network, reproduced from [29].

istics presents a low average path length and a high clustering coefficient.

2.2.1 The Caveman model

The Community-based mobility model is constructed using the so-called *Caveman model* [42]. The model is started from building K graphs, locally fully connected but disconnected each other (representing communities of men living in isolation). Individuals of one community are closely connected, whereas populations belonging to different caves are initially not connected. Connections between communities are defined by a dynamic rewiring process, that will be described in the following.

The network is constructed within an area of a given size, divided into squares to form a grid. We indicate with $S_{p,q}$ the square in position (p, q). The number of rows and columns are inputs of the mobility model. Each community is placed in a different square of the grid and composed by K/N individual, where N is the total number of individuals composing the network. Figure 2.3 represents an example of Caveman model with three disconnected caves, while Figure 2.4 show how the communities defined in the example above can be placed on a 3x4 grid. The dimension of the grid is configurable by the user and influences the density of the nodes in each square.

$$M = \begin{bmatrix} 1 & 0.76 & 0.64 & 0.11 & 0.05 & 0 & 0 & 0.12 & 0.15 & 0 \\ 0.76 & 1 & 0.32 & 0 & 0.67 & 0.13 & 0.23 & 0.45 & 0 & 0.05 \\ 0.64 & 0.32 & 1 & 0.13 & 0.25 & 0 & 0 & 0.15 & 0 & 0 \\ 0.11 & 0 & 0.13 & 1 & 0.54 & 0.83 & 0.57 & 0 & 0 & 0 \\ 0.05 & 0.67 & 0.25 & 0.54 & 1 & 0.2 & 0.41 & 0.2 & 0.23 & 0 \\ 0 & 0.13 & 0 & 0.83 & 0.2 & 1 & 0.69 & 0.15 & 0 & 0 \\ 0 & 0.23 & 0 & 0.57 & 0.41 & 0.69 & 1 & 0.18 & 0 & 0.12 \\ 0.12 & 0.45 & 0.15 & 0 & 0.2 & 0.15 & 0.18 & 1 & 0.84 & 0.61 \\ 0.15 & 0 & 0 & 0 & 0.23 & 0 & 0 & 0.84 & 1 & 0.65 \\ 0 & 0.05 & 0 & 0 & 0 & 0 & 0.12 & 0.61 & 0.65 & 1 \end{bmatrix}$$

Figure 2.2: Interaction matrix representing the social network in Figure 2.1.

To define the rewiring process, we associate to each host a couple (id, T_i), where id is the unique host identifier and T_i the target the host is assigned to move. id is fixed, while T_i changes over time. The model aims to build a sequence of targets T_0, T_1, T_2, etc. for each host in the network. T_0 is chosen randomly inside the square associated to the community.

Successive targets are defined as follows. Each square of the grid exerts a certain *social attractivity* for a host i. Let $C_{S_{p,q}}$ be the set of hosts associated to the square (p,q), then the social attractivity of a square towards host i (SA_{p,q_i}) is defined as:

$$SA_{p,q_i} = \frac{\sum_{\substack{j=1 \\ j \in C_{S_{p,q}}}}^{n} m_{i,j}}{w} \tag{2.1}$$

where w is the cardinality of $C_{S_{p,q}}$, i.e. the number of hosts associated to square $S_{p,q}$. In other words, the social attractivity of a square in position (p,q) towards a host i is defined as the sum of all interaction indicators $m_{i,j}$ representing the relationships between i and the hosts belonging to that square, normalized by the number of them. If $w = 0$, then SA_{p,q_i} is set to 0.

Selection of the target T_i, $i > 0$, is based on the social attractivity of each square toward the host. Here we define two possible ways of selecting such target:

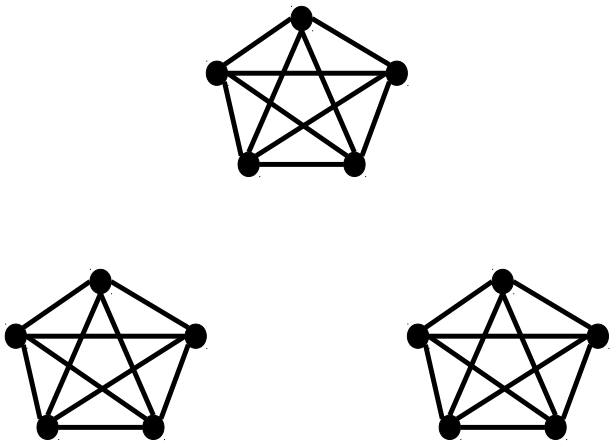


Figure 2.3: Example of Caveman model with three disconnected caves, reproduced from [29].

- *deterministic* selection, in which the host is assigned to move towards a random point inside the square with the highest social attractivity: it could be always the same or change from time to time, e.g. because of insertion or deletion of one or more hosts.

- *probabilistic* selection, proportional to social attractivity: we assign a probability P(s = S_{p,q_i}) of selecting the square $S_{p,q}$ as follows:

$$P(s = S_{p,q_i}) = \frac{SA_{p,q_i} + d}{\sum_{j=1}^{p \times q} SA_{p,q_j} + d} \tag{2.2}$$

  where d is a random value higher than 1 in order to ensure a probability always different from 0. The social attractivity of each community weights the selection probability, i.e. a community that exerts a high social attractivity to a host has a higher probability to be chosen as successive target than a community exerting a lower social attractivity to the same host.

Let us suppose that host A has reached its first goal inside the square $S_{a,2}$.

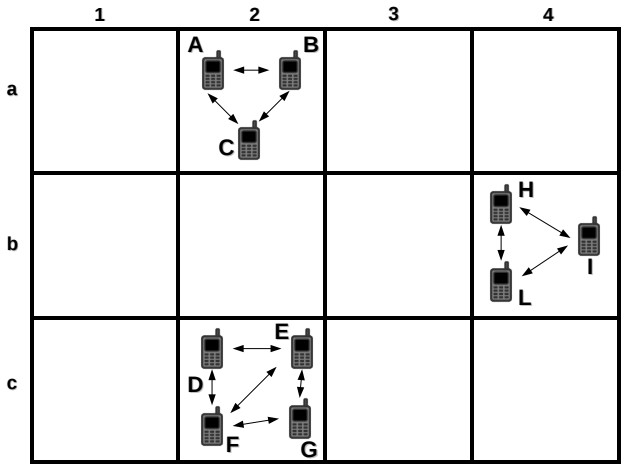


Figure 2.4: Example of initial simulation configuration, reproduced from [29].

Assuming deterministic selection of the next goal, it is chosen by calculating the social attractivities of all the squares that compose the simulation area and choosing the highest. If, say, square $S_{c,2}$ exterts the highest social attractivity (for example, because a host with strong relationship with node A has joined that community), the new goal will then be selected inside that square.

Considering probabilistic selection of the next goal, assume that square $S_{a,2}$ exerts a social attractivity $SA_{a,2_A}$ of 0.30, square $S_{c,2}$ a social attractivity $SA_{c,2_A}$ of 0.50 and square $S_{b,4}$ a social attractivity $SA_{b,4_A}$ of 0.20. The new goal will then be square $S_{a,2}$ with a probability of 20%, square $S_{c,2}$ with a probability of 50% or square $S_{b,4}$ with a probability of 30%.

Chapter 3

CAR

3.1 Introduction

Context-aware adaptive routing (CAR) [30], briefly introduced in Section 1.2.2, is a routing protocol designed for enabling both synchronous and asynchronous routing of messages in DTNs.

When sender and receiver of a message are in the same partition, the message is delivered using a synchronous routing protocol. In CAR a proactive routing protocol, DSDV [32] (Dynamic Source Distance Routing), is used. DSDV will be introduced in Section 3.2.

When direct communication with the receiver is not possible, asynchronous routing is performed. CAR selects the host with the highest chance of successful message delivery, called *delivery probability*. The selected host is called *carrier*.

To understand how a carrier behaves, consider the example represented in Figure 3.1. At time t = t_1, host S wants to send a message m to host D. S cannot reach D using the synchronous routing because no persistent path exists between the two hosts. So, S checks in its routing table whether exists a carrier to reach D or not. Assume S finds that host D can be reached from host B with a delivery probability equal to 0.3, while host A with a delivery probability equal to 0.8. Host A therefore represents the carrier for message m to reach D. S forwards the message m to A, that saves it in its

local buffer. Later, at time t = t_2, A leaves the partition to join the other partition, in which it will deliver the message to the receiver D using the underlying synchronous routing protocol.

Delivery probabilities are evaluated locally on each host based on a set of context attributes and exchanged together with the synchronous routing protocol updates. The values exchanged are used as input for Kalman filter predictors that will provide an estimation of the future value of delivery probability. Hence, each host exchanges predicted values with the other hosts rather than current ones. Details on computation and evaluation of delivery probabilities are provided in Section 3.3.

Design of CAR is based on two assumptions:

1. a host is not aware of its absolute geographical position, i.e. no GPS coordinates are available. A host has information about its local neighborhood only.

2. all hosts in the network cooperate in the message forwarding process.

More formally, we consider the network as a non-connected graph G = (V, E), with vertexes V representing hosts and edges E their connections. We also consider hosts of G placed in an area of well-defined size, partitioned in k disconnected subgraphs G_1, G_2, .., $G_k \subseteq$ G where each subgraph G_i represents a *partition*. Partitions are locally connected, i.e. hosts within G_i are connected each other, but no persistent connection exists with hosts belonging to different partitions.

CAR has been tested using the *Community-based mobility model* [29] to define host movements. Such model has been discussed in Section 2.2. Results of such tests are resumed in Section 3.4.

3.2 Message delivery

In CAR each host mantains a routing table for both synchronous and asynchronous routing.

Synchronous message routing exploits the DSDV protocol. DSDV is an extension of the Distributed Bellman-Ford algorithm (DBF) [6] for mobile

ad hoc networks. In DSDV, each entry of a host's routing table contains the destination host identificator, the host's shortest known distance (expressed in number of hops), and the identificator of the host that is the first hop in that shortest route. To mantain the routing table, each host periodically transmits a routing update to each of its neighbors, containing the information from its routing table. Each host uses the information advertised by its neighbors to update its table, so that each route for each destination uses as a next hop the neighbor that advertised the smallest distance for that destination; the host sets the distance in its table to one (hop) more than the distance received in order to count the neighbor as a hop too. Routing loops (i.e. paths that start and finish on the same host) are prevented by imposing a maximum distance, called *infinite distance*, to each route. Infinite distance is also assigned to a neighbor that is not in transmission range anymore. The main contribution of DSDV with respect to DBF is the use of *sequence numbers* attached on each update message in order to apply routing changes in temporal order.

CAR extends the synchronous routing protocol by adding a couple of entries for each destination in the routing table: the carrier with the highest delivery probability, called *best carrier*, and the associated delivery probability value. A Kalman filter predictor has been also associated to each entry of the routing table. Asynchronous routing information are distributed together with synchronous routing updates. The received delivery probability is always used to update the Kalman filter predictor, regardless the value advertised. The filter is used to achieve a more realistic prediction of the evaluation of the context of a host and to optimize the bandwidth usage. The output of the filter is actually used to drive the asynchronous routing process and distributed to the other hosts in the update process. When a neighbor advertises a carrier having a value of delivery probability higher than the current one, the entry is replaced with the advertised carrier and its delivery probability. The corresponding Kalman filter is reset. A host can also consider itself the best carrier for a given destination. In this case the filter is still used to predict the delivery probability.

When a host, say S, wants to send a message to another host, say D, first

S checks if D can be reached synchronously, i.e. S checks if there is an entry in the routing table containing D as destination with finite distance. S then transmits the message to the indicated neighbor host; each host, in turn, will use its routing table to forward the message along its next hop toward D.

If synchronous communication is not possible, then CAR tries to deliver the message asynchronously. Host S checks in the routing table if it does exists a carrier to reach D: if it's the case, S sends the message to that carrier using the underlying synchronous protocol. Upon receiving the message, the carrier stores the message in its local buffer, waiting to move to another partition and enter in contact either with the receiver D or with a carrier claiming a higher delivery probability for D.

If none of the mechanism can be exploited, the message is stored in the host's local buffer. The host periodically attempts to forward the message toward the receiver following the order described above: first trying synchronous routing and then trying with asynchronous routing; if the routing is still not possible, the host stores the message in the local buffer again. Retransmission interval is a parameter to select accordingly, usually somehow proportional to the rate in which the environment changes, e.g. according to hosts' speed. The number of the retransmissions is another configuration value, tested during the performance evaluation of the protocol (see [30]).

With respect to the process of message forwarding, it is worth to note that:

- a host can decide to keep a message in its local buffer because it considers itself the best carrier to reach a given destination.
- asynchronous message forwarding can occur also between hosts placed in the same network partition, e.g. because an incoming host has not enough routing information to reach a given destination yet.

3.3 Delivery probability

Delivery probability is a value representing the utility of each host to act as potential carrier for a message, its value resulting from the computation of

a utility function over context information. *Context* is defined as the set of attributes describing the aspects of the system that can be used to drive the process of message delivery. Examples of such attributes can be the change rate of connectivity, the colocation with other hosts, the residual battery power, etc. Currently the protocol is based on the evalutation of two context attributes:

- *change rate of connectivity* (CRC): how often a host appears and disappears from the partition, calculated in Formula 3.1.

$$U_{cdc_h}(t) = \frac{|n(t-T) \cup n(t)| - |n(t-T) \cap n(t)|}{|n(t-T) \cup n(t)|} \tag{3.1}$$

  where h is the current host, n(t) is h's neighbor set at time t. The formula yields the number of hosts that become neighbors (appeared) or disappeared in the time interval [t-T, t] (T is a tunable parameter), normalized by the total number of hosts met in the same time interval. In other words, CRC is an indicator of a host's mobility.

- *future host colocation*: likelihood of a host to get in contact (i.e. within transmission range) in the future. The colocation of h with a host i is calculated in Formula 3.2.

$$U_{col_{i,j}}(t) = \begin{cases} 1 & \text{if the host } h \text{ is colocated with host } i \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The objective of the utility function is to maximize the delivery probability using multi-attribute utility theory [22], exploiting the so-called *preemptive methodology*. Each attribute is considered separately, trying to maximize its value over all the others. It may happen that conflicting attributes determine the overall delivery probability, e.g. the host with the highest mobility has the lowest battery level; in such cases there is the need to obtain a trade-off between these values. The problem is solved using *significance weights* assigned to each attribute. Formula 3.3 shows the calculation of the utility function in case of mutually preferentially context attributes, i.e. attributes

that are not in conflict with each other.

$$Maximize \left\{ f(U(x_i)) = \sum_{i=1}^{n} a_i(x_i) w_i U_i(x_i) \right\} \tag{3.3}$$

$U(x_i)$ is the utility function over the context attribute x_i, and w_i are *significance weights* representing the relative importance of each context attribute, the value of which is fixed in advance.

In order to make the function reactive to dynamic changes in the environment, adaptive weights a_i have been introduced. They can be further decomposed in three different components:

$$a_i = a_{range_i}(x_i) * a_{predictability_i}(x_i) * a_{availability_i}(x_i)$$

- $a_{range_i}(x_i)$ adapts the attribute to the range of values assumed, e.g. when battery level decreases, we may want the relative attribute decreasing (linearly or non-linearly).

- $a_{predictability_i}(x_i)$ is used to check whether an attribute can still be predicted from the Kalman filter or not, e.g. because there are not enough fresh updates available in order to give an accurate prediction of the attribute; its value is currently 0 (non-predictable) or 1 (predictable).

- $a_{availability_i}(x_i)$ tells whether the attribute value is currently available or not, and its value is currently 0 (not available) or 1 (available). This weight can be used to add new context attributes "on the fly", simply assuming that all the previous values were 0.

The result of the utility function is used as input for Kalman filter predictors. To understand the reason of the use of predicted future values with respect to current ones, consider the following example. Assume two hosts have been in contact for a long time e.g. for the last two hours: if one of the two hosts moves away, it is reasonable that in the near future they will enter in contact again. So, it would have been a mistake to consider the current colocation value, equal to zero; a host should instead consider the predicted value, that is surely higher than zero.

The model is extendible, providing a framework in which different context attributes can be defined: for example, SCAR [27], an adaptation of CAR for sensor networks, uses also battery level as context attribute, since in a sensor environment the knowledge and prediction of its future value represents an important element for making routing decisions.

3.4 Performance

CAR has been compared with other protocols designed to allow communication in disconnected networks [30]: flooding, epidemic routing, PRoPHET, Spray and Wait, Random Choice.

Flooding represents the simplest approach in message delivery for ad hoc networks, in which each host periodically broadcasts a copy of a message to each of its neighbors. The interval between updates is chosen randomly.

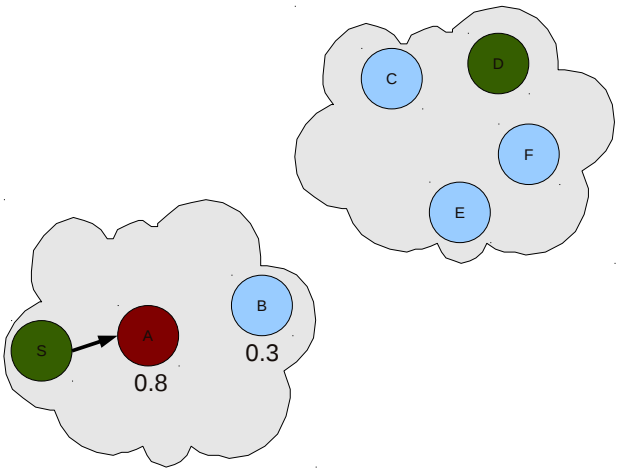
Random Choice is a modified version of CAR, missing of the algorithm for the selection of the best carrier. In other words, in Random Choice a message follows a random path to reach the destination.

It has been shown that:

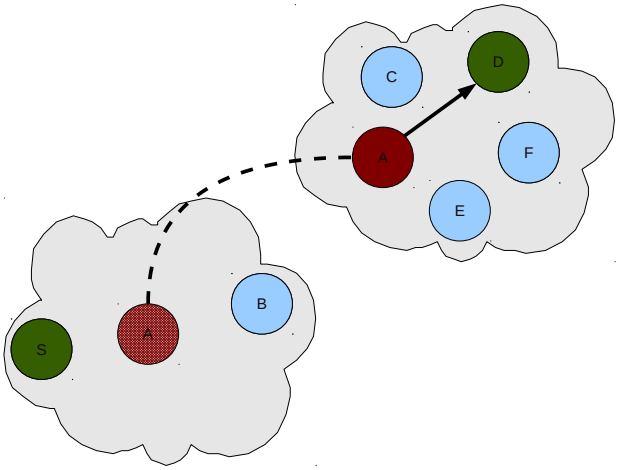
- messages in CAR are delivered with a lower delay when direct communication can be performed. The reason is the use of a synchronous protocol between hosts placed in the same partition.

- CAR well performs with buffers of limited sizes with respect to the other protocols. Multi-copy protocols suffer limitations on buffer sizes, and performance degrade because of overflows with consequent drop of some messages. CAR instead is a single-copy protocol, in which at every time exists only a copy of a message around the network. It implicitly also helps saving resources because of the reduced number of message transmissions compared to the other protocols.

- CAR is suitable for sparse scenarios, in which an accurate selection of the carriers has to be performed. In dense scenarios with no limitation on buffer sizes instead, multi-copy protocols perform better than CAR.



(a) Time t1: sender S passes the message to carrier A



(b) Time t2 > t1: carrier A moves to the other partition and delivers the message to the receiver D

Figure 3.1: Example of asynchronous communication.

Chapter 4

RCAR

4.1 Reputation: concept and definition

In mobile environments such as DTNs host encouters could be very frequent. Such encounters may happen among hosts that already met each other as well as among perfect strangers. As discussed in Section 1.3, several possible misbehaviors can endanger the successful accomplishment of the routing process: we think that a security approach based on reputation management can be used to contrast such misbehaviors.

Reputation approaches are used in many systems, from electronic market places [34] [48] to online communities [17] to measure the trustworthiness of a given person or service. We first give a general definition of reputation. In [26] reputation is defined as:

"*Reputation of an agent* [1] *is a perception regarding its behavior norms, which is held by other agents, based on experiences and observation of its past actions.*"

There are many challenges that a reputation system for DTNs has to cope with, starting from the initial reputation value to assign when two hosts meet for the first time. It could be adopted an optimistic policy, assigning to the new host a high reputation value or it could be adopted a conservative policy,

[1] We will refer to "host" rather than to "agent". However, "agent" represents a more general concept, and the definition is still valid.

assigning to the new host a low reputation value. The latter option is preferred since each newcomer has to "build" its reputation. It also discourages sporadic attacks and attacks based on frequent changes of identity.

A host can assign the initial reputation to a newcomer also evaluating recommendations from other trusted hosts. Nevertheless a host may not want to share a truthful recommendation or do not share a recommendation at all.

Misbehaviors in sharing recommendations can be summarized in the following attacks:

Inactivity Also called *free-rider problem*, in which a host may not want to share reputation information with peers, for example to take advantage of a contended service with the other hosts.

Defame A host reports bogus low reputation values with the purpose to discredit a given service or host.

Collusion Some hosts report reciprocal high reputation values in order to promote each other. This could be preparatory for other type of attacks willing to disrupt network services, e.g. Denial of Service (DoS).

Moreover, in DTNs is not advisable to share recommendations among hosts, in order to not overload the network with extra messages and save resouces.

We will denote with R_{ij} the reputation a host j assigns to another host i. We define the reputation value in the range [0,1], with a value equal to 0 meaning "totally untrustworthy" and a value equal to 1 meaning "totally trustworhty". Reputation updates (increment or decrement) can be periodical or triggered by a certain event or behavior, e.g. a carrier that correctly forwarded a message can trigger a reputation increment on the receiver.

The properties that a reputation system for DTNs has to accomplish are [26]:

Valid The system should be able to distinguish honest from dishonest hosts through the reputation system.

Distributed The system should not assume access to any trustworthy authority or centralized reputation values storage.

Robust The system should be always able to assign truthful reputation values, despite the presence of one of the attacks listed above (inactivity, defame, collusion).

Timely The system should be dynamic and reputation values managed in an up-to-date manner.

Resource-saving The system should take into consideration the limited computational and storage resources of each device.

We propose RCAR [12], a reputation-based mechanism extending CAR. The aim is to mitigate carrier misbehaviors in message forwarding using reputation for the selection of the best carrier for a given destination. At the same time RCAR aims to constrast selfish misbehaviors by assigning network resources proportional to host's reputation. RCAR has been designed with the aim to meet the requirements listed above.

4.2 Design

RCAR constrats both blackhole and selfish carriers. Blackhole carriers distribute a bogus high utility function value $U_c \rightarrow 1$ during routing updates with their neighbors. On the other hand, selfish carriers distribute a bogus low utility function value $U_c \rightarrow 0$.

A blackhole carrier, distributing a bogus value $U_c \rightarrow 1$ for some destinations (if not all), would quickly attract a relevant number of messages coming from hosts wishing to communicate with destinations placed in different partitions. They will put such messages on the hands of the malicious carrier, that will then use them according to its evil purposes: data retrieval, modification or dropping.

In order to contrast blackhole carriers, RCAR introduces the *Local Utility Function* (LUF), an extension of the utility function of CAR. RCAR makes routing decisions on the basis on the LUF value, rather than on the sole

utility function value, as performed in CAR. Let U_c be the utility function distributed by a carrier C. LUF_{cj} will be the LUF host j calculates for a carrier C, defined in Formula 4.1.

$$LUF_{cj} = R_{cj} \times U_c \tag{4.1}$$

RCAR contrasts blackhole carriers as follows. Assume that a carrier C is misbehaving for a reasonable amount of time, it will consequently have a low reputation $R_{cj} \ll 1$ over all the hosts. Its LUF will also be low $L_{cj} \ll 1$. A host j will not choose C as carrier or, at least, the probability that C will be chosen is significantly lowered.

The LUF value is also used in the carrier replacement process. In CAR it is performed as follows. Consider the case of a host receiving a routing update (Section 3.2) from a neighbor distributing the content of its routing table. The host, for each destination in its routing table, checks if the neighbor advertised a carrier with a higher delivery probability than the current one. If it's the case, the advertised carrier replaces the current one. To give an example, assume a host S cannot reach a given host D using the synchronous routing. S knows that the best carrier to reach D is C_1, with a delivery probability of 0.3. Upon receiving a routing update from a neighbor, say host A, S finds that another carrier, say C_2, can reach D with a delivery probability of 0.7. So, S replaces C_1 with C_2 as best carrier to reach D, and the corresponding delivery probability is also replaced. This process advantages a blackhole carrier distributing a high value of delivery probability. In fact, all the hosts will replace their current carriers with the attacker, that will start receiving a high percentage of messages toward receivers not directly reachable using the synchronous routing. The result is that such messages will get compromised, probably they will never reach the receiver.

RCAR modifies the carrier replacement of CAR. In fact, a carrier in the routing table is replaced only if its LUF value is higher than the current one. With reference to the example given above, C_1 is replaced with C_2 only if $LUF_{C_2} > LUF_{C_1}$. In other words, we check whether C_2 has a higher combination of delivery probability and reputation than C_1.

Let now consider a selfish carrier. Since it does not want to be chosen as message carrier, it distributes a low value $U_c \rightarrow 0$ for each destination. In the absence of any security mechanism, the selfish carrier can send messages around the network, exploiting the work of well-behaving carriers while saving its resources.

RCAR contrasts selfish carriers by imposing each host to forward a number of messages proportional to carriers' reputation. Let M_c be the number of messages a host j receives from a carrier C; j will actually forward M_{cj} messages on behalf of C. M_{cj} is defined in Formula 4.2.

$$M_{cj} = \lceil M_c \times R_{cj} \rceil \tag{4.2}$$

Hence, host j will forward a number of messages proportional to carrier's reputation. The principle is to follow a "do ut des" [2] policy, providing an incentive to hosts for well behaving in the routing/forwarding process: the more the host partectipates in the process, the more network resources the host is able to use.

The mechanism well fits in distinguishing selfish carriers from carriers with critical resource levels (e.g. residual battery power). Such carriers are compelled to behave selfishly because of the few resources left, that they want to save. A carrier with low resources will not be able to partecipate in the routing process, but neither to create and send a high number of messages around the network. In this sense, the mechanism can be considered *valid*.

4.3 Reputation update

Reputation update is based on the following principle: when a host receives a message, it means that all the carriers the messages passes through are not misbehaving or, otherwise, the host would have not been received the message. So, the receiver can increase the reputation of the carriers the message passes through. We only need a mechanism to keep track of such carriers.

[2] Latin for "I give, so that you may give" (source: Wikipedia).

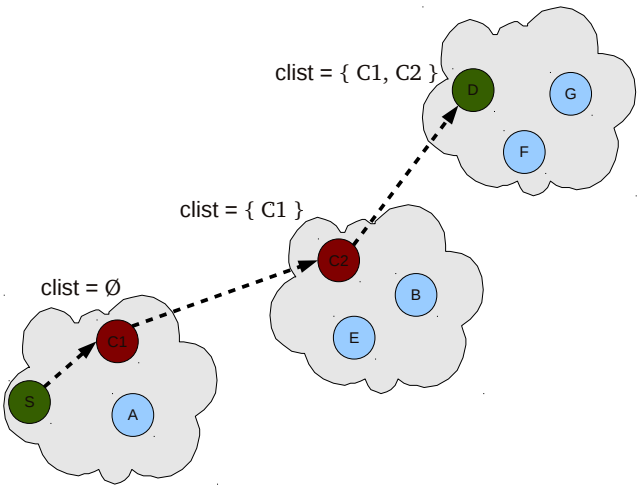


Figure 4.1: Example of *clist* update.

Suppose a host S $\in G_s$ wants to send a message to host D $\in G_d$: S builds a message m = *(mid, p, clist)* where *mid* is the unique identifier associated to each message, p is the message payload and *clist* is the list of carriers the message passes through.

Initially, *clist* is empty. Then, assuming S and D not placed in the same partition, S chooses the best carrier $C_1 \in G_s$ in order to reach D, and forwards m to C_1 using the underlying synchronous routing. C_1, after adding itself to *clist*, leaves G_s and joins G_p that we assume does not contain D. In such a case, if in G_p exists a carrier, say C_2, with a higher probability of reaching D, C_1 chooses C_2 as carrier and sends the message to it. C_2 adds itself to *clist*, leaves G_p and the process is repeated, with the message passing from carrier to carrier until it finally reaches D. Figure 4.1 provides an example of *clist* update.

Upon receiving the message, the receiver D extracts *clist* and increases the reputations of the carriers in it contained, according to one of the techniques we will define in the following.

Note that the integrity of clist must be protected. This can be achieved through the use of digital signatures.

Each node of the network is identified by a unique ID. Furthermore it

has a couple (public key, private key) and a certificate binding its ID to its public key and signed by a Certification Authority (CA) known by all the nodes. The presence of a Public Key Infrastructure (PKI) is not the best solution in a DTN due to the problem of certificate revocation and the communication from a node to the CA. However, certificate revocation can be applied using techniques proposed in [8, 46]. Furthermore, PKI could rely on a low bandwidth cellular infrastructure (e.g. a node could be equipped with a smartphone, to download other nodes certificates), as suggested in [5].

In order to support the clist protection, the message format must be extended. The new format is the following:

$$m = (mid, p, ts, S, D, clist, slist)$$

where ts indicates the timestamp put by the sender and $slist$ indicates the chain of digital signatures. With reference again to Figure 4.1, suppose that the sender (S) wants to send a message to the receiver (D). Furthermore, suppose that the message follows the path S-C_1-C_2-R. The generated messages are:

$S \rightarrow C_1 : (mid, p, ts, S, D, [\emptyset], [s_S])$, where $s_S = < mid, p, ts, S, D >_S$

$C_1 \rightarrow C_2 : (mid, p, ts, S, D, [C_1], [s_S, s_{C_1}])$, where $s_{C_1} = < s_S, C_1 >_{C_1}$

$C_2 \rightarrow D : (mid, p, ts, S, D, [C_1, C_2], [s_S, s_{C_1}, s_{C_2}])$, where $s_{C_2} = < s_{C_1}, C_2 >_{C_2}$

Each node adds itself to $clist$ and to $slist$ (both represented among square brackets [...]), then signs the chain of digital signatures with its own private key.

Replacement of the carriers in the routing table occurs also in case of reputation increments. The principle is that, since a carrier has an increased reputation, probably it can replace one or more entries currently present in the routing table. Reputation increments happen depending on the RCAR implementation considered, when one of the events in Table 4.1 occurs. The value of the increment and the increased reputation depend on the update policy considered.

In order to perform the replacement of a carrier upon a reputation increment, RCAR introduces an auxiliary data structure, called *reachability table* (reachTable). For each known carrier, reachTable stores the most recent routing updates received. It is basically a list storing the carriers that can reach a given destination together with the advertised delivery probabilities. To give an example of such replacement, assume that a host D receives a message *m*. D extracts *clist* and increases the reputation of all the carriers contained in within. Consider the first carrier, say C_1. D checks in its routing table all the destinations with a non null carrier. Assume that a given host X can be reached using a carrier C_x, with a delivery probability of 0.3. Assume also that the reputation of carrier C_x on D is 0.2. So, $LUF_{C_x} = U_{C_x} \times R_{C_xD} = 0.06$. Now, D checks in reachTable if C_1 can reach X. Assume that C_1 can effectively reach X with delivery probability 0.4 and that the reputation of C_1 after the increase is 0.7. So, we have $LUF_{C_1} = U_{C_1} \times R_{C_1D} = 0.28$. The result is that $LUF_{C_1} > LUF_{C_x}$. D replaces in its routing table C_x with C_1 as carrier to reach X. The process is then repeated for each carrier in *clist*. Section 6.2 provides further details on reachTable.

If a carrier misbehaves during the path from sender to receiver, there is no way to detect which is the malicious one. There is neither no way to know if the message arrives at destination or not. In order to cope with misbehaving carriers, each host performs a *periodical decrease* of all the reputations of the carriers the host interacted with in the past. It is worth to note that a periodical reputation decrease is also useful to detect carriers that change behavior along time.

Reputation can be increased and decreased using different functions. Here we propose three different approaches. Other approaches can be defined.

- *linear*: if a host j considers a carrier C well behaving, then j increases R_{cj} by a constant positive value X. Periodically host j decreases R_{cj} by a constant positive value Y. The mathematical formula is shown in 4.3:

$$R_{cj} = \begin{cases} min\{1, R_{cj} + X\} & \text{if does not misbehave} \\ max\{0, R_{cj} - Y\} & \text{periodical decrement} \end{cases} \tag{4.3}$$

- *exponential*: it relies on the idea that if a host is well behaving, then R_{cj} quickly assumes the highest value, while if C does not collaborate or do it infrequently, its reputation is drastically reduced. The formula is shown in 4.4:

$$R_{cj} = \begin{cases} 1 - e^{\alpha R_{cj}} & \text{if does not misbehave} \\ e^{\beta R_{cj}} & \text{periodical decrement} \end{cases} \tag{4.4}$$

  Note that exponents are multiplied by different factors α and β, in order to set the slope.

- *sinus-based*: it is similar to the linear case, but now the reputation converges from 0 to 1 more smoothly. The mathematical formula is shown in 4.5:

$$R_{cj} = \begin{cases} \sin(\frac{\pi}{2} \times R_{cj}) & \text{if does not misbehave} \\ \sin(R_{cj}) & \text{periodical decrement} \end{cases} \tag{4.5}$$

A host j may adopt different policies in order to assign the initial reputation value to a given carrier C: a conservative policy, assigning $R_{cj} = 0$ or an optimistic policy, assigning $R_{cj} = 1$. Intermediate policies can be defined.

4.3.1 Extensions to the update process

The update process described in Section 4.3 limits the knowledge of the good behavior of the carriers that carried a message m to the receiver only. Exploiting the information contained in *clist*, we propose three extensions to the basic technique: acknowledgement, step by step and gossip.

In the *acknowledgement* technique the receiver builds a message *ack = (mid, clist)*, where *mid* is the unique identifier associated with the message and *clist* is the list of the carriers the message passes through. Note that *clist* is the same received in the source message m. The acknowledgement message *ack* is then sent towards the host originating m: it can follow the same path (i.e. the same sequence of carriers) or a different one. However, *clist* is not altered during such path. The host originating m, upon receiving

ack, extracts *clist* and updates reputation values of the carriers contained in within.

Step by step extends the acknowledgment technique with the aim to let as many host as possible to have knowledge of the good behavior of the carriers that carried the message from sender to receiver. A host, upon receiving either a message m or an acknowledgement *ack*, can increase the reputation of the carriers contained in *clist*. In fact, such carriers surely cooperated in message delivery, otherwise the message would have not arrived until the host. A host receiving a message m extracts *clist*, increases the reputations of the carriers contained in within, then add itself to such list and forwards the message towards the receiver. A host receiving an acknowledgement message *ack* does not alter *clist*, it only increases the reputation of the carriers contained in within and then forwards the message towards the receiver (in this case the host originating m). Note that *ack* has the same message identifier of m, in a way that a host that forwards both m and *ack* cannot update twice a carrier's reputation.

The principle behind the *gossip* technique is different instead. In fact, it aims to let know to hosts in receiver's proximity of the good behavior of the carriers contained in *clist*. The neighbors can therefore exploit the presence of the "last hop" carrier to forward messages toward destinations placed in different partitions to it. Upon receiving a message m, the receiver builds a gossip message *gos = (mid, clist)*, the structure of which is equal to an acknowledgement message. The gossip message is then broadcasted to receiver's neighbors (i.e. to hosts at one hop distance). Each host receiving a gossip message only extracts *clist* and increases reputations of the carriers contained in within. The gossip message is no further propagated in order to not overload the network with extra messages.

Table 4.1 summarizes the events that can trigger a reputation increase and makes a connection with the different implementations of RCAR.

	RCAR implementation			
Event	Basic	Ack	Step-by-step	Gossip
message (m) received	√	√	√	√
acknowledgement (ack) received (source host)		√	√	
m or ack received (intermediate host)			√	
gossip (gos) received (receiver's neighbor)				√

Table 4.1: Reputation increments in different implementations of RCAR.

Chapter 5

Simulation and results

We have evaluated the performance of RCAR comparing it with CAR and Epidemic Routing (ER).

We implemented the protocols using the OMNeT++ simulation environment. RCAR extends an existing implementation of CAR, provided by the authors of [30]. Some details about such implementation and our extension are provided in Chapter 5. ER has been implemented following the specifications in [40].

We have evaluated the following metrics:

- *delivery ratio*, defined as the ratio between the number of messages received and the total number of messages sent.
- *average delivery delay*, calculated as average interval between the generation of a message and its delivery to the final recipient.
- *total number of sent messages* around the network, including intermediate transmissions and acknowledgements.

Simulations have been executed using a HP Pavillon dv6-1058el, with a 2.53 Ghz Intel Core2 Duo Processor and 4GB of RAM.

We abstract the scenario at the network level: in fact, the aspects that are of our interest do not depend on the particular MAC protocol considered or the physical device used. However, they may impact on the overall performance of the protocols: we left this problem to future studies.

5.1 Scenario and parameters

We considered two simulation scenarios with common parameters, differing in host sparseness. Scenario A consists of 50 hosts with 30 carriers. Scenario B consists of 100 hosts with 60 carriers. We defined *carriers* as hosts with non null speed, moving among partitions.

Hosts have been divided in 4 groups distributed in a square area of $1km \times 1km$. The area has been divided in a grid of 4 columns and 4 rows, the groups placed at the 4 edges of the area. Carriers have been distributed among the 4 groups. Each carrier moved according the Community-based mobility model (see Section 2.2), with speed uniformly distributed in the range [0-20] m/s and probabilistic selection of the successive target. This distribution of speeds aims at emulating different types of traffic, that can range from pedestrian to vehicular. The rewiring probability was set to 0.1.

We have assumed the use of bidirectional antennas with a transmission range of 200 mt. We assume that the transmission of messages may happen and be completed when two hosts are in radio range. We did not model retransmission of packets.

We have evaluated the performance by sending 5000 messages while analyzing BCs and 10000 messages while analyzing SCs. In the analysis of BCs, sender and receiver of each message were chosen randomly among the "non-carriers" (i.e. the hosts with null speed, that stay during all the simulation in their initial community), in a way to maximize the work of carriers. Note that it may happen that a sender and a receiver can be placed in the same partition: in such a case, the synchronous routing is exploited only. In the analysis of SCs instead, sender and receiver were chosen randomly among all the hosts. We have created a higher number of messages because we wanted to let both carriers and standard hosts to build their own reputation.

The simulation time was set to 5000 seconds. The messages were sent after 300 seconds, in order to allow the convergence of the routing tables after the initial exchanges. The interval between the generation of two subsequent messages was set to 0.8 seconds for BCs and 0.4 seconds for SCs: the last message is therefore created after 4300 seconds, the last 700 seconds are

left for the message forwarding process only. We executed 3 runs per each configuration. We decided to analyze coarse-grained results because this work represents a preliminar study on the possibility of using a reputation-based mechanism for DTN security. More accurate results are not worth the effort and the time of a higher number of runs. Table 5.1 resumes the general simulation parameters and the two scenarios analyzed. Table 5.2 resumes instead the parameters of the mobility model.

With reference to the parameters of CAR, we assigned the same number of maximum retransmissions for each message than those used in [30], i.e. equal to 20 and 40 for Scenarios A and B, respectively. The buffer size was set to 1000 slots, except for the studies made on its influence on the performance of the protocol (Section 5.3). The routing table size was set to infinite, i.e. 50 and 100 for Scenario A and B, respectively. We made this choice in order to alleviate the influence of reachTable in the route replacement process of RCAR. As discussed in Section 4.3, reachTable stores the last received routing update by each host and in this sense it can be considered as a "second-level" routing table. Since it is used during the carrier replacement process to decide the best carrier for a message, using a routing table size lower than the total number of hosts would have been an advantage for RCAR with respect to CAR, since RCAR may have a higher amount of information to drive the routing process.

The retransmission interval of the messages stored in the local buffer was set to 30 seconds. The same value has been selected for both the update of the Kalman filters associated to each entry of the routing table and for the interval between routing updates. Table 5.3 resumes CAR parameters.

We have to point out that we experimented memory allocation problems while simulating ER on the computer we used to execute simulations. Reasons of such memory errors have to be found in the huge number of messages sent around the network by ER (see Section 5.4, Total number of sent messages). So, we had to execute ER with a lower number of messages and a lower message buffer size while simulating Scenario B. However, results of that "scaled" simulations do not impact on the considerations we will make in the following. Table 5.4 resumes the parameters used for simulating ER

Parameter	**Value**	
Total sim. time	5000 sec.	
Number of messages	5000 for BCs, 10000 for SCs	
Message creation interval	0.8 sec. for BCs, 0.4 sec. for SCs	
Initial wait time	300 sec.	
Number of runs per each config.	3	
	Scenario A	**Scenario B**
Number of hosts	50	100
Number of carriers	30	60

Table 5.1: General simulation parameters.

Parameter	**Value**
Area	1km × 1km
Grid	4 rows × 4 columns
Groups	4, one at each edge
Carrier speed	uniform. distrib. in [0-20] m/s
Selection of the next goal	Probabilistic
Rewiring prob.	0.1
Transmission range	200 mt.

Table 5.2: Mobility model parameters.

performance.

With reference to the choice of parameters of RCAR, we had to determine the values for:

- the initial reputation to assign locally on each host.
- the increment and the periodical decrement of reputation.
- the interval of the periodical reputation decrease process.

For simplicity, we considered linear updates only. Exponential and sinus-based updates are left for future work.

We have performed the periodical decrease of the reputation values on each host every 500 seconds. We considered this interval a trade-off between responsivity of the system to the detection of misbehaviors and a too stringent punishment policy. In fact, lower values of such interval lead to a

Parameter	**Value**	
Message buffer size	1000	
Routing update int.	30 sec.	
Message retransmission int.	30 sec.	
Kalman filter update int.	30 sec.	
	Scenario A	**Scenario B**
Routing table size	50	100
TTL	20	40

Table 5.3: CAR parameters.

Parameter	**Value**	
Message retransmission int.	30 sec.	
	Scenario A	**Scenario B**
Number of messages	5000	2500
Buffer size	1000	500

Table 5.4: ER parameters.

situation in which a well-behaving carrier cannot build its own reputation because it is decreased straight after. On the other hand, higher values of such interval lead on a too slow system in detecting and punishing misbehaving carriers. We left for future work a more specific analysis of such parameter.

We have evaluated the three extensions of RCAR: acknowledgement (RCAR-Ack), step-by-step (RCAR-Step-by-step) and gossip (RCAR-Gossip). RCAR-Ack has been evaluated with three different set of initial reputation (R_0), increment (R^+) and decrement (R^-), as explained in Section 5.2.

Blackhole carriers (BC) and selfish carriers (SC) have been considered

	Reputation		
RCAR implementation	R_0	R^+	R^-
RCAR-Ack-1	1	0.1	0.05
RCAR-Ack-2	0	0.5	0.5
RCAR-Ack-3	0	0.1	0.05
RCAR-Step-by-step	0	0.1	0.05
RCAR-Gossip	0	0.1	0.05

Table 5.5: Reputation values for the different RCAR implementations.

separately.

5.2 Choice of R_0

We have investigated the influence of the initial reputation value R_0 on the performance of the system. We have noted that all the implementations of RCAR have the same behavior, so hereafter we consider only RCAR-Ack.

We have evaluated the delivery ratio and the average delay of RCAR-Ack in scenario A by varying R_0 from 0 to 1, using a step of 0.2, in presence of an increasing percentage of BCs. An analysis could be done on SCs, giving similar results.

The reputation increment R^+ was set to 0.1, while the reputation decrease R^- was set to 0.05. Remaining configuration parameters were the same reported in Table 5.1, 5.2 and 5.3.

Delivery ratio. Figure 5.1 shows the delivery ratio measured in presence of an increasing percentage of BCs, varying the initial reputation value. With respect to the number of BCs, the delivery ratio degrades with the number of BCs. This will be discussed in the next sections.

With respect to R_0, we note a general behavior, independently on the number of BCs present in the network: the delivery ratio decays as soon as $R_0 > 0$, then it tends to remain at the same value.

The reason of such decay is that RCAR has a warm-up phase, in which $R_{ij} = R_0 \forall i, j$. The warm-up phase ends after 800 seconds, i.e. the initial wait time (300 seconds) plus the reputation table update interval (500 seconds). During the warm-up phase, if $R_0 > 0$, a BC is not recognized so it is able to attract a relevant number of messages and discard them. If $R_0 = 0$, instead, a host trusts only carriers that correctly forward a message and a BC is not able to attract messages. In other words, before the protocol converges, the 12,5% of total messages are sent considering trusted all the carriers. This means that RCAR works better if $R_0 = 0$.

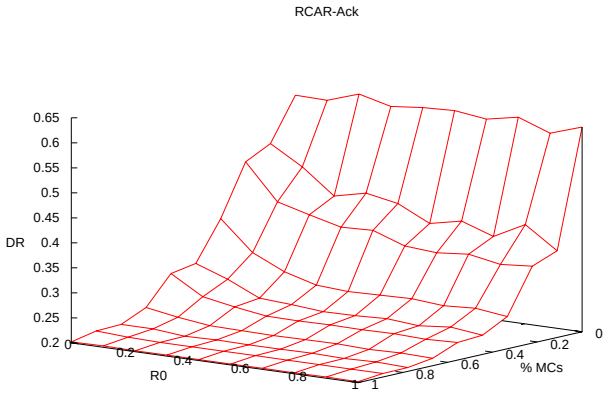


Figure 5.1: Scenario A - Blackhole carriers: Delivery ratio vs. percentage of MCs vs. initial reputation value.

Average Delivery Delay. Figure 5.2 shows the average message delivery delay measured in presence of an increasing percentage of BCs, varying the initial reputation value. With respect to the number of BCs, the average delivery delay degrades with the number of BCs. This will be discussed in the next sections.

With respect to R_0, we note that the average delivery delay decreases when R_0 increases. This means that with a lower value of R_0 the average delivery delay is higher. The value of average delivery delay, indeed, is strictly connected to delivery ratio as discussed in the next sections.

Since delivery ratio suggests to choose a low value of R_0, but average delivery delay a high value, we have performed our simulations considering different implementations of RCAR-Ack. Table 5.5 shows the configurations of RCAR we have chosen to perform our simulations.

Total number of sent messages. Figure 5.3 shows the total number of sent messages measured in presence of an increasing percentage of MCs, varying the initial reputation value.

With respect to the number of MCs, the total number of sent messages degrades with the number of MCs. This will be discussed in the next sections.

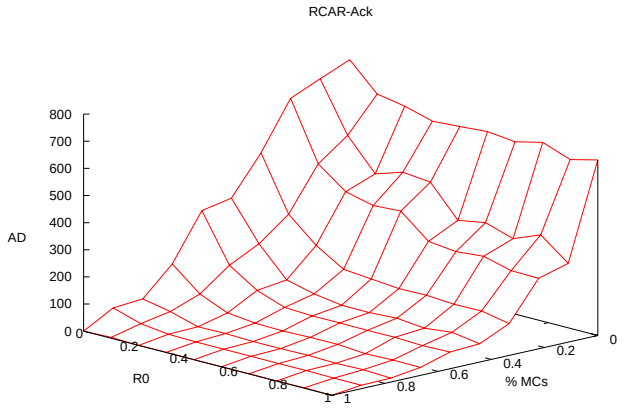


Figure 5.2: Scenario A - Blackhole carriers: Average Delivery Delay vs. percentage of MCs vs. initial reputation value.

With respect to R_0, we note that the total number of sent messages does not depend on the value of R_0. This is due to the fact that the value of R_0 influences only the reputation update mechanism, not the number of generated messages.

Table 5.5 shows the configurations of RCAR we have chosen to perform our simulations. We reserve for future work the analysis of RCAR-Gossip and RCAR-Step-by-step with $R_0 = 1$.

5.3 Choice of the buffer size

We have investigated the influence of the buffer size on the delivery ratio of the protocols. We have noted that the behaviors of CAR and RCAR implementations are similar, with respect to the buffer size. Thus, hereafter we consider RCAR-Ack-2 only.

We have evaluated the delivery ratio of ER and RCAR in Scenario A with different buffer sizes, ranging from 20 to 5000, in presence of an increasing percentage of blackhole carriers. A similar analysis could be done for selfish carriers. However we leave it for future work and we use the values of buffer size found for blackhole carriers also to analyze selfish carriers.

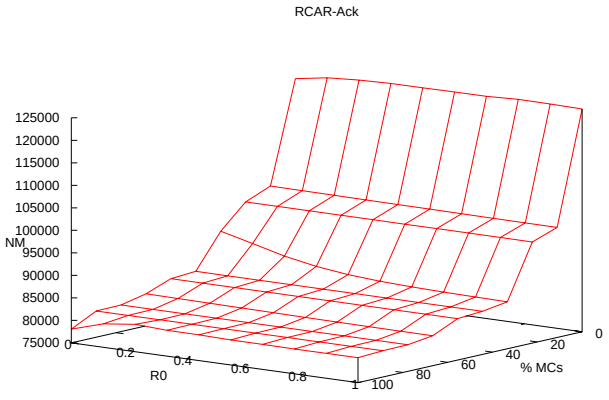


Figure 5.3: Scenario A - Blackhole carriers: Total number of sent messages vs. percentage of MCs vs. initial reputation value.

Delivery ratio. Figure 5.4 and 5.5 show the delivery ratio of ER and RCAR with respect to buffer size, measured in presence of an increasing percentage of misbehaving carriers. With respect to the number of BCs, the delivery ratio degrades when their percentage increases. This will be discussed in the next sections.

The delivery ratio of the protocols increases with the buffer size. In fact, with small buffer sizes carriers may not store all the messages they receive and buffer overflows may occur. However, for buffer sizes greater than 500, the delivery ratio remains constant and tends to be independent on the buffer size.

Referring to ER, we note that its delivery ratio strongly depends on the buffer size. When the buffer size is small (i.e. 20), the delivery ratio of ER reduces of about 54% with respect to the case of a large buffer size (e.g. 500). This means that ER is not indicated when the network is composed of devices having different resource availabilities (etherogeneous networks). However, also with a small buffer size, the delivery ratio of ER remains higher than that of RCAR.

Referring to RCAR, we note that its delivery ratio depends on the number of BCs. When the buffer size is small (i.e. 20), the delivery ratio of RCAR

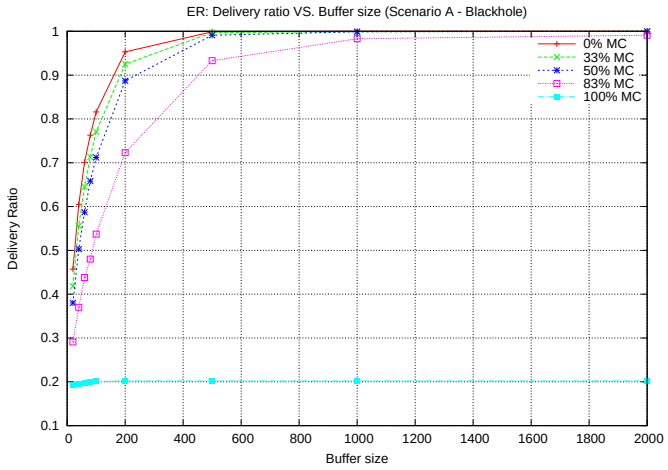


Figure 5.4: Scenario A - Blackhole carriers - ER: Delivery Ratio vs. percentage of misbehaving carriers vs. Buffer Size.

reduces of about 28% with respect to the case of a large buffer size (e.g. 500). This means that RCAR has a lower dependence of the delivery ratio on the buffer size with respect to ER. Thus, RCAR is indicated for etherogeneous networks. However, the delivery ratio of RCAR depends on the presence of misbehaving carriers.

Average Delivery Delay. Figure 5.6 and 5.7 show the average delivery delay of ER and RCAR with respect to buffer size, measured in presence of an increasing percentage of MCs.

With respect to ER, for small values of buffer size (up to 100), the average delivery delay increases since also the delivery ratio increases (this will be explained later). With a buffer size larger than 100 but lower than 500, the average delivery delay decreases. In fact, having a larger buffer, each carrier is able to store more messages. This means that the probability that a faster carrier stores a message in its local buffer is higher. Thus the average delay decreases because fastest carriers forward messages more quickly. With buffer sizes larger than 500, the average delay remains constant, and dependent on the speed of the fastest carriers.

With buffer size larger than 500, the average delivery delay remains al-

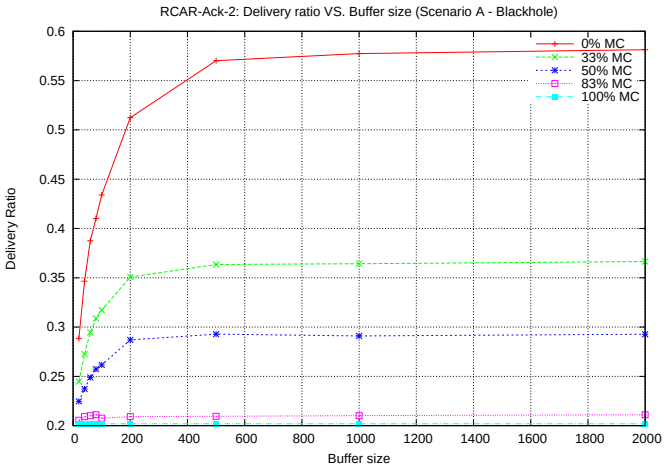


Figure 5.5: Scenario A - Blackhole carriers - RCAR-Ack-2: Delivery Ratio vs. percentage of misbehaving carriers vs. Buffer Size.

most at the same value because the delivery ratio is also costant.

With respect to RCAR, the average delivery delay increases with the buffer size because of the higher number of messages routed asynchronously. The average delay increases until it reaches the maximum value corresponding to the maximum value in the delivery ratio.

It could be interesting analyze the delivery ratio of RCAR with respect to the speed of the carriers. However, we reserve it to future work.

Note that in presence of 100% of BCs the average delivery delay is constant for both ER and RCAR and independent on the buffer size.

Total number of sent messages. Figure 5.8 and 5.9 show the total number of sent messages of ER and RCAR with respect to buffer size, measured in presence of an increasing percentage of BCs.

In both cases, the total number of sent messages increases when the buffer size increases. This is due to the fact that having a larger buffer, more messages can be forwarded. For values of buffer size larger than 500, the total number of sent messages remains constant, because also the delivery ratio remains constant.

As already said for delivery ratio and average delivery delay, with a buffer

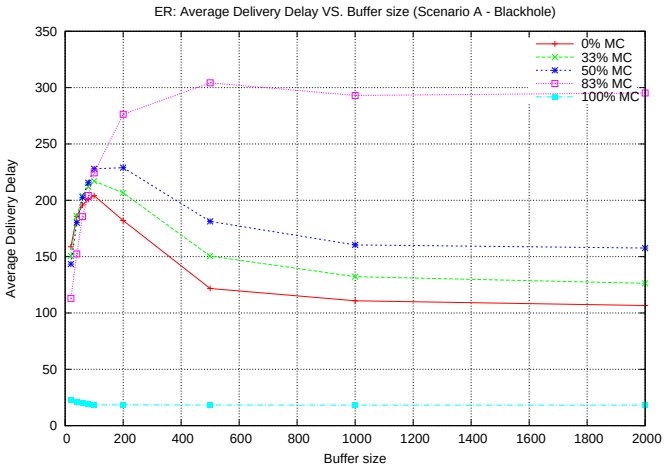


Figure 5.6: Scenario A - Blackhole carriers - ER: Average Delivery Delay vs. percentage of misbehaving carriers vs. Buffer Size.

size larger than 500 also the total number of sent messages tend to remain constant in both ER and RCAR. Thus in the next sections we have considered a buffer size of 1000 because using this value since the behavior of the protocols does not depend on the buffer size.

5.4 Blackhole carriers

In our implementation, BCs ditribute a bogus utility function $U_c = 1$ for each destination for which they are carriers. They also drop all the packets they receive from the other hosts for which they are not receivers.

In ER we have modeled packet dropping only, since there is no distribution of delivery probability.

We have evaluated the performance of the protocols in presence of an increasing percentage of BCs, ranging from 0% to 100%.

Delivery ratio. Figure 5.10 and 5.11 show the delivery ratio of the protocols for Scenario A and B, respectively.

We can see that ER outperforms both CAR and RCAR in both scenarios, presenting a delivery ratio higher than 90% with up to 83% BCs. The price

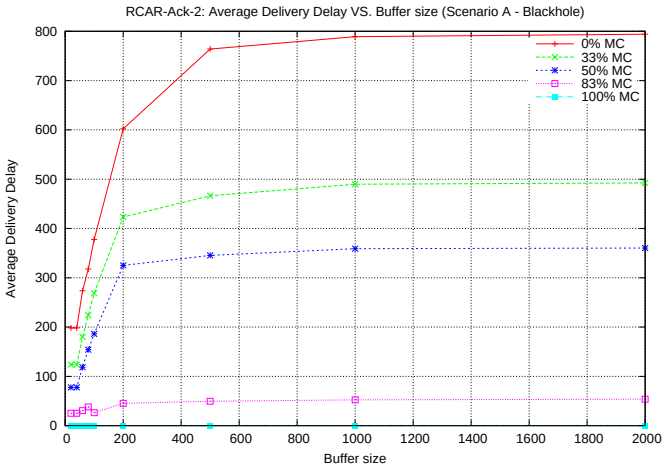


Figure 5.7: Scenario A - Blackhole carriers - RCAR-Ack-2: Average Delivery Delay vs. percentage of misbehaving carriers vs. Buffer Size.

to pay for such high performance is the number of sent messages, typical of dissemination-based approaches. We will discuss such problem later. With more than 83% of BCs, the delivery ratio of ER start to decrease. With 100% of BCs, ER has the same delivery ratio than CAR and RCAR.

Referring to CAR and RCAR, their delivery ratio is always less than one. This is due to the nature of the protocol. See [30] for details.

Referring to CAR and RCAR, with no BCs, delivery ratio in Scenario A are higher than in Scenario B. This is due to the TTL value. Each message passes from a carrier to another before arriving to the receiver. In a dense scenario (B), the number of crossed carriers is higher than a sparse scenario (A). However, due to the high number of crossed carriers, many messages in Scenario B are dropped because their TTL elapses.

When there is no BCs, RCAR-Ack-1 outperforms the other RCAR implementations because at the beginning it trusts all the carriers ($R_0 = 1$). However, in presence of BCs, it is not able to recognize them, due to its low value of R^-. Thus its behavior is similar to CAR.

As soon as BCs appear, CAR delivery ratio severely degrade. We can see that all RCAR implementations outperform CAR and RCAR-Ack-1, because the choice $R_0 = 0$ prevents the selection of BCs in the routing process (they

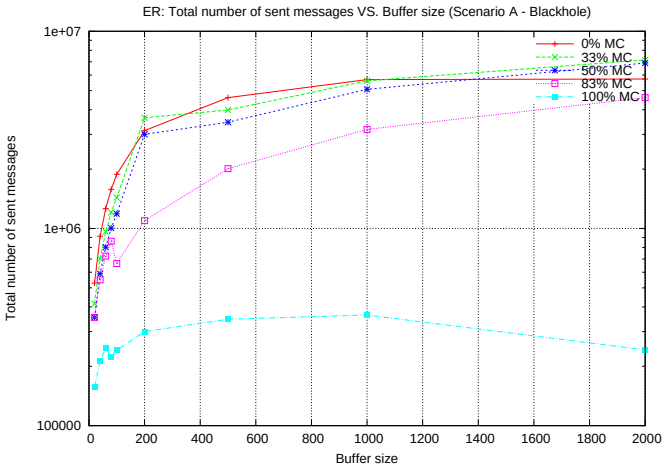


Figure 5.8: Scenario A - Blackhole carriers - ER: Total number of sent messages vs. percentage of misbehaving carriers vs. Buffer Size.

have $LUF \rightarrow 0$). In this case, hosts increase the reputation of the well-behaving carriers only, while BCs are discarded. We note that RCAR-Ack-3 performs better than RCAR-Ack-2. The reason has to be found in the higher value of R^- that RCAR-Ack-2 has with respect to RCAR-Ack-3. A higher value of R^- leads to a more rapid reputation decrease, so that if a well-behaving carrier does not carry messages for a certain period of time time its reputation is quickly decreased until zero among all the hosts.

RCAR-Gossip and RCAR-Step-by-step have the highest delivery ratio. In RCAR-Step-by-step, each host that receives a message (be it a "normal" message or an acknowledgement) increases the reputation of the carriers the message passes through. Hence, a higher number of hosts can increase the reputation of the well-behaving carriers. It is therefore created a more general consensus on the carriers that well-behaved and on those that misbehaved. In RCAR-Gossip the reputation increment is limited to the receiver's neighborhood. Considering the area of a community (250 mt $\times$ 250mt) and the transmission range of a host (200 mt), we can assume that the gossip message reaches almost all the hosts in receiver's community. A host can therefore increase the reputation of the carriers and replace them for one or more destinations in the routing table.

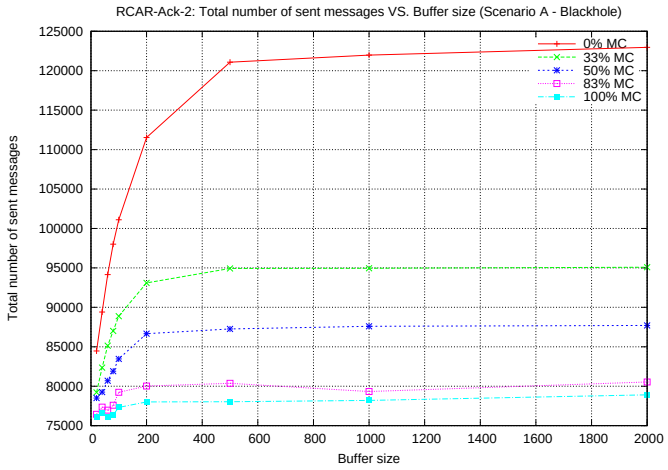


Figure 5.9: Scenario A - Blackhole carriers - RCAR-Ack-2: Total number of sent messages vs. percentage of misbehaving carriers vs. Buffer Size.

In presence of a high percentage of BCs the protocols tend to have the same delivery ratio. This means that RCAR tolerates misbehaviors until a certain percentage of well-behaving carriers is maintained. In presence of 100% BCs, CAR, RCAR and ER present the same delivery ratio because only synchronous messages are sent.

When there are 100% of BCs, performance in Scenario B are higher than in Scenario A. This is because in Scenario B a higher number of messages is sent synchronously than in Scenario A, thus there is no contribution of carriers. These considerations show that CAR, RCAR and its various implementations work better in a sparse scenario.

Average delivery delay. Average delivery delay is calculated on messages routed both synchronously and asynchronously. A low value of average delivery delay indicates that only synchronous messages are sent, whereas a high value of average delivery delay indicates that both asynchronous and synchronous messages are sent.

Figure 5.12 and 5.13 show the average delivery delay for Scenario A and B, respectively.

With no BCs, ER has the lowest average delay in both scenarios. The

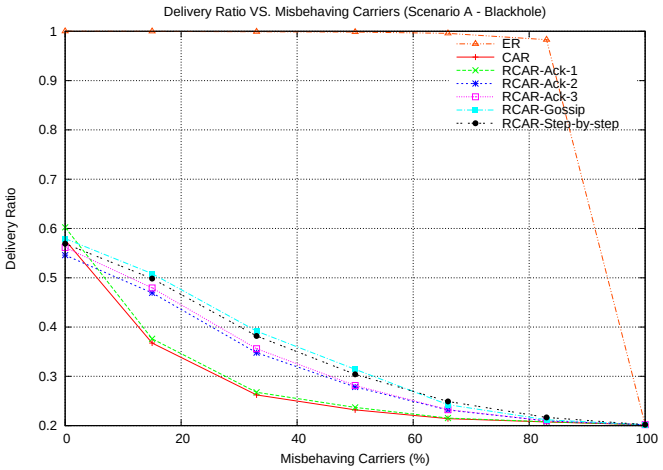


Figure 5.10: Scenario A - Blackhole carriers: Delivery ratio vs. percentage of MC.

reason is the epidemic mechanism, that allows a message to be spreaded over a high number of hosts. However, the delay increases with the number of BCs. This is due to the fact that well-behaving carriers are overloaded of messages and cannot deliver them efficiently. In presence of 83% of BCs there is a peak in the delay, due to the flop of delivery ratio (see Figure 5.10).

CAR has a lower average delay with respect to RCAR in both scenarios. Note that this is not because CAR performs better than RCAR, but because the delivery ratio is lower, so the number of messages routed asynchronously is lower with respect to those sycnhronously routed.

With no BCs, the average delivery delay of the RCAR implementations is higher because the best carrier is chosen not only according to its delivery probability, but also according to its reputation. In fact, the most trusted carrier may not have the highest delivery probability. Messages may therefore not take the fastest route to reach the destination, causing an increment on the delivery delay. It is worth to note that in Scenario A RCAR-Gossip and RCAR-Step-by-step have a lower average delay than RCAR-Ack. This is due to the reputation management technique used, that allows a higher number of hosts to have knowledge of the well-behaving carriers, so that the best carriers can be more efficiently selected during the routing decision process.

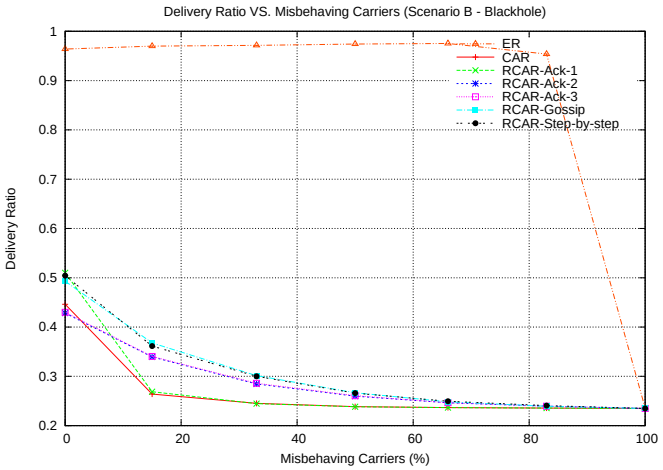


Figure 5.11: Scenario B - Blackhole carriers: Delivery ratio vs. percentage of MC.

In Scenario B instead the average delay of RCAR-Gossip and RCAR-Step-by-step is higher since the number of messages successfully routed is higher. Such messages are routed asynchronously, so their delays are higher and contribute to raise up the average value.

In presence of BCs, the average delay decreases with the number of BCs. RCAR-Gossip and RCAR-Step-by-step have higher values for the same reasons above.

In presence of more than 83% of BCs, the average delay of RCAR is much higher than CAR (almost 200 seconds of difference for RCAR-Gossip and RCAR-Step-by-step), in spite of the low increment in the delivery ratio. This is because messages are sent to the most trusted carriers, that may not be the fastest ones.

Total number of sent messages. Figure 5.14 and 5.15 show the total number of sent messages for Scenario A and B, respectively. ER sends a very high number of sent messages, around 6×10^6 and 1.6×10^7 for Scenario B. Even with 83% of BCs, ER sends 3.17×10^6 messages in Scenario A and 6.37×10^6 messages in Scenario B. This explains why ER outperforms CAR and RCAR in both scenarios in terms of delivery ratio. However, the very high number of sent messages constitutes the greatest limit of ER, because

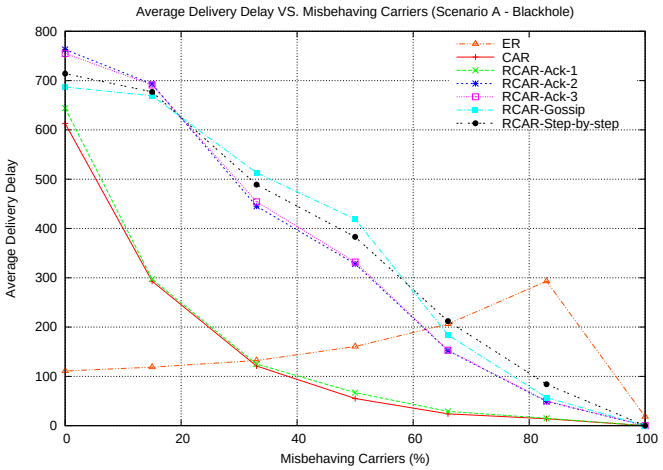


Figure 5.12: Scenario A - Blackhole carriers: Average delivery delay vs. percentage of MC.

it causes the exahustion of network resources and suggests to not use it in DTNs.

Figure 5.16 and 5.17 show instead the total number of sent messages for CAR and RCAR only in Scenario A and B, respectively. With not BCs, CAR sends the least number of messages. Note that the number of generated messages (from sender to receiver) is 5000, while the total number of sent messages in the network is 100000 for Scenario A and about 400000 for Scenario B. This is due to message retransmissions and also to messages forwarded from carrier to carrier until the TTL elapses.

In presence of BCs, the number of messages sent by CAR degreades. This follows what previously discussed for delivery ratio.

RCAR-Gossip generates less messages than RCAR-Step-by-step and RCAR-Ack. This is due to the fact that, once a neighbor receives a gossip message, it does not further forward such message. As disccused in the case of delivery ratio, the network scenario we choose allows a gossip message to reach a relevant number of hosts. On the other hand, RCAR-Step-by-step, RCAR-Ack-2 and RCAR-Ack-3 generate the same number of messages, because they all rely on the acknowledgements mechanism. RCAR-Ack-1, instead, generates a lower number of messages, although it relies on the acknowledgements

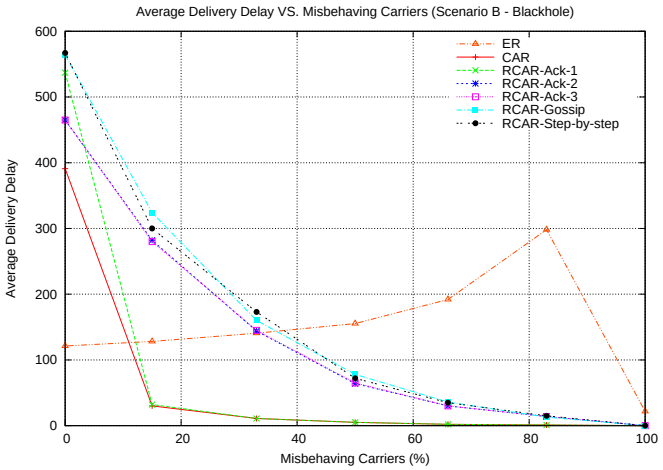


Figure 5.13: Scenario B - Blackhole carriers: Average delivery delay vs. percentage of MC.

mechanism. This is due to the fact that it is not able to update reputations quickly, thus the reputation mechanism is not used and it behaves like CAR.

5.5 Selfish carriers

In our implementation SCs do not distribute routing updates, neither synchronous nor asynchronous. The aim is to hide theirself from the other hosts of the network. However, SCs forward the messages they receive.

With respect to ER, since there is no distribution of routing information, the behavior of a SC should be similar to those considered in Section 5.4 while discussing BCs, as well as the considerations we could make. In other words, in ER a carrier that drops the messages it receives has a selfish behavior. Hence, we will not consider ER, only CAR and RCAR.

The amount of network resources a carrier is allowed to use is limited by the reputation value assigned to that carrier. Moreover, network resources are limited for carriers only, i.e. for messages created by carriers.

Since we are not limiting the number of messages of standard hosts, we implicitly consider such hosts as *fully trustworthy*.

We have evaluated the performance of the protocols in presence of an

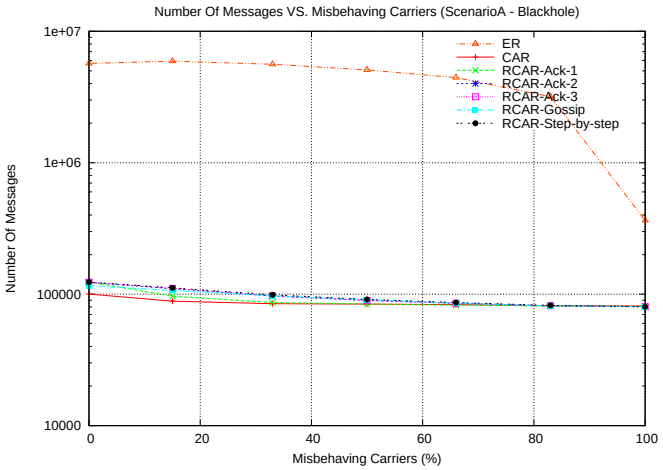


Figure 5.14: Scenario A - Blackhole carriers: Total number of sent messages vs. percentage of MC. ER included.

increasing percentage of SCs, ranging from 0% to 100%. The value of M_{cj} strongly depends on the value of M_c (Message Quota). Thus in our simulations we have supposed it constant. We have performed a series of preliminar simulations to establish the best value of M_c for each RCAR implementation and then we have employed such value for our tests.

Message Quota. We have evaluated the following values of Message Quota: 5, 20, 50, 80, 100, 200, 300, 500.

In order to establish the best value of M_c, we have considered both the delivery ratio and the average delivery delay. We defined the ideal delivery ratio of RCAR in presence of X% of SC, expressed in Formula 5.1.

$$DR(X)_{ideal} = DR(0)_{CAR} - \frac{X}{100} \times DR(0)_{CAR} \times [1 - DR(100)_{CAR}] \quad (5.1)$$

Where DR(X) is the delivery ratio in presence of X% of SCs. In other words, the ideal delivery ratio is given when the fraction of messages the X% of SCs is not forwarded. The best value of Message Quota MQ, considering only the delivery ratio is that for which the difference between DR_{ideal} and DR_{MQ} is minimum. However the obtained value must be combined with the average delivery delay.

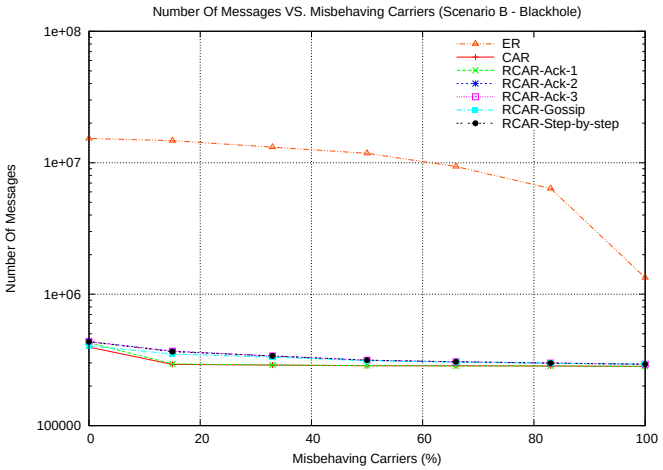


Figure 5.15: Scenario A - Blackhole carriers: Total number of sent messages vs. percentage of MC. ER included.

Protocol	Optimal value	Suboptimal value
RCAR-Ack-1	80	100
RCAR-Ack-2	≥ 200	-
RCAR-Ack-3	500	-
RCAR-Gossip	500	200
RCAR-Step-by-step	≥ 300	-

Table 5.6: Optimal and suboptimal values of Message Quota in Scenario A.

The value of Message Quota which gives the best average delivery delay must be calculated as follows. Considering the only synchronous routing (100% of SCs) the delay must be less or equal to that of CAR in the same conditions. Furthermore, considering the best conditions (0% of SCs), the delay must be also less or equal to that of CAR in the same conditions.

Combining both results of delivery ratio and average delivery delay, we have obtained the optimal values of Message Quota. Table 5.6 and 5.7 show the optimal and suboptimal values of Message Quota for each protocol in the case of Scenario A and Scenario B respectively. Hereafter, we discuss the measured metrics using the optimal values of Message Quota.

Delivery ratio. Figure 5.18 and 5.19 show the delivery ratio of the protocols for Scenario A and B, respectively.

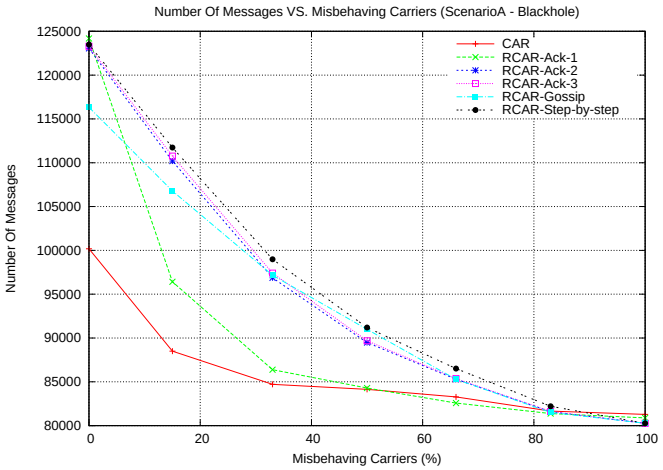


Figure 5.16: Scenario A - Blackhole carriers: Total number of sent messages vs. percentage of MC. CAR and RCAR only.

Protocol	Optimal value	Suboptimal value
RCAR-Ack-1	5	20
RCAR-Ack-2	500	300
RCAR-Ack-3	500	300
RCAR-Gossip	100	-
RCAR-Step-by-step	500	≥ 200

Table 5.7: Optimal and suboptimal values of Message Quota in Scenario B.

CAR-ideal is the ideal behavior that CAR should have to limit the amount of network resources to selfish carriers only. A protocol having a delivery ratio higher than CAR-ideal does not contrast SCs, because it allows them to send their messages without resource limitations. On the other hand, a protocol with lower delivery ratio than CAR-ideal penalizes both well-behaving and SCs, because it overly limits the amount of network resources.

As the number of SCs increases, we note a general decrease of the delivery ratio, because of an increasing number of carriers hiding themselves from the routing process makes it difficult to find routes for messages. Well-behaving carriers get overloaded, and messages are dropped because of buffer overflows.

CAR delivery ratio is always higher than those of CAR-ideal, because it has no mechanism to contrast MCs. The RCAR protocols have a delivery ratio lower than CAR-Ideal. However, RCAR-Ack-1 has the most interesting

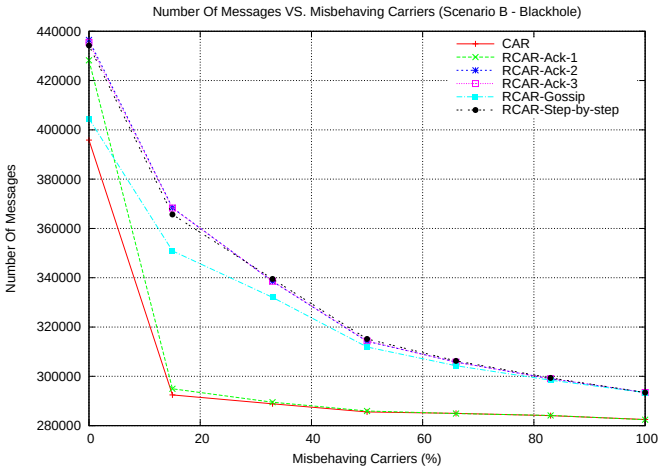


Figure 5.17: Scenario B - Blackhole carriers: Total number of sent messages vs. percentage of MC. CAR and RCAR only.

behavior. With a low number of MCs, its delivery ratio is lower than CAR-ideal. As the number of SCs increases (more than 60%), its delivery ratio becomes higher than CAR-Ideal. This is due to the initial reputation value assigned to each host, as discussed before.

With no MCs, RCAR-Gossip and RCAR-Step-by-step have higher delivery ratio than RCAR-Ack-1, because of the different reputation management technique adopted. Both RCAR-Gossip and RCAR-Step-by-step allow a higher number of hosts in the network to recognize a well-behaving carrier, thus to create a more general consensus on its reputation, finding the best carriers.

When the number of MCs increases, all the implementations of RCAR, except for RCAR-Ack-1, tend to CAR-ideal. This means that they work better with a high number of MCs.

With reference to Scenario B, we first note that the delivery ratio of all the protocols is lower than in Scenario A. This is due to the limitation imposed by the TTL value, as discussed in Section 5.4.

CAR has a delivery ratio higher than RCAR because there is no limitation on the network resources. With a low number of SCs (less than 50%), all the RCAR protocols have delivery ratio lower than CAR-ideal. This is because

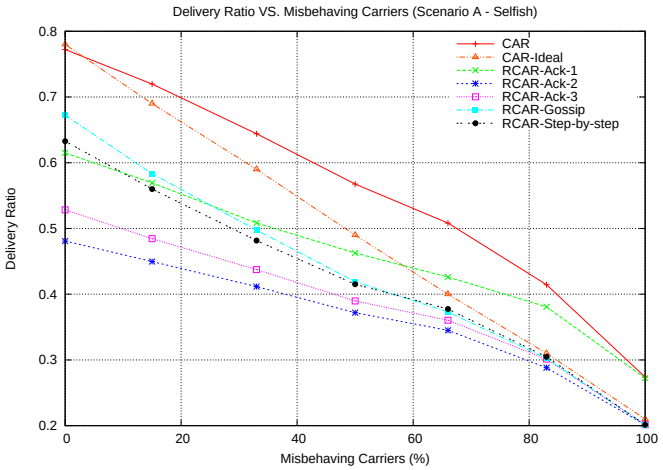


Figure 5.18: Scenario A - Selfish carriers: Delivery ratio vs. percentage of MC.

of the reduced reputation update opportunities, that penalize both well-behaving and misbehaving carriers. In fact, in a dense scenario (B) the probability to commit a message twice to the same carrier is lower than in a sparse scenario (A) and consequently there are less opportunities to increase the reputation of a carrier. RCAR-Ack-1 has a higher initial reputation value R_0 and it has therefore the highest delivery ratio.

In presence of more than 50% and up to 83% of SCs , the delivery ratio of all the protocols (CAR and RCAR) remains almost at the same value. This is because of two reasons. Firstly, in a dense scenario the routing mechanism is tolerant to misbehaviors since there are enough carriers to take alternative routes to reach a given destination. However, this implies a delay increase, as discussed after. Secondly, the delivery ratio depends also on the nature of selfish carriers. As already said, a selfish carrier is a carrier which does not send routing updates, but it receives them from the well-behaving hosts (or carriers). Therefore, a selfish carrier is able to send its own messages to well-behaving hosts (or carriers). In CAR, where there is no limitation of resources, a selfish carrier is able to exploit network resources (i.e. both synchronous and asynchronous routing). In RCAR, instead, a selfish carrier is able to send only messages that have itself as sender and a host (or carrier)

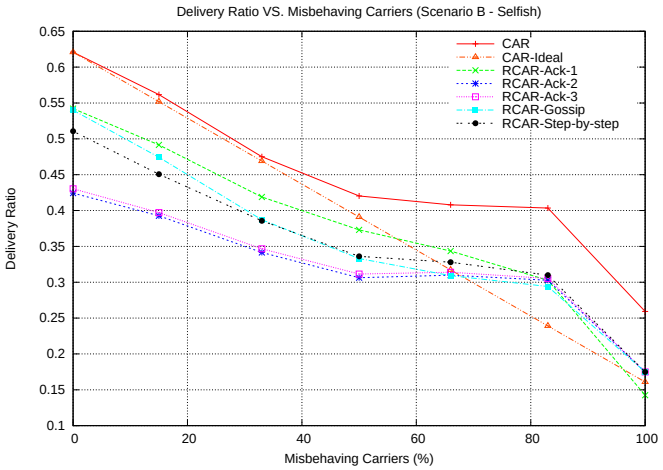


Figure 5.19: Scenario B - Selfish carriers: Delivery ratio vs. percentage of MC.

directly in reach as receiver. In other words, in RCAR a selfish carrier cannot exploit synchronous and asynchronous routing. These considerations explain why the delivery ratio is constant (from 50% to 83% of SCs): the fraction of messages generated by SCs and sent directly to the other hosts always gives its contribute to delivery ratio, independently on the number of SCs. However this happens only for high percentages of MCs, because for low values its contribute is low.

With 100% of MCs all the RCAR protocols tend to CAR-Ideal.

Average delivery delay. Figure 5.20 and 5.21 show the average delivery delay of the protocols for Scenario A and B, respectively.

Note that the delay in both scenarios has the same order of magnitude. This does not happen for blackhole carriers, where the order of magnitude is different in scenario A and B.

With no selfish carriers, RCAR-Ack-2 and RCAR-Ack-3 have the lowest average delivery delay because of the lower values of delivery ratio. With less than 83% of SCs, the average delivery delay of all the protocols increases because hosts have to select alternative routes to forward their messages. In presence of a high number of selfish carriers, the average delivery delay of the protocols is higher than those resulting from the analysis of blackhole

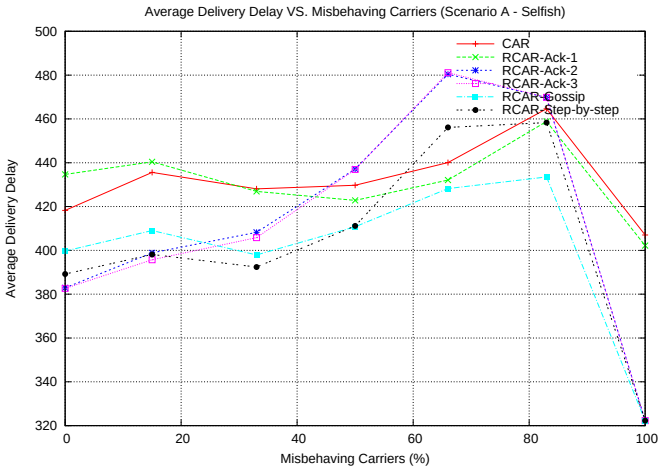


Figure 5.20: Scenario A - Selfish carriers: Average delivery delay vs. percentage of MC.

carriers. This is due to the fraction of messages with a selfish carrier as sender that are directly carried by it until the final recipient. With 100% of SCs delay is drastically reduced, because of the low value of delivery ratio.

In Scenario B, the average delivery delay of RCAR around 83% of SCs has a peak. This corresponds to a delivery ratio quite constant (see Figure 5.21). This is due to the fact that each SC must send its own messages directly to the receiver. Thus, before meeting it, it must store the message in its local buffer. This contributes to increase the average delay. In CAR, instead, there is not the peak because selfish carriers are able to exploit both synchronous and asynchronus routing to forward their own messages thus messages are not stored in the buffer.

RCAR-Gossip represents the best trade-off among the RCAR implementations in terms of both delivery ratio and average delay.

Total number of sent messages. Figure 5.22 and 5.23 show the total number of sent messages in Scenario A and B, respectively.

With reference to Scenario A, with no SCs, RCAR-Ack-1, RCAR-Gossip and RCAR-Step-by-step send a number of messages higher than CAR because of the reputation management messages (acknowledgements and gos-

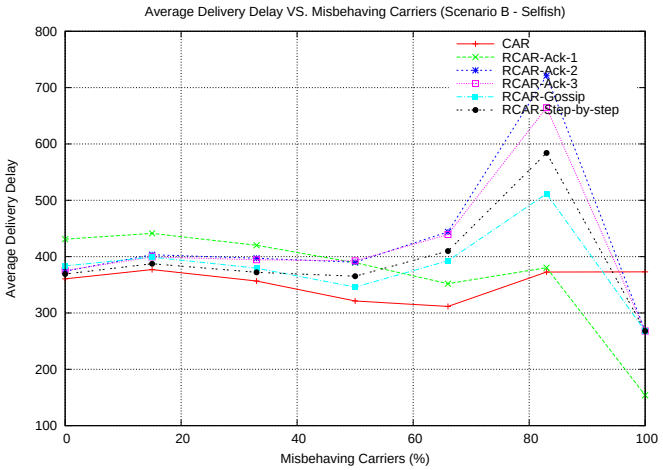


Figure 5.21: Scenario B - Selfish carriers: Average delivery delay vs. percentage of MC.

sip). RCAR-Ack-2 and RCAR-Ack-3 send instead a lower number of messages since their delivery ratio is also lower.

With more than 15% of SCs, RCAR-Gossip sends more messages than the other RCAR implementations since the gossip message is forwarded through the synchronous routing, while acknowledgements are immediately discarded because of the resource limitations on the well-behaving carriers.

With reference to Scenario B, all RCAR implementations send a lower number of messages than CAR because they overly limit network resources. The reason of such limitation has to be found in the lower number of reputation increment opportunities per host in this scenario, as previously discussed.

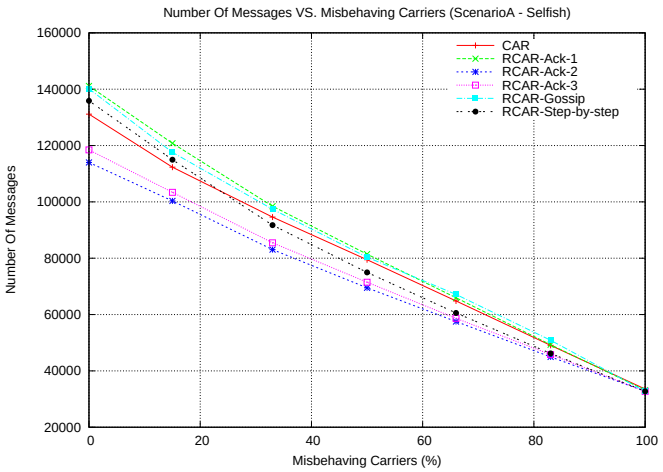


Figure 5.22: Scenario A - Selfish carriers: Total number of sent messages vs. percentage of MC.

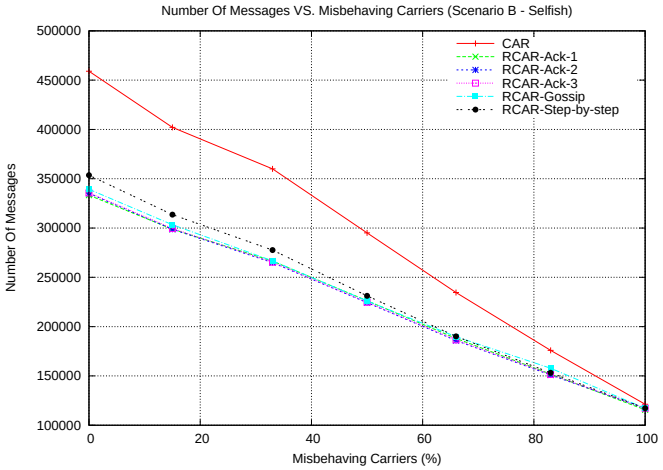


Figure 5.23: Scenario B - Selfish carriers: Total number of sent messages vs. percentage of MC.

Chapter 6

Implementation

RCAR is a reputation-based extension of CAR to mitigate host misbehaviors. In order to test the protocol, we started from an existing implementation of CAR, provided by the authors of [30]. CAR has been implemented under the OMNeT++ simulation environment (see Appendix A). Section 6.1 introduces the CAR implementation. Section 6.2 describes the extensions we provided to CAR in order to embed the reputation technique.

6.1 CAR

This section introduces the main data structures and messages of such implementation.

6.1.1 Modules

CAR simulation model in based on four modules: Host, simController, Engine and DataCollector.

Host. Main module, implementing the protocol functionalities. The main data structures defined into this module are:

- *info and context routing table* (infoAndRoutingTable): routing table maintaned on each host. It stores information about both synchronous

targetHostId	nextHopHostId	distance	carrierHostId	deliveryProb

Table 6.1: infoAndRoutingTable format.

targetHostId	nextHopHostId	distance

Table 6.2: infoAndRoutingTable: synchronous part.

and asynchronous routing. Table 6.1 represents an entry of infoAndRoutingTable. The table can be thought as composed by two logically separated tables: a table for synchronous routing, represented in Table 6.2 and a table for asynchronous routing, represented in Table 6.3.

With respect to the synchronous routing table, *targetHostId* is the receiver, *nextHopHostId* and *distance* respectively the next hop host in the path to reach the receiver and the distance (in term of number of hops) to reach it. The table is maintained following the specifications of the DSDV routing protocol [32] (see also Section 3.2). The infinte distance was set to 16, as is used in RIP [18].

With respect to the asynchronous routing table, *carrierHostId* is the identifier of the carrier with the highest delivery probability, the value of which is stored in *deliveryProb*. A Kalman filter predictor is associated to each entry of the table. The filter is updated every time a new *deliveryProb* value is received from *carrierHostId*. The filter is also periodically updated between routing table exchanges: the value of such interval is a configurable parameter. The output of the filter is used as input (i.e. is short-circuited) when one or more updates are not received, e.g. because of interference in the wireless channel or because of host disconnection. When receiving a routing table update, delivery probability is always updated to guarantee that forwarding decisions are supported by fresh information. An entry is replaced with a different carrier in case a routing update distributes a higher

targetHostId	carrierHostId	deliveryProb

Table 6.3: infoAndRoutingTable: asynchronous part.

delivery probability for a given destination. After a certain number of missing updates, the predicted value is not considered accurate enough, the relative entry is considered stale and removed from the table (see [30]).

Update intervals of both infoAndRoutingTable and Kalman filters have to be chosen carefully. Typically they are somehow proportional to the rate at which changes happen: changes in the neighborhood in case of routing table, changes in the observed value in case of Kalman filter. They are currently set at the same value.

- *local predictor*: structure encapsulating Kalman filters, one per each context attribute evaluated. CRC and colocation with other hosts (a predictor per each host) are the context attributed currently evaluated. Local predictors are used for the prediction of the values assumed by each context attribute. They are used during the periodical update of the Kalman filters associated to each entry of the infoAndRoutingTable for the computation of the predicted values of delivery probabilities for each known destination.

- *local message buffer*: it stores messages in case the receiver is currently unreachable, i.e. there is no information about to reach that receiver in infoAndRoutingTable. Delivery of the messages stored in the local buffer is performed with periodical attempts. The interval between attempts is a configuration parameter. It has to be chosen carefully, trying to balance responsivity with resource consumption. In case of full buffer, messages are replaced following a circular replacement strategy.

simController. It is responsible of two functions: select the misbehaving carriers at random, and create the messages. The policy to follow in selecting sender and receiver while creating messages depends on the misbehavior currently analyzed: in case of analysis of BCs we selected sender and receiver at random among the "non-carriers", while analyzing SCs we selected sender

```
<parameter>
   <area_shape>rectangle</area_shape>
   <xsize>1000.000000</xsize>
   <ysize>1000.000000</ysize>
   <extension_factor>1</extension_factor>
   <numberOfNodes>50</numberOfNodes>
   <range>200.000000</range>
   <mobility_model>socialFoundedMM</mobility_model>
</parameter>
```

Listing 6.1: Mobility traces file: parameter section.

and receiver at random among all the hosts in the network. The total number of messages to create is a INI paramater.

Engine. Module responsible for hosts' mobility. The authors selected the Community-based mobility model (Section 2.2) to evaluate the performance of the CAR protocol. The module periodically parses an XML file containing the mobility traces for each host in the network. Such XML file containing the mobility traces has been generated using an external tool, available for free at [3]. It generates traces for the ns-2 simulator. However, the traces can be used for OMNeT++ too, with the only difference that some parameters will not be used. The trace file consists basically of three sections:

- *parameter* section, defining general parameters for the simulation area. Size and transmission range of each host are the parameters of our interest. An example is reported in Listing 6.1.

- *node settings* section, defining for each host the initial position, expressed in (x, y) coordinates. An example is reported in Listing 6.2. Note that carriers will then move around the area, while the hosts with null speed will stay in their initial position during all the simulation.

- *mobility* section, defining for each carrier the (x, y) coordinates to which to move, and the related speed. An example is reported in Listing 6.3.

```
<node>
   <node_id>1</node_id>
   <position>
      <xpos>298.857049</xpos>
      <ypos>484.139795</ypos>
   </position>
</node>
```

Listing 6.2: Mobility traces file: node settings section.

```
<position_change>
   <node_id>8</node_id>
   <start_time>3070.000000</start_time>
   <destination>
      <xpos>498.315967</xpos>
      <ypos>427.158313</ypos>
   </destination>
   <velocity>13.982477</velocity>
</position_change>
<position_change>
```

Listing 6.3: Mobility traces file: mobility section.

DataCollector. Its purpose it to collect statistical information for post-processing analysis. Every time a host receives a message, it delivers a notification to this module. The notification contains the message identifier and the time at which the message was created by the module simController. At the end of the simulation, it will elaborate the information contained in the notifications to evaluate the performance metrics. The metrics currently evaluated are the average message delay, delay distribution, total number of sent messages around the network and delivery ratio (see Chapter 5). Such metrics are written in separated log files.

6.1.2 Messages

CAR implementation defines different messages, implementing "data" messages, protocol updates and timers. Data messages have no payload, they are only formed by a header. A message implementing a timer is called

self-message. Here follows a list of the message types defined in current implementation of CAR and how they are handled:

- **CONTROL_MSG**: created by the module simController, this message is directly delivered to the Host module representing the sender and contains the identificator of the receiver. The sender, upon receiving this message, creates a INFO_MSG (representing a data message) and delivers it toward the indicated receiver.

- **INFO_MSG**: represents data messages. Such messages are forwarded toward the indicated receiver following the algorithm described in Section 3.2. The host tries first to send the message synchronously; if not possible, it tries to send the message asynchronously. If a carrier is found, the message is forwarded to the carrier using the underlying synchronous protocol. If the message instead can not be delivered nor sycnhronously neither asynchronously, it is stored in the local buffer. To each INFO_MSG is assigned a TTL field, defining a maximum number of retransmissions.

- **AUTO_MSG**: self-message, used to update the synchronous part of the infoAndRoutingTable. The purposed is to periodically control the neighborhood, detecting the hosts that are not in transmission range anymore, assigning them an infinite distance in infoAndRoutingTable. The operation is currently performed every second.

- **INFO_AND_ROUTING_TABLE_UPDATE_MSG**: self-message, part of the DSDV routing protocol update. The host creates a PROTOCOL_MSG, containing the content of its own infoAndRoutingTable, both the synchronous and asynchronous part. The message is then broadcasted to the neighbors.

- **PROTOCOL_MSG**: represents a routing update. The host analyzes the message in search of "better" routes and to update the delivery probabilities of the carriers they both know. With respect to the synchronous part, a next hop is replaced if a new path with a lower number

of hops is announced, or if the same next hop can reach the target with a lower number of hops. With respect to the asynchronous part, a carrier's delivery probability is always updated, the value is also used as input of the Kalman filter. It may happen that for some destinations the routing update advertises a carrier with a higher delivery probability: in such cases the current carrier is replaced and the Kalman filter reset. If a new destination is announced, it is insered in the infoAndRoutingTable. If the table is full, a circular replacement strategy is performed.

- **LOCAL_BUFFER_FLUSH_MSG**: self-message, representing the periodical attempt to transmit the messages stored in the local buffer. A INFO_MSG is created, and the forwarding policy is the same. If a message can not be delivered, it stays in the local buffer until the next attempt. Note that a message can also be deleted from the buffer because of the circular replacement strategy, i.e. a new and more recent message can replace an existing one.

- **CRC_UPDATE_MSG**: self-message, emulating the process of CRC prediction. The current CRC value is 1 if no hosts are in reach (it probably means that the host is a carrier moving between partitions), otherwise the value is calculated according to Formula 3.1. The current CRC value is then passed to the local predictor.

- **KALMAN_FILTER_AND_COLOCATION_UPDATE_MSG**: self-message, representing the periodical update of the Kalman filters of the infoAndRoutingTable. The output of the Kalman filter is used to replace the delivery probabilities of the carriers not updated during the last interval.

  Colocation with other hosts is also updated. Its current value is 1 if the target host is within the transmission range, 0 otherwise, passed as input of the local predictor. Formula 6.1 describes the formula used to

```
message mobMessage
{
    int type;
    int protocolMessageType;
    int messageId;
    int recipient;
    int intermediateRecipient;
    int sender;
    int receiver;
    int numHops;
    int totalNumHops;
    double timestamp;
    int hostIdList[500];
    int distanceList[500];
    int carrierHostId[500];
    double deliveryProb[500];
}
```

Listing 6.4: Declaration of mobMessage.

calculate the new delivery probabilities for each know host:

$$deliveryProb = colocW * colocationPred + CRCW * CRCPred \quad (6.1)$$

colocW is the weigth assigned to the colocation value (currently 0.75), *colocationPred* the predicted value; *CRCW* is the weight assigned to CRC (currently 0.25) and *CRCPred* the predicted value. If the current delivery probability, locally evaluated for each known destination, is higher than the probability of some infoAndRoutingTable entries, then the host sets itself as carrier.

Listing 6.4 show the general format of the OMNeT++ message used to implement the different types listed above. Note that *recipient* is used for INFO_MSG and CONTROL_MSG, whereas *receiver* is used for routing update messages.

6.2 RCAR

RCAR extends CAR introducing data structures, messages and integrating the existing operations.

The data structures introduced by RCAR are the crossed hosts list, the reputation table and the reachability table.

The *crossed hosts list* corresponds to the *clist*, defined in Section 4.3. It contains a list of the carriers the message passes through. *clist* is added to the message mobMessage, defined in Listing 6.4. In our implementation we did not implement any security mechanism to protect *clist*. We left this problem for future work.

The *reputation table* (repTable) stores the reputation values for each carrier. Reputation evaluation is performed according to one of the techniques described in Section 4.3. Current implementation considers linear updates only. repTable also stores the value M_{cj}, the number of messages a host j is allowed to forward on behalf of a given carrier C. Every time a host forwards a message on behalf of a carrier, it decrements M_{cj} by one. In case such value reaches zero, then the host/carrier will not forward any other messages on behalf of that carrier until the next periodical decrement. If the reputation of a given carrier reaches zero, the host will not forward messages for it anymore. M_{cj} is updated during the periodical reputation decrease, according to formula 4.2.

The *reachability table* (reachTable) is a multidimensional structure of size $C \times N$, where C is the total number of carriers and N the total number of hosts in the network. We currently assume such values known in advance. The table stores the delivery probability that a carrier has to reach a given destination. Such values are obtained by storing the last routing updates received. For example, the entry $reachTable[c][j]$ stores the delivery probability of carrier C to reach host j. In this sense, reachTable represents a "second-level" routing table, i.e. a cache of routing information that otherwise would have been not considered. reachTable is used during the route replacement process in case of increase of a carrier's reputation (see Section 4.3). The table is updated every time a new routing update is received.

The table is also periodically reset, in order to avoid using obsolete routing information. Currently the interval is the same of the routing update.

RCAR introduces the following messages:

- **REPUTATION_TABLE_UPDATE**: represents a timer, associated to the periodical decrease of reputation values in repTable. The value of M_{cj} is also updated.
- **ACK_MSG**: can be either an acknowledgement message or a gossip message, depending on the reputation update technique considered (see Section 4.3). The gossip message is no further forwarded. The acknowledgement message is instead forwarded following the same rules of a INFO_MSG (see Section 3.2).

Chapter 7

Conclusions

Delay Tolerant Networking (DTN) is an emergent network paradigm, designed to allow communication in disconnected networks. Such networks are characterised by frequent disconnections, long and variable delays, high error rates. They are typical of extreme environments such as disaster relief, military battlefield, developing regions where no fixed infrastructure is present. Communication in such networks relies on the mobility of the so-called *carriers*, hosts physically carrying messages among network partitions.

Context-aware Adaptive Routing (CAR) is a routing protocol designed for DTNs, with the aim to select the best carrier, i.e. the carrier having the highest probability of successful message delivery. Computation of such delivery probability is based on a weighted sum of a set of context attributes. They can be the change rate of connectivity, colocation with other hosts, residual battery power, etc. CAR uses forecasting techniques based on Kalman filter theory to predict the future values assumed by the context attributes, for a better prediction of future host's movements.

CAR relies on the assumptions that carriers are collaborative, i.e. that are partecipating in message forwarding, carrying messages on behalf of other hosts. In real-life systems a carrier, because of selfishness or malicious purposes, may not wish to collaborate in message forwarding. This work presented RCAR, a reputation-based approach to mitigate host misbehaviors in DTNs. RCAR extends CAR, exploiting the concept of reputation to de-

tect and exclude malicious carriers from the network. Low reputation values indicate misbehaving carriers, whereas high reputation values indicate well-behaving carriers. The reputation system is completely decentralized, each host keeping a local table with the reputations it assigned to each known host of the network. Reputation increments basically happen whenever a host receives a message, while reputation decrease happen according to periodical updates. Misbehaving carriers are also excluded from the network, by imposing each host of the network to forward a number of messages proportional to host's reputation. We proposed and evaluated three different reputation management techniques.

CAR and RCAR, together with Epidemic Routing (ER), have been compared via simulation, using the OMNeT++ simulation environment. Host movements have been defined using a human mobility model, called Community-based mobility model. This model defines host mobility based on the strength of social relationships among them. Results of simulations show that ER outperforms both CAR and RCAR at the cost of a very high number of messages sent around the network, with consequent waste of network resources. In presence of misbehaving carriers, RCAR shows up 14% increment in delivery ratio with respect to CAR. It has also been shown that both CAR and RCAR work better in sparse scenarios.

This work represents an initial study on the use of reputation techniques on DTNs, and future work have to be done. First, we propose to perform simulations implementing both physical and MAC layer, in order to test the influence of interferences, memory and battery consumption on the routing process. We then propose to investigate different reputation update functions and the resilience of the reputation system to different types of attacks.

Appendix A

OmNet++ simulation environment

Omnet++ [41] is a C++-based discrete event simulator for modelling communication networks, multiprocessor and other distributed or parallel systems. It is developed since 1997 by Andras Varga and is available for free for non-profit use. Omnet++ is available under the most common platforms: Windows, Linux and MAC OS/X. A C++ compiler (the GCC toolchain or Microsoft Visual C++ compiler) is needed. Its main design choices are:

- an object-oriented structure, facilitating the creation of frameworks like the Mobility Model [4] and the INET framework [1], and model reusage.
- provision of graphical tools for editing, simulation animation, debugging, results visualization and analysis.

An Eclipse-based IDE (Integrated Desktop Environment) is also provided, allowing the user to edit files, run simulations and analyze results without leaving the GUI.

Following sections provide an introduction of the simulation environment. For detailed documentation, please refer to the online user manual [2].

A.1 Model structure

An Omnet++ model is a collection of modules communicating each other via message passing. Events are also represented as messages. For example, a

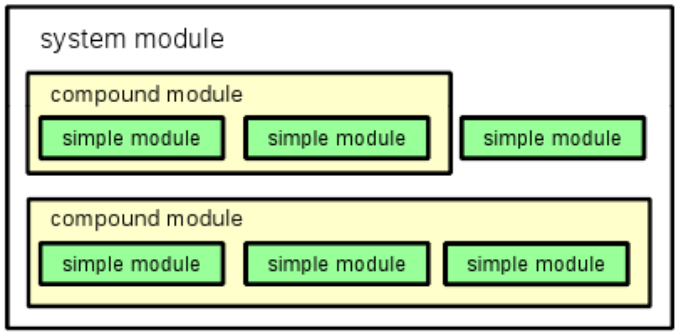


Figure A.1: Omnet++: single and compound modules

timer is implemented as a message that a module sends to itself with a certain delay (timeout expiring). Omnet++ is object-oriented, so all modules are instances of certain classes, representing *module types*. All classes have to be derived from the abstract class *cSimpleModule*, providing the definition for the basic module functionalities: initialization, message handling and module finalization. Active modules are called *simple modules*, written in C++ using the simulation library. Simple modules can be aggregate together to form *compound modules* (see Figure A.1). There is no behavioral distinction from the "outside world" between simple and compound modules. Compound modules enable infinite nesting, making easy a hierachical structuring of simulation programs.

A.2 Messages

Messages may contain arbitrary data, defined in .msg files. The syntax used to defined such files is very close to C++. While compiling a simulation program, a precompiler parses the .msg files: using reflection code it generates a _m.h and a _m.cc files. Listing A.1 shows an example of custom message declaration, while Listing A.2 shows how to use such declared message in simulation code. Note that getter and setter methods are automatically generated.

```
// file myMessage.msg

message myMessage
{
    int payload;
}
```

Listing A.1: Example of .msg file

```
#include "myMessage_m.h"

myMessage msg = new myMessage("messageName");
msg->setPayload(55);
// ...
int pl = msg->getPayload();
```

Listing A.2: Example of message usage

A.3 Topology description: the NED language

Modules typically send messages via *gates*, representing the communication interfaces. Input and output gates can be linked with a connection, created within a single module hierarchy. Connections are allowed only between modules at the same level. The user can assign specific parameters to connections (random distributions are also provided), creating the so-called *channels*.

Modules can have parameters as well, mainly used to pass data from the configuration file to simple modules. Gates and the entire network topology (i.e. how modules are grouped and connected each other) can be defined using NED (NEtwork Description), the Omnet++'s topology description language. NED allows the user to declare simple and compound modules and their interconnections.

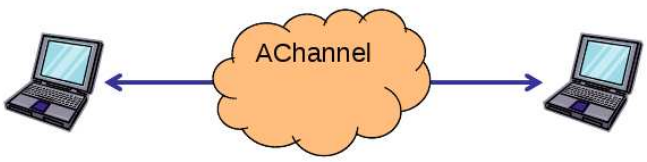


Figure A.2: Omnet++: example network for NED representation

```
// file myNetwork.ned

// host declaration
simple host
    gates:
        in: inPort;
        out: outPort;
endsimple

// channel definition
channel AChannel
    delay normal(0.0015, 0.005)
    error 0.0001
    datarate 2000
endsimple

// network definition
// module connection are defined here
module Network
    submodules:
        hostA: host;
        hostB: host;
    connections:
        hostA.outPort− −>AChannel− −>hostB.inPort;
        hostA.inPort<− −AChannel<− −hostB.outPort;
endmodule
```

Listing A.3: .ned file describing the example network in the picture

NED syntax is C++-like and straightforward. Listing A.3 shows the constructs for the declaration of the simple network represented in Figure A.2.

A.4 Module implementation

As said, modules are instances of certain classes derived from the abstract class *cSimpleModule*, declared in the simulation header library *omnetpp.h*. A typical Omnet++ model implementation is a collection of .h and .cc files, with class names that for convention should be the same of their respective

modules: for example, simple module X is implemented in the C++ class X, declared in X.h and defined in X.cc.

cSimpleModule mainly provides three functionalities:

- module initialization (function *initialize()*: during the setup of a simulation, usually the user has to define or initialize variables and data structures, or start sending messages (e.g. starting timers).
- message handling (function *handleMessage()*): every time a new message is sent to the module, the simulator allows the user to handle such message. The user can perform several actions, typically one of the following:
  - send a message to other module/s, using the variety of send()-like functions offered for this purpose. Such functions allow to send a message after a given delay or processing time, via a certain channel, or send it directly to another module.
  - schedule an event to be delivered to the module itself (i.e. issuing a timer) using the method *scheduleAt()*.
  - cancel an event that has been previously scheduled (with *scheduleAt()*), using the function *cancelEvent()*.
- module finalization (function *finalize()*: allow the user to collect statistical data, perform clean up, etc. at the end of the simulation.

It is worth to note how model behavior is captured by C++ files as code, while model topology (and the parameters defining the topology) is defined in the NED files. This approach is useful to keep different aspects of a simulation separated as much as possible.

A.5 Simulation execution

An Omnet++ simulation is a collection of .h and .cc files for each module defined. A .ned file is also necessary to specify the network topology, i.e. single and compound modules and their interconnections. A separate file, named

omnetpp.ini, is defined for general simluation settings (simulation time limit, memory size, file names, etc.) and the definition of module paramters. Omnet++ can use different random streams per each run, their number (up to 32) defined also in *omnetpp.ini*.

Simulation executions are conceptually structured in a hierarchy, starting from a model and arriving to a single run of the program:

- *model*: executable (C++ files and libraries) and NED files.
- *study*: a series of *experiments* to study some phenomenon on one or more models. Note that a study may contain experiments on different modules, but an experiment is always performed over one specific model.
- *experiment*: exploration of a single paramater on a model, e.g. network behavior with different number of hosts (5, 10, 20, 50, 100). It consists of several measurements.
- *measurement*: a set of simulation runs on the same model with the same parameters. The difference is typically in the parameter and simulation settings in the omnetpp.ini file.
- *replication*: one repetition of a measurements, using different seeds.
- *run*: one actual run of the program.

A single simulation run can be executed in different environments: *Envir*, when simulation is run within the simulation IDE; *Cmdenv* when user executes the simulation in a terminal, useful for batch execution; *Tkenv*, a Tcl/Tk GUI, provides automatic animation of the simulation execution and displaying messages exchanged and module movements. It also allows to inspect the values assumed by specific modules.

Cmdenv is the fastest in terms of execution time, Envir is slower due to the interaction with the IDE, and Tkenv is the slowest because of the many components to display and update on the screen, recommended for teaching and debugging purposes only. Figure A.3 shows screenshots of the Tkenv GUI.

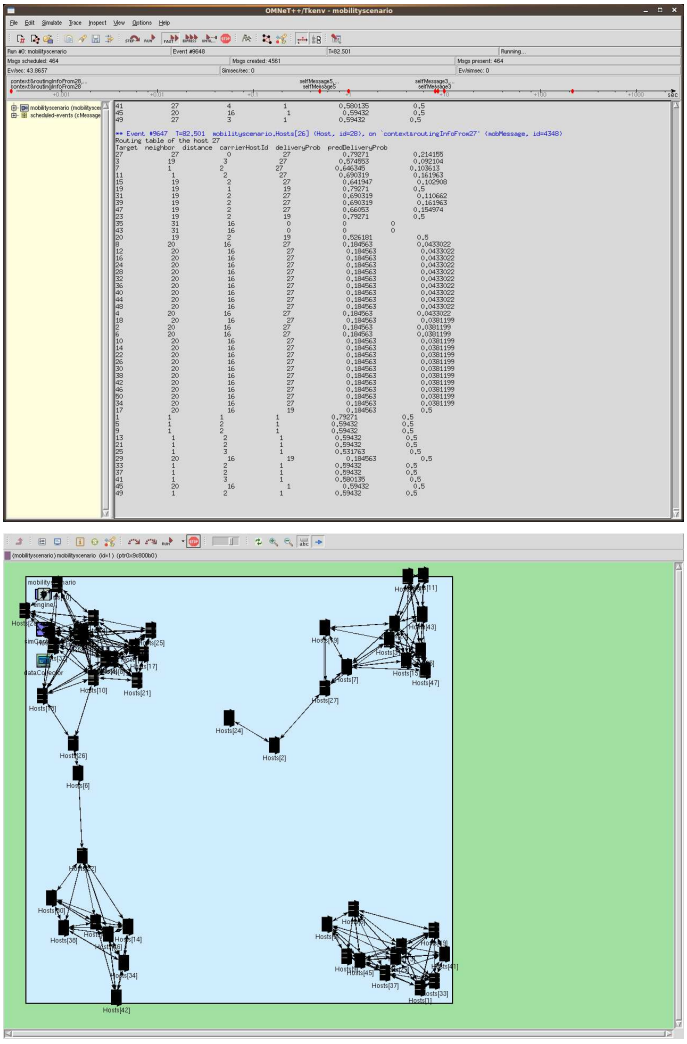


Figure A.3: Omnet++: screenshots of the Tkenv GUI environment

Bibliography

[1] INET Framework for OMNeT++ Web Page. site: *http://inet.omnetpp.org*.

[2] *OMNeT++ online manual*. site: *http://www.omnetpp.org/doc/manual/usman.html*.

[3] Social Network Founded Mobility Models for Ad Hoc Network Research Web Page. site: *http://www.cl.cam.ac.uk/research/srg/netos/mobilitymodels*.

[4] The Mobility Framework for OMNeT++ Web Page. site: *http://mobility-fw.sourceforge.net*.

[5] N. Asokan, K. Kostianinen, P. Ginzboorg, J. Ott, and C. Luo. Towards securing disruption-tolerant networking. *Nokia Research Center, Tech. Rep. NRC-TR-2007-007*.

[6] D.P. Bertsekas and R.G. Gallager. Distributed asynchronous bellman-ford algorithm. *Data Networks*, pages 325–333, 1987.

[7] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine*, 41(6):128–136, 2003.

[8] S. Capkun, L. Buttyan, and J.P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on mobile computing*, 2(1):52–64, 2003.

[9] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, E. Travis, and H. Weiss. Interplanetary internet (ipn): Architectural definition. *Relatório técnico, IPN Research Group*, 2001.

[10] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Pocket switched networks: Real-world mobility and its consequences for opportunistic forwarding. *University of Cambridge, Computer Lab, Tech. Rep. UCAM-CL-TR-617, Feb*, 2005.

[11] BP Crow, I. Widjaja, LG Kim, and PT Sakai. IEEE 802.11 wireless local area networks. *IEEE Communications magazine*, 35(9):116–126, 1997.

[12] Gianluca Dini and Angelica Lo Duca. A reputation-based approach to tolerate misbehaving carriers in delay tolerant networks. In *15th IEEE Symposium on Computers and Communications*, Riccione, Italy Note: accepted, 6 2010.

[13] D. Djenouri, L. Khelladi, and AN Badache. A survey of security issues in mobile ad hoc and sensor networks. *IEEE Communications surveys & tutorials*, 7(4):2–28, 2005.

[14] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, page 34. ACM, 2003.

[15] K. Fall and M. Demmer. Disruption/Delay-Tolerant Networking Tutorial. site: *http://www.cs.berkeley.edu/~demmer/talks/dtn-tutorial-mobihoc-may06.ppt*.

[16] DJ Goodman, J. Borras, NB Mandayam, and RD Yates. Infostations: A new system model for data and messaging services. In *1997 IEEE 47th Vehicular Technology Conference*, volume 2, 1997.

[17] T. Grandison and M. Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4):2–16, 2000.

[18] C. Hedrick et al. Routing information protocol. Technical report, Citeseer, 1988.

[19] T. Henderson, D. Kotz, and I. Abyzov. The changing usage of a mature campus-wide wireless network. *Computer Networks*, 52(14):2690–2712, 2008.

[20] M. Ilyas. *The handbook of ad hoc wireless networks*. CRC, 2002.

[21] D.B. Johnson, D.A. Maltz, J. Broch, et al. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad hoc networking*, 5:139–172, 2001.

[22] R.L. Keeney and H. Raiffa. *Decisions with multiple objectives: Preferences and value tradeoffs*. Cambridge Univ Pr, 1993.

[23] P. Kyasanur and N. Vaidya. Detection and handling of MAC layer misbehavior in wireless networks. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 173–182. Citeseer, 2003.

[24] J. Leguay, T. Friedman, and V. Conan. DTN routing in a mobility pattern space. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, page 283. ACM, 2005.

[25] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *Service Assurance with Partial and Intermittent Resources*, pages 239–254, 2004.

[26] J. Liu and V. Issarny. Enhanced reputation mechanism for mobile ad hoc networks. *Trust Management*, pages 48–62, 2004.

[27] C. Mascolo and M. Musolesi. SCAR: context-aware adaptive routing in delay tolerant mobile sensor networks. In *Proceedings of the 2006 international conference on Wireless communications and mobile computing*, page 538. ACM, 2006.

[28] M. McNett and G.M. Voelker. Access and mobility of wireless PDA users. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(2):55, 2005.

[29] M. Musolesi and C. Mascolo. Designing mobility models based on social network theory. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(3):70, 2007.

[30] M. Musolesi and C. Mascolo. Car: Context-aware adaptive routing for delay-tolerant mobile networks. *IEEE Transactions on Mobile Computing*, 8(2):246–260, 2009.

[31] L. Pelusi, A. Passarella, and M. Conti. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Communications Magazine*, 44(11):134–141, 2006.

[32] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the conference on Communications architectures, protocols and applications*, page 244. ACM, 1994.

[33] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *wmcsa*, page 90. Published by the IEEE Computer Society, 1999.

[34] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[35] G. Resta and P. Santi. An analysis of the node spatial distribution of the random waypoint mobility model for ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, page 50. ACM, 2002.

[36] J. Scott. Social network analysis. *Sociology*, 22(1):109, 1988.

[37] RC Shah, S. Roy, S. Jain, W. Brunette, I. Res, and WA Seattle. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *2003 IEEE International Workshop on Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE*, pages 30–41, 2003.

[38] T. Small and Z.J. Haas. The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, page 244. ACM, 2003.

[39] T. Spyropoulos, K. Psounis, and 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*, page 259. ACM, 2005.

[40] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical report, Technical Report CS-200006, Duke University, 2000.

[41] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[42] D.J. Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton Univ Pr, 2003.

[43] M. Weiser. Ubiquitous Computing. *Computer*, 26:71–72, 1993.

[44] M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1995.

[45] J. Widmer and J.Y. Le Boudec. Network coding for efficient communication in extreme networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, page 291. ACM, 2005.

[46] R.N. Wright, P.D. Lincoln, and J.K. Millen. Efficient fault-tolerant certificate revocation. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, page 24. ACM, 2000.

[47] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *IEEE INFOCOM*, volume 2, pages 1312–1321. Citeseer, 2003.

[48] G. Zacharia and P. Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14(9):881–907, 2000.

[49] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 187–198. ACM, 2004.