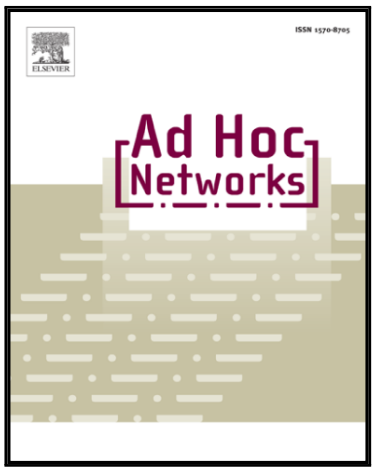


Mobile Matrix: Routing under Mobility in IoT, IoMT, and Social IoT

Bruno P. Santos, Olga Goussevskaia, Luiz F.M. Vieira,
Marcos A.M. Vieira, Antonio A.F. Loureiro

PII: S1570-8705(18)30241-5
DOI: 10.1016/j.adhoc.2018.05.012
Reference: ADHOC 1680

To appear in: *Ad Hoc Networks*

Received date: 22 February 2018
Accepted date: 21 May 2018

Please cite this article as: Bruno P. Santos, Olga Goussevskaia, Luiz F.M. Vieira, Marcos A.M. Vieira, Antonio A.F. Loureiro, Mobile Matrix: Routing under Mobility in IoT, IoMT, and Social IoT, *Ad Hoc Networks* (2018), doi: 10.1016/j.adhoc.2018.05.012

Mobile Matrix: Routing under Mobility in IoT, IoMT, and Social IoT

Bruno P. Santos[a], Olga Goussevskaia[a], Luiz F. M. Vieira[a], Marcos A. M. Vieira[a], Antonio A.F. Loureiro[a]

[a]*Computer Science Department, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil*

Abstract

The explosive growth of "things" connected to the Internet (Internet of Things, IoT) raises the question of whether existing ready-to-go networking protocols are enough to cover social and mobile IoT's demands. IoT aims to interconnect static devices attached to some physical infrastructure. However, mobility is a major factor present in everyday life, and naturally the "things" can move around (Internet of Mobile Things, IoMT) and create social ties (Social IoT, SIoT) in the cyber-physical space. In that context, we present Mobile Matrix (μMatrix), a routing protocol that uses hierarchical IPv6 address allocation to perform any-to-any routing and mobility management without changing a node's address. In this way, device mobility is transparent to the application level favoring IoMT and SIoT implementation and broader adoption. The protocol has low memory footprint, adjustable control message overhead, and it achieves optimal routing path distortion. Moreover, it does not rely on any particular hardware for mobility detection (a key open issue), such as an accelerometer. Instead, it uses a passive mechanism to detect that a device has moved. We present analytic proofs for the computational complexity and efficiency of μMatrix, as well as an evaluation of the protocol through simulations. We evaluate the protocol performance under human and non-human mobility models. For human mobility, we generated mobility traces using Group Regularity Mobility (GRM) Model, setting its parameters based on real human mobility traces. For the non-human mobility, we propose a new mobility model, to which we refer as Cyclical Random Waypoint (CRWP), where nodes move using a simple Random Waypoint and, eventually, return to their initial position. We compared μMatrix with three baseline protocols: Routing Protocol for low-power and lossy networks (RPL), Mobility Management RPL (MMRPL), and Ad hoc On-Demand Distance Vector (AODV). The results show that μMatrix and RPL offer $\approx$ 99.9% of bottom-up delivery rate, but only μMatrix offer $\geq$ 95% of top-down traffic in highly dynamic and mobile scenarios, while other protocols $\leq$ 75%. Moreover, μMatrix uses up to 65% of the routing table while RPL and AODV fulfill theirs in all scenarios, which leads to poor top-down and any-to-any reliability.

Keywords: Internet of Things, Mobility, Hierarchical Address, Routing protocol

1. Introduction

Internet of Things (IoT) has become a reality with the explosive adoption of smart environments, where everyday objects ("things") are capable of communicating through the Internet. Usually, IoT is a set of interconnected *static* "things" forming a cyber-physical environment. For example, a smart grid composed by smart meters and smart sensors into a smart building. However, mobility is a major factor present in human and non-human life. It makes life easier and turns smart application more flexible and suitable in the mobile world. Nowadays, we already have mobile phones and vehicles in the IoT, in the near future we will have further mobile devices. Naturally, IoT will need to evolve encompassing mobile things (IoMT) and, furthermore, such devices will be able to develop social ties (Social IoT) in the cyber-physical space. With this evolution, IoT takes a step towards ubiquitous computing, where virtually everything is connected with everything at anytime and anywhere.

In the literature, one can find several applications and service proposals for IoMT and SIoT [1, 2, 3, 4]. However, they assume that the networking stack is capable of meeting their requirements of mobility management, memory,

Email addresses: bruno.ps@dcc.ufmg.br (Bruno P. Santos), olga@dcc.ufmg.br (Olga Goussevskaia), lfvieira@dcc.ufmg.br (Luiz F. M. Vieira), mmvieira@dcc.ufmg.br (Marcos A. M. Vieira), loureiro@dcc.ufmg.br (Antonio A.F. Loureiro)

energy, and processing efficiency. Actually, we find that we are far from covering all issues imposed by mobility, especially, regarding very tiny devices with resource constraints.

IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) defines standards for low-power devices that comprise the IoT. Standard routing protocols have been defined, such as CTP [5] and RPL [6], and are widely used to build IoT applications and services. Nonetheless, they do not always meet the requirements of IoT, IoMT, and SIoT apps and services, such as the mobility management, any-to-any communication, memory efficiency, and others [7].

To address these challenges, we present Mobile Matrix (μMatrix), a complementary solution to standard routing protocols for IoT, which provides better support for IoMT and SIoT. μMatrix is a routing protocol that uses hierarchical IPv6 address allocation to manage mobility without changing a node's IPv6 address, while being memory efficient, favoring resource-constrained devices. μMatrix manages mobility transparently to the upper networking layers in the stack, therefore favoring IoMT and SIoT implementation and adoption. One of the building blocks of μMatrix is a passive mechanism, named Reverse Trickle Timer (RevTT), to detect mobility in a device's neighborhood. Therefore, the protocol does not need extra hardware to operate. Given that there is an intrinsic trade-off between the delay to detect mobility and the number of control messages, one can tune μMatrix's frequency of control messages according to the application or the mobility pattern.

μMatrix is built upon a previously proposed protocol, named Matrix [8]. Besides integrating mobility management into Matrix, μMatrix presents the following features:

- *Transparent mobility management*: devices can move in the cyber-physical space without ever changing their IPv6 address;
- *Optimal routing path distortion*: messages addressed to a mobile device, from anywhere in the network, are sent along the shortest path from the source to its current location, using its original IPv6 address;
- *Any-to-Any routing*: devices running μMatrix are able to not just perform bottom-up data collection or top-down dissemination, but also to send messages to any other device in the 6LoWPAN;
- *Passive mobility detection*: μMatrix uses RevTT to detect neighbor device mobility, which can be tuned according to the application or the mobility pattern;
- *Low memory footprint*: μMatrix is in consonance with IPv6 addressing and uses a hierarchical address allocation to reduce memory usage to store routing information in a dynamic mobile environment;
- *No fixed devices required*: μMatrix does not rely on fixed devices to manage mobility, except for the border router, commonly employed in 6LoWPAN;
- *Link dynamics and fault-tolerance support*: μMatrix inherits from Matrix the capacity to overcome temporary device failure or link dynamics by rerouting the data flows using a local broadcasting mechanism [8];
- *Platform-independent*: μMatrix does not rely on any specific platform or extra hardware (e.g., GPS, accelerometer) to operate;

Moreover, we propose a new mobility model, to which we refer as *Cyclical Random Waypoint mobility model (CRWP)*. In CRWP, nodes are assigned a home location and might make several moves in random directions, connecting to the 6LoWPAN at different attachment points (and forming social ties), and eventually return to their home locations. Our motivation for proposing a new mobility model comes from application scenarios, where communication is carried out in environments with limited mobility, such as 6LoWPANs deployed in an office or school buildings, university campuses or concert halls or sports stadiums.

The main contributions of this paper can be summarized as follows:

1. We present μMatrix, a communication protocol that performs hierarchical IPv6 address allocation and manages routing and mobility without ever changing a node's IPv6 address. The protocol favors the implementation and adoption of IoT, IoMT, and SIoT with constrained devices. μMatrix has low memory footprint, adjustable control message overhead and achieves optimal routing path distortion;

Feature	μMatrix	RPL	Co-RPL	MMRPL	MV-RPL	ME-RPL	mRPL	DMR	XCTP	Hydro
Bottom-up	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Top-Down	✓	✓	✓	✓	✓	✓	✓		✓	✓
Any-to-any	✓	✓	✓	✓	✓	✓	✓			✓
IPv6 support	✓	✓	✓	✓	✓	✓	✓	✓		
Address Allocation	✓									
Memory efficiency	✓									
Fault Tolerance	✓									✓
Local Repair	✓									
Mobility Detection	RevTT	T	P	RTL	P	T	T	T	T	P
Constraints	Nodes have a home location			Static nodes		Static nodes	Static nodes	Static nodes	Static nodes	

Table 1: Routing protocol properties (RevTT – Reverse Trickle Timer; RTL – Trever Trickle Timer Like; T – Trickle; P – Periodic)

2. We provide analytic proofs for the computational complexity and efficiency of μMatrix, as well as an evaluation of the protocol through simulations. We evaluate μMatrix under human and non-human mobility patterns, therefore showing its functionality;

3. An essential building block of μMatrix is the Reverse Trickle Timer, a passive mobility detection mechanism that captures changes in topology without requiring additional hardware (e.g., accelerometer, GPS or compass).

4. We propose the *Cyclical Random Waypoint mobility model (CRWP)*, a new mobility model for scenarios where devices have a home location and perform periodic random movement patterns.

5. The source code of μMatrix was made publicly available, so all experimental results presented in this work can be reproduced[1].

The rest of this paper is organized as follows. We discuss some related work in Section 2. The design overview of Matrix Mobile is presented in Section 3. We analyze the message complexity of the protocol in Section 4. We describe the mobility modeling in Section 5. In Section 6, we present our simulation results. Finally, in Section 7, we present the concluding remarks.

2. Background and Related Work

Wireless Sensor Networks (WSN) are a type of network, where usually tiny static devices are employed to sense, process, store and communicate information surrounding the device. With the integration of Internet Protocol (IP) to WSN emerges the IoT. The concept appeared early 1982, but its real implementation and adoption started in the last years [9]. More recently, two new paradigms have arisen from IoT: the Social Internet of Things (SIoT) [2] and the Internet of Mobile Things [4]. Those new paradigms hold a common characteristic: their devices are no longer static, but are able to move by itself or are attached to mobile entities.

Several mobility-enabling routing protocols have been proposed for IoT. Table 1 summarizes properties of these routing protocols. We use a check mark if the protocol has the feature or empty otherwise. Ten features have been considered, which are related to traffic patterns, addressing, memory, reliability, and protocol limitations.

RPL [6] is a well-known standard routing protocol for 6LoWPANs. Nevertheless, it presents some limitations, in particular in mobility scenarios, such as scalability issues, reliability and robustness for top-down traffic [7, 8]. Most recent mobile-enabled routing protocols are RPL extensions [10]. They focus on mobility issues but not always handle the drawbacks of RPL.

Co-RPL [11] provides mobility support for RPL but does not take advantage of dynamic features of the Trickle Timer algorithm, which is intrinsic to RPL. This turn Co-RPL more responsive by using the corona mechanism, but it has a higher overhead. Co-RPL builds on top of RPL's strategies to build and repair downwards routes, which is inefficient regarding memory.

[1] `https://bps90.github.io/mmatrix-code/`

MMRPL [12] modifies the RPL beacon periodicity by replacing Trickle with a reverse Trickle-Like mechanism. Thus, their reverse Trickle decays exponentially as longer as a node stay attached to the same parent. MMRPL assumes that as longer as a node remains attached to a parent, the higher will be the probability of the node to move. Also, MMRPL assumes that some nodes are static, which helps its mobility management, therefore requiring heterogeneous devices and more complex code to operate.

MRPL-V [13] modifies RPL to work in vehicular networks. The protocol replaces the Trickle mechanism by a periodic fixed timer to fire RPL control messages. However, since RPL is not built to support highly dynamic scenarios, like vehicular ones, using periodic beaconing to advertise topology changes might be too fast, while the global routing repair is too slow to keep with the changes. Thus, downwards routing can present poor reliability.

ME-RPL [14] also assumes that some nodes are static and others can move. In the RPL route discovery phase, when a node is choosing its preferred parent, static nodes have higher priority than mobile ones. When a node is detached from a parent, it sends RPL DIS messages in dynamic intervals. Such alterations turn ME-RPL responsive to mobile nodes rejoining the RPL tree and identify mobile nodes. However, the memory requirement to maintain downward routes is still prohibitive.

mRPL [15] is based on the so-called smart-HOP algorithm, a hand-off mechanism for mobile nodes in RPL. They separate nodes into mobile nodes (MN) or serving access points (AP). AP are static and MN nodes use AP nodes as parents in the RPL routing tree.

DMR [16] enhances RPL to support data transfer in mobile nodes. DMR removes some features from RPL like top-down and any-to-any traffic handling. DMR also stores routing information to choose the best parent to deliver data reliably.

XCTP [17] extends CTP [5] to support bidirectional traffic and manage mobility. However, XCTP uses a flat address structure instead of IPv6 addressing. Also, XCTP does not fully support any-to-any routing. Hydro [18] fills the gap of any-to-any communication traffic, but it requires static nodes with a large memory to maps nodes' current locations. In fact, XCTP and Hydro were created for static scenarios, but can operate under mobility.

Mobile IPv6 (MIPv6) [19] supports mobility among different domains by assigning multiple IP addresses to each mobile node: a fixed home address and a Care-of Address (CoA), which changes depending on the current subnet where the node is. A home agent (HA), a fixed entity, is required to manage the mobility and map the addresses. All data traffic is firstly routed to the HA, and then to the CoA, thus MIPv6 does not use the shortest path to routing data flow, presenting a sub-optimal routing path distortion. Hierarchical Mobile IPv6 (HMIPv6) [20] enhances MIPv6 by reducing the signaling load among mobile devices. However, these protocols were not designed for 6LoWPANs, and they do not present a mobility detection mechanism, nor adjustable timers to handle network dynamics.

Protocols for mobile ad hoc networks, like AODV [21] and OLSR [22], have high memory footprint and control message overhead, which makes them not suitable for low-power devices or 6LoWPAN. Actually, there are efforts to extend those for 6LoWPAN such as LOAD [23] and DYMO-Low [24]. These protocols were inspired by AODV and DYMO, but they still present drawbacks in mobile scenarios in terms of memory.

Differently from most of the RPL-based protocols, in this work, we present a mobility-enabling routing protocol that solves many of RPL's drawbacks. μMatrix presents low memory footprint for top-down and any-to-any routes under mobility, transparent mobility management, optimal routing path distortion, and fault-tolerance support. Thus, μMatrix is designed to support IoMT and SIoT implementation and broader adoption.

This article is an extension of a preliminary conference version [25]. The journal version includes new experiments with different mobility models (as opposed to the single CRWP model used in [25]), simulating real-world mobility traces with variable characteristics, such as inter-contact time and number of encounters, which serve as a validation of μMatrix in mobile scenarios, envisioned for IoMT and SIoT. Moreover, additional baseline protocols were simulated (MMRPL and AODV) and optimizations were performed in the protocol implementation, increasing the data delivery success rate from approx. 70% to 95%+ in the worst-case simulated mobility scenarios.

3. Design Overview

As mobility is a new factor to IoT, a question arises: where can we handle mobility in IoT? It is possible to handle mobility with different purposes in different layers of the IoT network protocol stack. However, we argue that mobility in the network layer plays a crucial role in the entire IoT operation in mobile scenarios. Firstly, this is due

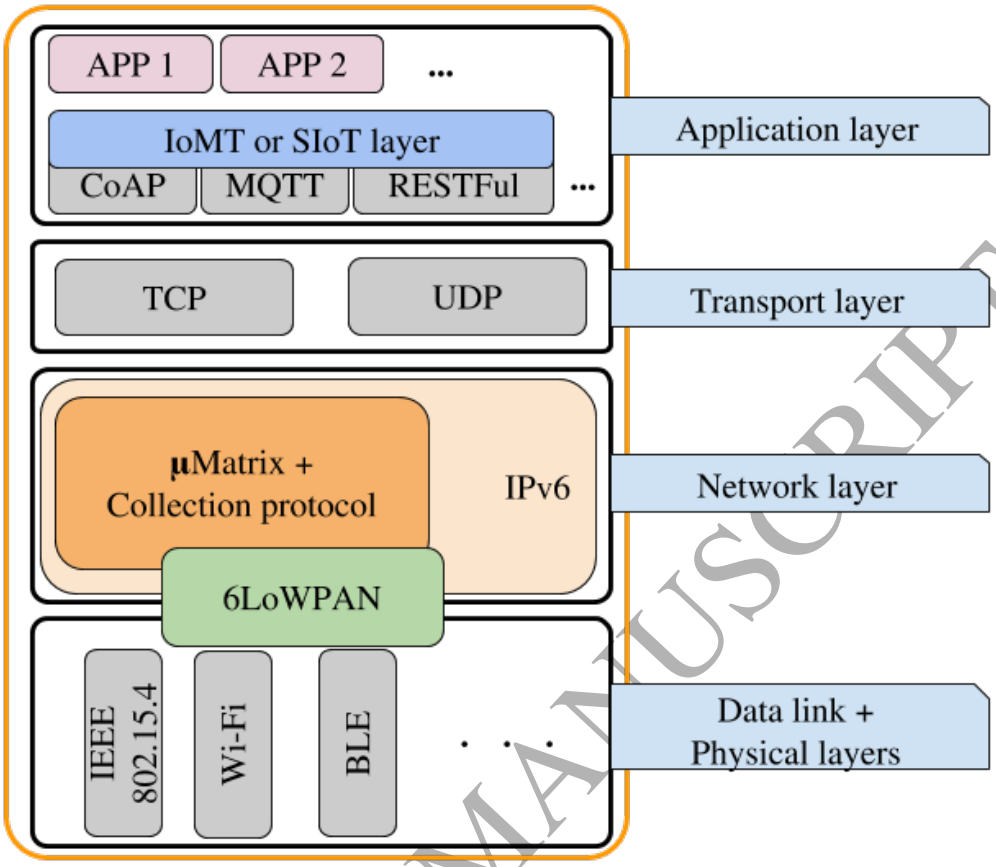


Figure 1: μMatrix integrated into the network stack.

to the need to maintain routes under device mobility. Secondly, if the device address changes, all protocol layers above the network layer need to be aware of it, increasing the overall protocol complexity. Finally, if devices have constrained resources, then managing routes for mobile devices can be expensive in terms of memory and energy.

There are three data traffic patterns that a routing protocol should provide for IoT, IoMT and SIoT applications [7]:

1. Bottom-up,
2. Top-down,
3. Any-to-any.

The first type of traffic pattern provision is the primary function of standards protocols, such as CTP [5] and RPL [6], and efficient implementations are widely available. However, top-down and any-to-any traffic implementation is not always supported or is not optimized for performance by standard protocols. Therefore, μMatrix is designed to function on top of an existing bottom-up, or collection, scheme (we use CTP in our implementation), and adds an efficient solution for the remaining two data traffic patterns, besides providing support for mobility.

3.1. Mobile Matrix Architecture

Figure 1 shows how μMatrix is integrated into the overall network protocol stack. μMatrix is implemented in the network layer and is comprised of two planes and sub-modules to handle routing states and manage mobility, as illustrated in Figure 2:

- **Control plane:** hierarchical IPv6 address allocation, routing table maintenance, and mobility management;

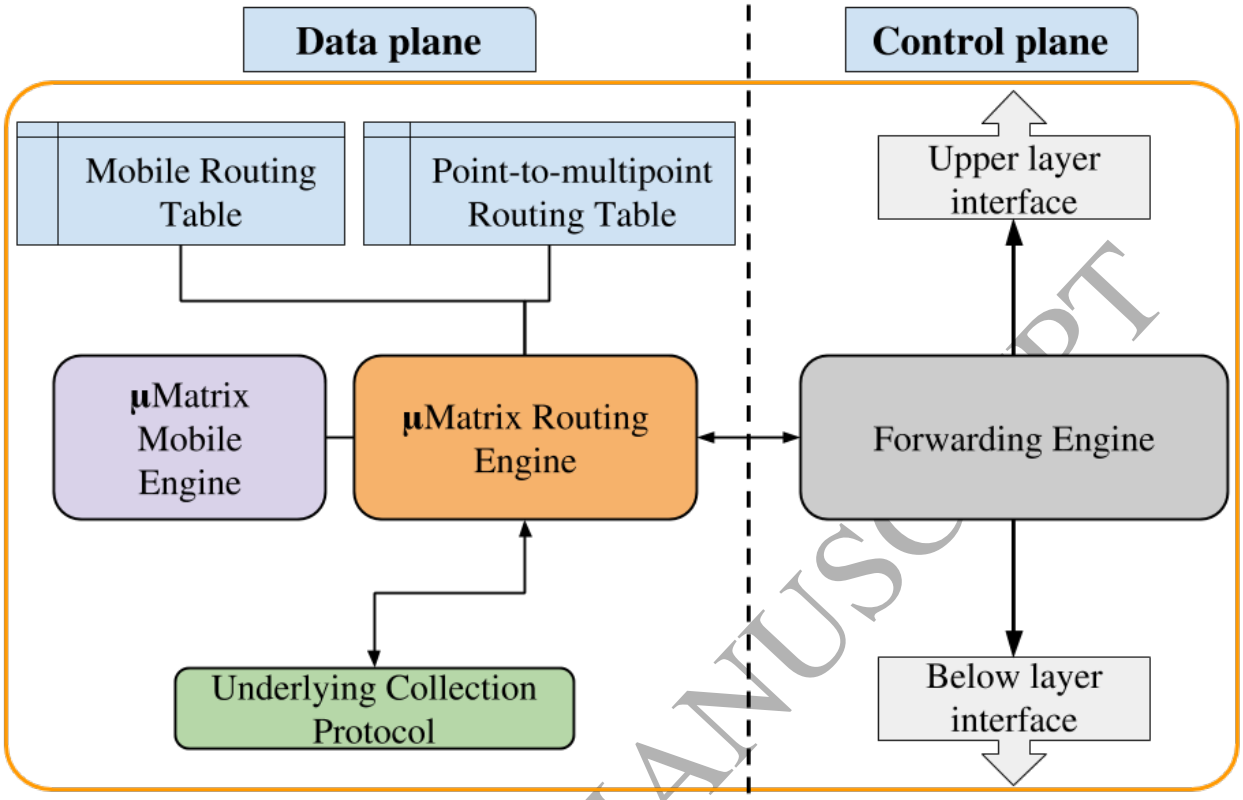


Figure 2: μMatrix protocol's architecture.

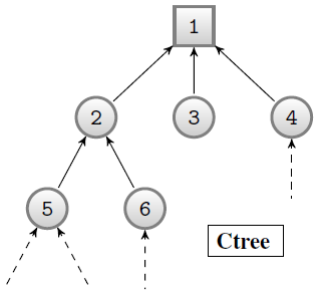
- **Data plane:** routing information querying and data and control forward data and control packet forwarding.

μMatrix operates according to the following phases, described in detail in Sections 3.2 through 3.4:

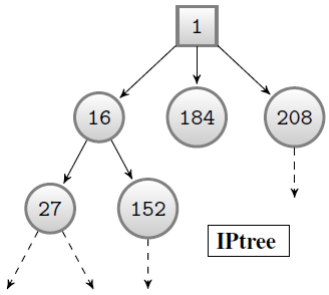
1. **Collection tree initialization:** an underlying bottom-up routing protocol (e.g., CTP or RPL) creates a collection routing tree, which is dynamically updated to reflect current connectivity conditions;
2. **Hierarchical IPv6 address allocation:** once the collection tree was built, μMatrix performs a convergecast to gather information about the network's initial topology, which is used to partition the available IPv6 address space in a hierarchical way among all nodes.
3. **Packet forwarding:** bottom-up data traffic is forwarded along the collection tree built and maintained in phase (1); top-down data is forwarded using IPv6 addresses allocated in phase (2); any-to-any packet forwarding is performed by combining the routing structures maintained in phases (1) and (2).
4. **Mobility management:** μMatrix uses additional routing data structures to reflect the topology changes due to device mobility.

3.2. Control Plane: Routing Engine

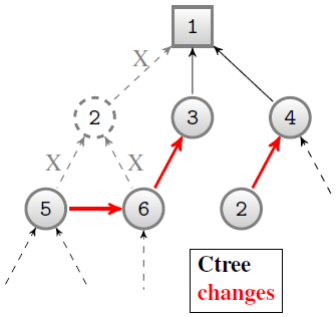
The control plane manages all routing table structures and makes decisions based on information from other modules, such as mobile or forwarding engines and the underlying collection protocol (Figure 2). In this section, we describe the basic routing functionalities of the control plane of μMatrix, in the following order: routing engine data structure (Section 3.2.1), control packets and parameters (Section 3.2.2), IPv6 multihop host configuration (Section 3.2.3). Then, in Section 3.3, we present the mobile engine of μMatrix.



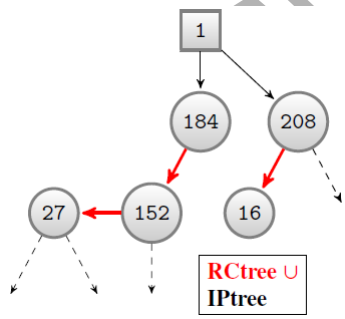
(a) Ctree structure, ID inside the nodes.



(b) IP addres assignment by MHCL hierarchical distribution. Simplified 8-bit IP inside the nodes.



(c) Node 2 moves, then Ctree changes.



(d) *RCtree* ∪ *IPtree*. Red and thicker links are in RCtree, but not in IPtree.

Figure 3: Routing structures: Ctree, IPtree, and RCtree.

3.2.1. Routing Engine: data structures

μMatrix and the underlying collection protocol build and maintain three routing trees structures:

1. **Ctree**: a collection tree built by the underlying collection protocol (e.g., CTP [5] or RPL [6]). CTP was used as a data collection routing protocol providing efficiently bottom-up routes. We chose CTP purposely due to its lower code complexity than RPL's [8], robustness, loop avoidance, and its wide acceptance by the community.

2. **IPtree**: an IPv6 hierarchical tree is created using MHCL algorithm [8] at μMatrix initialization phase. The IPtree is kept static, except when new devices join the network[2];

3. **RCtree**: a tree that reflects the topology changes caused by devices mobility.

Figure 3 shows those routing structures graphically. Initially, in Figure 3(a), the underlying collection protocol builds the Ctree structure, note that we have a border router (node 1) that starts the process. Then, μMatrix starts the MHCL algorithm to distribute the available range of IPv6 hierarchically[3]. At this moment, IPtree = $Ctree^R$ and RCtree = $\emptyset$ (see Figure 3(b)). Whenever a topology change occurs due to mobility in Ctree, a new reverse link is added into RCtree, and it is maintained while the change remains, therefore $RCtree = Ctree^R \setminus IPtree$ (see Figures 3(c)(d)). RCtree is not essentially a tree since it contains only reversed links in Ctree but not in IPtree. Nevertheless, $RCtree \cup IPtree$ is, in fact, a tree, which μMatrix uses to downward routing. Each node η keeps the following information to build and maintain those trees:

[2]As described in detail in [8], each node is assigned a reserve address space for nodes joining the network after the initialization phase.
[3]The network operator defines this range of IPs, which will be distributed along the IPtree [8].

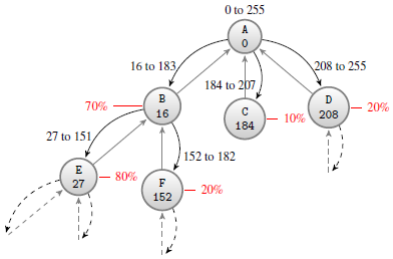


Figure 4: Simplified hierarchical address assignment with 8-bit available address space and 6.25 % of addresses reserved for delayed nodes. Inside the nodes, its label and IP assigned, the % next to the nodes express the approximate sub-tree size. Thick downwards arrows indicate the available IP range fairly distributed.

- $CT\,parent(\eta)$: the ID of the current parent of a node η in the Ctree;
- $PRV\,parent(\eta)$: the ID of η's previous $CT\,parent(\eta)$;
- $IP parent(\eta)$: the ID of the node that assigned to η its IPv6 and IP range;
- $IP children(\eta)$: the standard (top-down) routing table with IPv6 ranges for one-hop descendants of η in the IPtree;
- $Mtable(\eta)$: a temporary alternative routing table for mobility management. Each *Mtable* entry has the following fields: IPv6 range, next hop, and Time Has Lived (THL).

3.2.2. Control Packets and Parameters

μMatrix introduces one new control packet and one parameter to exchange topology information and update the *Mtables* upon mobility events:

1. `nodeInfo` frame has 7 fields: seqNum, IP Node, IP range, IPparent, CTparent, TTL, and Type. The fields are self-explanatory, except by the type field, which specifies if the frame is a `keepRoute` to an entry into the *Mtable*, or `rmRoute` to remove an entry. Following, we will use `keepRoute` or `rmRoute` for short;
    - A node sends `keepRoute` beacons to inform its current location and the in-network nodes can update its *Mtables* to reflect the present network topology;
    - `rmRoutes` is an optional beacon. It is sent when nodes move from a location to another in order to quickly remove inconsistent states in-network nodes.
2. δ parameter specifies the time between sending two consecutive `nodeInfo` packets. Note others time-based strategies can be used than the periodic one.

A device fills its *Mtable* by receiving `keepRoute` beacons from mobile nodes. The node keeps *Mtable* entries as long as it receives `keepRoute` (see Section 4.1 for *Mtable* memory footprint analysis). Otherwise, it uses a THL-based mechanism or `rmRoutes` to remove entries. In static scenarios any node stores one-hop neighborhood information in $IP parent(\eta)$ table, this requires $O(k)$ entries, where k is the number of direct children of a node in the Ctree. This memory footprint is better than state-of-the-art, e.g., RPL would need at least 1 routing entry for every child in a node sub-tree to perform top-down routing.

3.2.3. IPv6 Multihop Host Configuration

μMatrix relies on an underlying collection routing protocol to build the Ctree. Once the Ctree is stable[4], the address space available to the border router, e.g., the 64 least-significant bits of the IPv6 address (or its compressed

[4]A node is stable in the Ctree if it reaches k times the maximum maintenance beacon period of Ctree protocol without changing its parent. We use Trickle Timer [26] as beacon scheme.

16-bit representation), is hierarchically partitioned among nodes in the Ctree by using the MHCL algorithm [8]. The (top-down) address distribution is preceded by a (bottom-up) convergecast phase, in which each node counts the total number of its descendants and propagates it to its parent. Thus, a node knows how many descendants each child has. Such information is required to distribute IP ranges in a fairly way. As result of this procedure is obtained the IPtree.

Figure 4 shows the IP host configuration process. First, the underlying routing protocol creates the Ctree (upwards grey arrows), and then, after the Ctree stabilization, the convergecast phase occurs allowing nodes to know the size of theirs sub-tree (% next to each node). Next, the border router starts the IP distribution by auto-setting its IP (e.g., the first available IP from range) and then reserving a portion of the range for delayed nodes. After that, the node distributes the remaining range fairly among its children (e.g., in Figure 4 B receives 70% of available range, i.e., from 16 to 183). Finally, each node repeats the IP distribution process.

3.3. Control Plane: Mobile Engine

The mobile engine plays a central role in the μMatrix operation. It is responsible for identifying when mobility events happen and feed the routing engine with current node status. With this, it helps the routing engine to take action upon mobility events properly.

Following, in Section 3.3.1, we present details of RevTT a passive mobility detection algorithm. Next, in Section 3.3.2, we present the μMatrix finite state machine used to keep track the node's status. Finally, in Section 3.3.3, we present the μMatrix strategies to prevent and recover from the loop.

3.3.1. Mobility Detection

Mobility detection is a crucial issue to handle mobile devices in routing protocols. Most of the related protocols (see Section 2) modifies this component to improve the routing protocol performance under device mobility. There are two classes of mobility detection events: i) active mobility event; ii) passive motion event. In the first one, the devices by using extra hardware (e.g., accelerometer or GPS) inform their motion to the routing protocol to take actions. In the second one, the protocol, by itself, infers the movement of the devices(e.g., by using beaconing mechanisms).

Standards 6LoWPAN routing protocols, such as CTP or RPL, make use of Trickle Timer algorithm [26] that passively detects topology changes. However, Trickle Timer lacks in agility to detect changes in dynamic network and mobile nodes. We propose Reverse Trickle Timer (RevTT) that operates similarly to the standard algorithm, but in reverse order.

RevTT introduces three parameters which the network operator must tune it according to the application and the mobility pattern requirements. Also, RevTT has three methods to manipulate its behavior. The parameters and methods are described below:

1. **Parameters:** I_{max} and I_{min} the maximum and minimum time interval to fire an event. I_k the number of attempts before declaring an inconsistency.

2. **Methods:** *Start* aiming to start the RevTT operation, *Stop* aiming to break the timers, and *Reset* that basically stops the current times using *Stops* method and then the *Start* method.

RevTT operation is straightforward. The algorithm starts with I_{max} interval to fire an event. If *Stop/Reset* methods are not used, then RevTT goes to the I_{min} interval for I_k attempts. If nothing happens during I_k fires, then RevTT triggers an event indicating inconsistency.

Figure 5 shows the RevTT procedure within μMatrix Mobile Engine. First, a node starts sending unicast `hasMoved` beacons to its parent in I_{max} intervals. If the node did not receive an ack for a `hasMoved` beacon, then RevTT sets the interval to I_{min}. After I_k unsuccessful attempts, the node knows that a movement or fault happens. Thus, the node can take actions, for example, properly perform a handover to another parent and then the procedure restarts. Note that by setting the RevTT parameters, the network operator should consider the trade-off between delay to detect mobility and number of beacons. For a smaller delay in mobility detection, I_{max} must be tuned to small values at the cost of more `hasMoved` beacons. In our experiments (see Section 6), RevTT parameters were set according to application data rate.

In [10], the authors argue that a common modification to support mobility is to change the control message periodicity. The typical approach uses a simple periodic timer or the standardized Trickle Timer. While RevTT waits

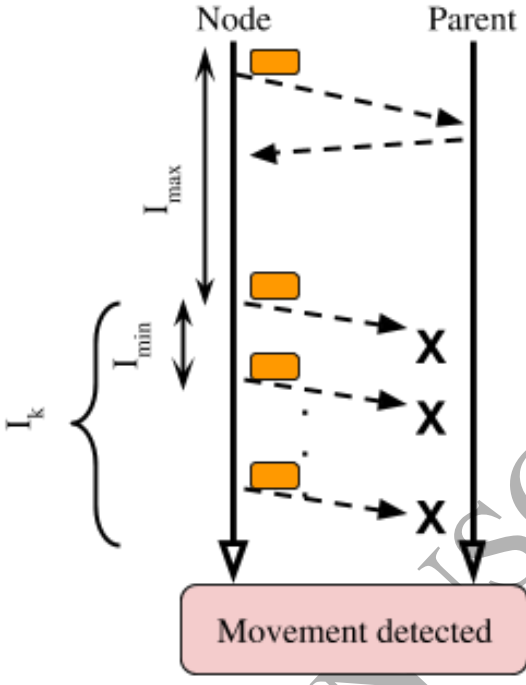


Figure 5: Reserve Trickle Timer operation with μMatrix.

for $I_{max} + T_k \times I_{min}$ to detect a topology change, where $I_{min} \ll I_{max}$, the periodic and standardized Trickle approaches wait for at least $2 \times I_{max}$.

3.3.2. Mobile Engine: Finite State Machine

μMatrix starts the mobile engine as soon as the host configuration finishes. The mobile engine allows the nodes to move around the 6LoWPAN. This module uses a finite state machine, as shown in Figure 6. Each node can be in one state depending on its previous condition and the knowledge about its neighborhood. The engine uses RevTT to recognize mobility and transit among states. Following, we discuss the actions taken in each one of those states.

Each node starts at Home Location (HL) state. In HL, the nodes start RevTT sending a `hasMoved` beacon to its $CTparent(\eta)$. When RevTT identifies an inconsistency and triggers a mobility event, the node transits to Movement Detected (MD) state.

When a node reaches the MD state, it knows that a mobility event happen, but it does not know if itself or its parent moved. There are at last two ways to automatically find out who moved. First, a node can pro-actively queries its children ($IPChildren(\eta)$), if no one answer then the node moved; otherwise the parent moved. Second, a node must wait for a period (e.g., one RevTT I_{max} interval) to receives `hasMoved` beacons from its children and then infer who moved. We use the second approach in our implementation. After discovering who moved, the node goes to Node Moved (NM) or Parent Moved (PM) state.

Several actions are taken when a node reaches NM state. Firstly, the node disables the $IPChildren(\eta)$ table usage due to the node new position in the Ctree. Next, the *Mtable* is cleaned, because it must be outdated. Then, the node triggers the new parent discovery from underlying collection protocol. When the node is attached again to the Ctree, it restarts RevTT with new CTparent and begins sending `keepRoute.IP_ONLY` at a frequency of δ to its IPparent. At NM state, the nodes do not fill the `nodeInfo.IP_RANGE` field because the node is no longer at home location in the IPtree, being incapable of using its $IPChildren(\eta)$ table. Figures 7(a)(b) illustrate this situation. The Figure 7(a) shows the node B before its movement, then when B is attached to a new CTparent (Figure 7b), it sends `keepRoute` beacons to A. The beacons are forwarded upward to the lowest common ancestor LCA(A, B) and then downwards to the node A.

HL	Home Location
MD	Movement Detected
NM	Node Moves
PM	Parent Moves
1	IPparent does not answer
2	Children are active
3	Children are NOT active
4	CTparent does not answer
5	IPparent is back

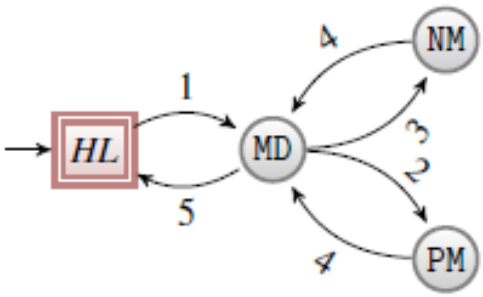


Figure 6: Mobile Engine state machine.

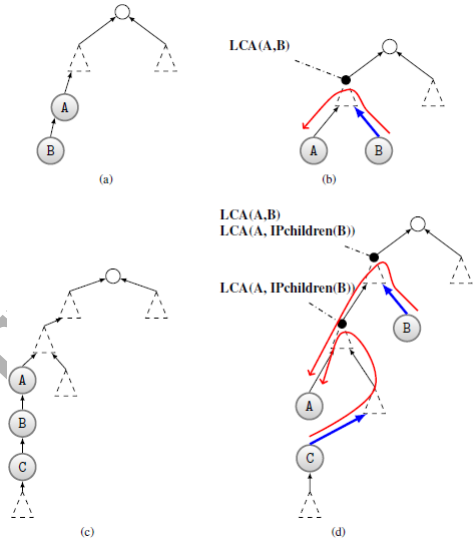


Figure 7: Mobile engine operation after mobility events.

When a node reaches PM, this means that its parent moved. Then, the node triggers the parent discovery mechanism from the underlying collection protocol. When it is attached again to Ctree, it restarts RevTT and starts sending `keepRoute` beacons (if it has children in the IPtree, it fills the `IP_RANGE`) at a frequency of δ. The first beacon is addressed to its grand IPparent. The Figure 7(c)(d) illustrate this situation. The Figure 7(c) shows C, a non-leaf node, before B movement. Then, in Figure 7(d), B moves and then C eventually reaches PM state, next C starts sending `keepRoute` beacons to its $grandIPparent = A$. The messages travel to LCA(A, C) and then to the ultimate destinations. The node B moves to NM state and takes the actions accordingly.

Eventually, nodes return to their home position being attached again to its IPparent in the Ctree. This situation also triggers some actions. First, the node stops to sending `keepRoute` beacons. Also, the returned node restarts the

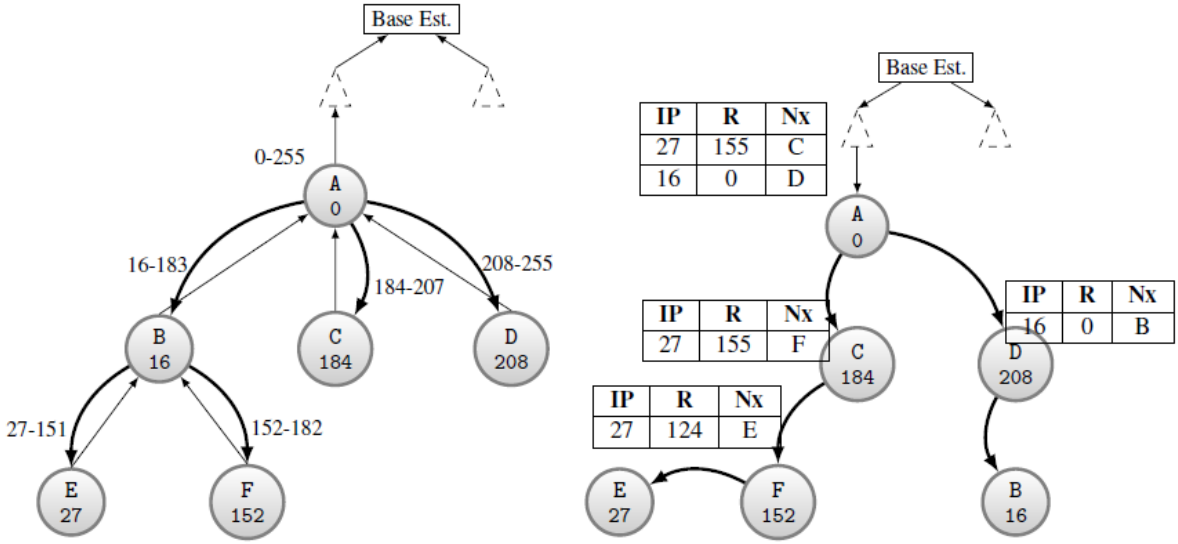


Figure 8: μMatrix's preserves locality when it updates the routing table under mobility events. *Mtables* above LCA do not need updates.

RevTT with its IPparent.

Besides the action in each state, optional actions can be made. If a node is attached to a sequence of CTparent before returning to home location, several states will be installed in-network *Mtables*. Although the *Mtable*' THL field exists to remove inconsistent entries, it is possible to send `rmRoute` beacons to each node's PRVparent to eliminate route inconsistency quickly.

Discussion: note that a sub-tree can move and nodes still hear each other. For instance, in Figure 7(c) suppose A and B move together. Then, A and C eventually will transit to PM, while B and C's sub-tree remain in HL. In this case, the LCA has two *Mtable* entries matching with C's sub-tree, but one more restrictive than other. Thus, the LCAs play a key role, which they always route through the most restrictive *Mtable* range match available.

Also, note that μMatrix preserves locality when manages mobile nodes since no *Mtables* need updates above LCA. Figure 8 illustrates this situation. In Figure 8(a), shows μMatrix's routing structures in a static situation. Then, the node B moves, Figure 8(b), it eventually reaches the NM state, therefore the *Mtables* comprised in the shortest path from B to $A = IPparent(B) = LCA(B, IPparent(B))$ will receive B's `keepRoute` updates. Also, B movement causes E and F to transit from HL to PM state. Then, when E and F eventually find new routes, they will update the *Mtables* comprised between them and its grandIPparent = A.

Note that $Mtable(C)$ and $Mtable(A)$ require only one entry for both E and F sub-trees. We explore the fact of the contiguous IP ranges can be aggregated into unique entries in the *Mtables* improving the memory usage efficiency.

3.3.3. Loop avoidance

Dynamic links and mobile nodes cause turn route information outdated, which may leverage routing loops [5]. μMatrix uses data path validation and adaptive beaconing to detect loops such as CTP and RPL [5, 13]. Besides that, if a node receives more than one unique control packet[5], then this indicates an inconsistency in the tree triggering the control packet suppression and the underlying collection protocol route update. Besides that, *Mtable* and `keepRoute` beacon have, respectively, Time Has Lived (THL) and Time To Live (TTL) fields, which are used to remove inconsistent routes and packets from the network.

[5]The `keepRoute` fields together (see Section 3.2.1) denote a unique packet instance.

3.4. Mobile Matrix Data Plane: any-to-any routing

The Forwarding Engine (see Figure 2) is responsible for data traffic forwarding. Any-to-any routing combines both routing schemes: bottom-up and top-down. The engine uses bottom-up routes to forward packets until the LCA between the sender and receiver, and then it uses top-down routes to forward data to the destination. Upon receiving a data packet, the node checks if the message is for itself. Second, the node tries to match the destination with the most restrictive entry in the *Mtable*. Third, if any *Mtable* entry matches with the target address, then the node checks if the packet destination falls within some range in $IPchildren(\eta)$, if a match occurs, then the node forwards the packet downwards according. Finally, if all previous attempts fail, then the node sends the packet upwards using $CTparent(\eta)$.

4. Complexity Analysis

For the formal analysis, we assume a synchronous communication message-passing model with no faults. Thus, all nodes start executing the algorithm simultaneously and the time is divided into synchronous rounds, i.e., when a message is sent from node v to its neighbor u in time-slot t, it must arrive at u before time-slot $t+1$, and $d(v,u)$ is the shortest path length between v and u in $Ctree \cup IPtree \cup RCtree$. The performance of μMatrix in faulty scenarios is analyzed through simulations in Section 6.

4.1. Memory footprint

As described in Section 3, the temporary routing information needed to maintain mobility is stored in the *Mtable* data structure of some nodes. Each entry is kept for at most THL_{max} seconds, a time interval pre-configured by the network operator, and is deleted unless a `keepRoute` beacon is received. In the following theorem, we bound the total number of *Mtable* entries in the network, necessary to manage the routing of each mobile node $\mu \in CTree$.

Theorem 1. *The memory footprint to manage the mobility of one node $\mu \in Ctree$ with μMatrix is $\mathcal{M}(\mu) = O(depth(Ctree))$.*

Proof. Consider a node $\mu \in Ctree$ that has moved from its home location in time-slot t_0 and returned in time-slot t_f. Consider the permanent $IPparent(\mu)$ and the temporary $CTparent_i(\mu)$ in time-slot $t_0 < t_i < t_f$. A routing entry for the temporary location of μ will be stored in the *Mtable* of every node comprising the shortest path between $IPparent(\mu)$ and $CTparent_i(\mu)$. Moreover, if μ has descendants in the IPtree, in other words, $k(\mu) = |IPchildren(\mu)| > 0$, then each node $c \in IPchildren(\mu)$ will send `keepRoute` beacons to their respective $CTparent_i(c)$, and a (temporary) routing entry will be stored in the Mtable of every node comprising the shortest path between $CTparent_i(c)$ and $IPparent(\mu)$. Therefore, the total memory footprint to manage the mobility of a node μ is:

$$\begin{aligned}
\mathcal{M}(\mu) &= d(CTparent_i(\mu), IPparent(\mu)) + 1 \\
&+ \sum_{c \in IPchildren(\mu)} (d(CTparent_i(c), IPparent(\mu)) + 1) \\
&\leq (k(\mu)+1) \times (depth(Ctree)+1) \\
&= O(depth(Ctree))
\end{aligned}$$

Theorem 1 implies that the total memory footprint to manage the mobility of m nodes is $O(m \times depth(Ctree))$. Note that μMatrix preserves locality when managing mobile routing information of a node μ, since no *Mtable* needs to be updated at nodes above the $LCA(IPparent(\mu), CTparent(\mu))$.

4.2. Control message overhead

Control messages used by μMatrix are comprised of three types: i) those used by Matrix to set up the initial IPtree and address allocation; ii) `keepRoute` beacons, defined in Section 3.2.1; and iii) `hasMoved` beacons, defined in Section 3.3.1.

For any network of size n with a spanning collection tree Ctree rooted at node r, the message and time complexity of Matrix protocol in the address allocation phase is $\mathcal{M}sg(Matrix^{IP}(Ctree)) = O(n)$ and $\mathcal{T}(Matrix^{IP}(Ctree)) = O(depth(Ctree))$, respectively, which is asymptotically optimal, as proved in [8]. Next, we bound the number of control messages of type (ii) and (iii).

Theorem 2. *Consider a network with n nodes, with a spanning collection tree Ctree rooted at node r, and m mobility events, consisting of m nodes μ_i, changing location during time intervals $\Delta_i \leq \Delta$ time-slots. Moreover, consider the* `hasMoved` *beacon parameters I_{min}, I_{max} and I_k and the* `keepRoute` *beacon interval of δ time-slots. The control message complexity of μMatrix to perform routing under mobility of m nodes is*

$$\begin{aligned} \mathcal{M}sg(\mu Matrix(Ctree)) &= O\left(\frac{m \times I_k}{I_{min}} + \frac{n}{I_{max}}\right) \\ &+ O\left(\frac{m \times \Delta}{\delta} depth(Ctree)\right). \end{aligned}$$

Proof. Firstly, we bound the number of `hasMoved (hM)` beacons, which are sent periodically by all nodes in order to detect mobility events. As described in Section 3.3.1, when there is no mobility, the periodicity of `hasMoved` beacons is $1/I_{max}$. If some node μ has moved (an ack is lost), then I_k messages are sent at intervals of I_{min} time-slots. Using the fact that the network is a tree and the number of edges is $O(n)$, this gives a total of messages

$$Msg(\mu Matrix^{\mathtt{hM}}(Ctree)) = O\left(\frac{m \times I_k}{I_{min}} + \frac{n}{I_{max}}\right).$$

Now, we bound the number of `keepRoute (kR)` beacons. As described in Section 3, mobile nodes send periodic `keepRoute` beacons at a frequency of δ to keep the *Mtable*s up-to-date. Consider a node $\mu \in Ctree$ that has moved from its home location in time-slot t_0 and returned in time-slot t_f. Consider the $IPparent(\mu)$, $CTparent_i(\mu)$ in time-slot $t_0 < t_i < t_f$, and $\Delta = t_f - t_0$. When μ is attached to a $CTparent_i(\mu)$, μ sends `keepRoute` beacons at a rate of δ for at most Δ time-slots, such beacons travel through the shortest path $|(CTparent_i(\mu), IPparent(\mu))| \leq 2 \times depth(Ctree)$. Furthermore, if μ has descendants, i.e., $k(\mu) = |IPchildren(\mu)| > 0$, then each node $c \in IPchildren(\mu)$ will also send `keepRoute` beacons at a rate of δ for at most Δ time-slots, such beacons will travel the shortest path $|(CTparent_i(c), IPparent(\mu))| \leq 2 \times depth(Ctree)$. Therefore, the total control overhead to manage the mobility of a node μ is $\leq 2 \times depth(Ctree)(k(\mu) + 1)\Delta/\delta$, which results in

$$\mathcal{M}sg(\mu Matrix^{\mathtt{kR}}(Ctree)) = O\left(\frac{m \times \Delta}{\delta} depth(Ctree)\right).$$

Finally, the total control overhead is bounded by:

$$\mathcal{M}sg(\mu Matrix) = \mathcal{M}sg(\mu Matrix^{\mathtt{hM}}) + \mathcal{M}sg(\mu Matrix^{\mathtt{kR}})$$

Once again μMatrix preserves locality when managing mobile routing state of a node μ since no messages need to be sent to nodes above the $LCA(IPparent(\mu), CTparent(\mu))$.

4.3. Routing path distortion

We analyze the route length of messages, addressed to mobile nodes. Consider the underlying collection protocol (e.g. CTP or RPL), which dynamically optimizes the (bottom-up, or upwards) links in the collection tree *Ctree*, according to some metric, such as ETX [5, 6]. We define an *optimal route* length as the distance of the shortest path between (s, d), comprised of the upwards links of the collection tree *Ctree* and the downwards links of the union of the IPv6 address tree and the reverse-collection tree, i.e., $IPtree \cup RCtree$.

Theorem 3. *μMatrix presents optimal path distortion under mobility, i.e., all messages are routed along shortest paths towards mobile destination nodes.*

Proof. Consider a mobile node $\mu \in Ctree$, which has moved from its home location in time-slot t_0. Messages addressed to μ and originated by some node $\eta \in Ctree$ in time-slot $t_i > t_0$ can belong to traffic flows of three kinds: (1) bottom-up: $LCA_i(\mu, \eta) = \mu$; (2) top-down: $LCA_i(\mu, \eta) = \eta$; and (3) any-to-any: $LCA_i(\mu, \eta) \neq \mu \neq \eta$. In case (1), messages are forwarded using the underlying collection protocol, using the upwards links of the collection tree Ctree, which is optimal. In case (2), messages are forwarded using *Mtables* of η and its descendents, until reaching the mobile location of μ in some time-slot $t_f > t_0$. This path is comprised of the downwards links of $IPtree \cup RCtree$ in time-slot $t_0 < t_i \leq t_f$, which is the optimal route from η to the mobile location of μ in that time-slot. In case (3), the route between η and $LCA_i(\mu, \eta)$ falls into the case (1) and the route between $LCA_j(\mu, \eta)$ and μ falls into the case (2), for some $t_0 < t_i \leq t_j \leq t_f$, which is optimal.

5. Mobility Modelling

As previously discussed in Section 3.3, the protocol's mobility management was designed upon the assumption that devices hold a home location position, for which they eventually return after moving around in the cyber-physical space. Thus, μMatrix presents better performance under mobility patterns that present this characteristic (see Section 4). Although μMatrix can, in principle, also work without the home location assumption, it affects its memory efficiency. Moreover, other protocols, like MANET [21, 22, 23], are designed to deal with this feature accordingly.

Fortunately, the home location assumption is often present in mobility patterns ranging from human [27, 28, 29, 30] to non-human behavior [31, 32]. *Humans mobility pattern* tends to include group meeting dynamics and regularity. For example, people typically have a home, and they go to places nearby (meet friends, work, go shopping, etc.), then, eventually, they return to their homes. On the other hand, in a *non-human mobility pattern*, entities move and also can maintain an initial fixed position where they eventually come back. For instance, consider a team of autonomous robotic vacuum cleaners, assigned with the task of cleaning an office building. Usually, the robotic cleaners move randomly and, when the batteries are low, they return to their initial positions to recharge.

These characteristics fit well with the properties of μMatrix, exploiting its performance gains over other solutions for mobile environments. However, there is a lack of available real mobility traces in such domains, usually due to privacy-related or technical issues. Opportunely, researchers have developed mobility models to fill in this gap [33, 30, 32]. A mobility model simulates the real mobility behavior and allows us to generate variable traces in several dimensions: spatial, temporal, and size. We employ mobility models to evaluate μMatrix in different scenario conditions and highlight its potential to support IoT, IoMT, and Social IoT. Following, we present the mobility model used in this work, its parameters, and behavior.

5.1. Human Mobility Model

Hess et al. [30] survey available mobility models in the literature. Here, we highlight two: small world in motion (SWIM) [29] and group regularity mobility model (GRM) [27]. SWIM produces synthetic traces with similar properties of real mobility traces. It assumes that humans go to places nearby its home, where they meet others, and eventually they return to their homes. GRM presents the same features of SWIM, but it introduces the dynamics of group meetings and social community structure. GRM produces synthetic traces with human and group regularity while other models do not.

These mobility models and others with similar characteristics [30, 32], especially the home location assumption, are suitable for μMatrix. In this work, we use GRM as mobility model to generate traces based on real traces parameters. We produce three traces using GRM mobility model: GRM-Inf06, GRM-Camb., and GRM-MIT. Table 2 lists the GRM parameters for each trace[6]. The simulation parameters are self-explanatory, except by the path time which defines the time of a mobile entity takes to move from a location to another. The statistical parameters are parameters of truncated power laws with cut-off where α_* is the power law exponent and β_* the cut-off value: α_{gmt} and β_{gmt} define the group meeting times distribution parameters; α_{dur} and β_{dur} characterize the time that a group of entities will spend together; finally, α_{size} and β_{size} define which entities will be at each group meeting. The reader can find more parameter details in [27]. Following, we describe the traces produced by GRM:

- *GRM-Inf06*: this trace was produced based on the *Infocom06* [35] real trace. The original trace was produced during a scientific conference. It has 78 nodes, of which 34 were assigned to 4 groups with sizes 4, 5, 10, and 5. The original conference trace has three different levels, but we simplify to 1 since GRM does not support this feature. The trace covers 3 days.
- *GRM-Camb*: we generate this trace based on the *Cambridge* [35] real trace, which uses 54 nodes (36 mobile + 18 fixed) distributed into two groups of students in the University of Cambridge. The data set covers 11 days.
- *GRM-MIT*: the GRM's authors provided a ready to go GRM-MIT trace [27]. The synthetic trace was produced based on *Reality MIT* trace [36] where 100 smart phones were deployed to students in two university buildings. This is the most representative trace in terms of large area, and higher mobility. In the experiments (Section 6), we highlight the contrast between GRM-MIT and the other traces.

[6] We extract the parameters from the following references:[34, 29, 27, 35].

Parameters	**GRM-Inf06**	**GRM-Camb.**	**GRM-MIT**
Simulation parameters			
# of nodes	78	54	100
Duration (days)	3	11	15
Group duration (h)	12	24	720
DIM (m^2)	300	500	1000
Path time	120	120	300
Statistical parameters			
α_{gmt}	1.35	1.35	2
β_{gmt}	12	24	720
α_{dur}	1.5	1.5	2
β_{dur}	3	20	720
α_{size}	2.24	2.24	2.24
β_{size}	30	30	30

Table 2: GRM parameters

The above real traces and others can also be found in Crawdad site [37]. But, the mobility traces usually have only the contact trace (when two nodes meet each other) and the time when the contact happened. Most of the mobile network simulators use mobile traces of positions instead of contact traces. Thus, mobility models have the advantage of generating plausible locations (coordinates) of the nodes, instead of using heuristics to infer positions from contact trace [38].

5.2. Non-human Mobility Model

In the literature, there are several non-human mobility models available [39, 32, 33]. We highlight the Random Way Point (RWP) well-known mobility model to evaluate MANET routing protocols [39]. In RWP, the mobility entities move freely in a random direction, velocity, and acceleration. In this work, we propose the Cyclical Random Waypoint Mobility Model (CRWP), a mobility model based on the RWP. CRWP is useful to model scenarios where some of the entities move to different destinations, and eventually, they return to their initial positions, which is the case of objects (e.g., portable devices) that move in offices, universities, hospitals, factories, etc.

In CRWP, the entities move independently to random destinations and speeds as in RWP. When an entity arrives at the destination, it stops for a given time T_{pause}. A difference in CRWP is that after n chosen destinations, the mobile entity returns to its initial position. Besides that, only $k\%$ of mobile entities are outside of their initial position in each instant of time. CRWP has four parameters: i) *PerMobNodes:* maximum percentage of entities that are mobile in each instant of time; ii) *Stops:* number of stops that the mobile entity do before returning to its original position; iii) *Speed:* speed which the mobile entity moves; iv) T_{pause}*:* the amount of time that the entity stays in a destination position.

Parameters	Values		
Simulation time	1.5 h		
# nodes	100 in grid		
# traces	10 traces/scenario		
Dim (m^2)	400		
Node speed	constant 4 m/s		
CRWP T_{pause}	constant 300 s		
CRWP $Stops$	Uni. Distribution (1,3)		
CRWP Trace	**Low**	**Mod.**	**High**
PerMobNodes	5%	10%	15%

Table 3: CRWP parameters

Mobility Metrics (Avg)	CRWP-Low.	CRWP-Mod.	CRWP-High
# of link failures	1621	3057	4838
Link duration (s)	761.90	457.4	345
Node degree	4.12	4.36	4.44
Time for a link to fail (s)	227.6	216.1	204.5

Table 4: CRWP mobility metrics

We produce three different traces using CRWP by varying the parameter $k\%$ (percentage of mobile nodes away from home location): i) CRWP-Low; ii) CRW-Mod; iii) CRWP-High .Table 3 shows the parameters for each trace.

5.2.1. CRWP mobility scenario analysis

We use the BonnMotion [33] to implement CRWP as well as to generate and analyze mobility traces. Table 3 presents the CRWP simulation parameters. We simulated scenarios where $n = 100$ mobile entities are assumed to be in an office or building, and they can move around and return to a predefined home position. The nodes are deployed in a grid. We divided the mobility scenarios into three groups: low, moderate, and high mobility. Table 4 presents the average of some mobility metrics [33] that characterize each scenario (low, moderate, and high mobility). We highlight that the metric number of link failures is an important metric in the performance of the network protocol. Observe that high mobility scenarios present up to 20 % more topology changes than in low mobility. As expected, the average link duration decreases when *PerMobNode* increases. The metrics node degree and time for a link to fail do not vary much from each mobility scenario.

5.3. Results

Simulation parameter	Values
Number of experiments	10 runs/trace
Border Router	1 center
`keepRoute` period	$\delta = 60$ s
RevTT	$I_{max} = 60$ s, $I_{min} = 1$ s, $I_k = 3$
RPL Trickle	$I_{max} = 60$ s
Downwards table	Size = 20 entries

Table 5: Simulation parameters

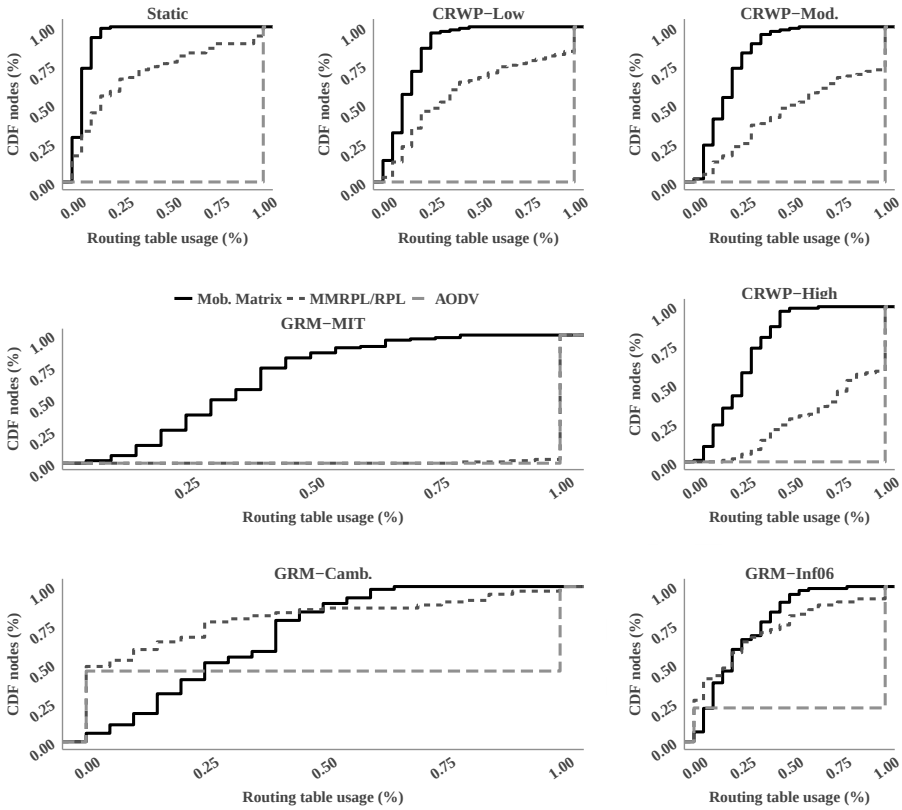


Figure 9: CDF of routing table usage. For μMatrix *Mtable + IPchildren*, for RPL only downwards routing table. The maximum table size is 20.

6. Evaluation

We implemented μMatrix in ContikiOS [40] and made the source code publicly available [7]. The experiments were executed on Cooja [41] simulator and the following three baseline protocols were used: RPL, AODV, and MMRPL [12]. The former two protocols were already available on ContikiOS, while the latter we implemented on top of the available RPL implementation.

We simulated seven different scenarios. The first scenario represents the static network, in which devices do not move. One can interpret this scenario as the traditional static IoT. The remaining scenarios present different mobility patterns by using CRWP and GRM as mobility models (see Section 5 for more details): 3 using CRWP named low, moderate, and high; 3 using GRM which we refer as Inf06, Camb., and MIT. One can interpret these scenarios as SIoT or IoMT cyber-physical mobile spaces being a step forward from standard IoT. Table 5 lists the default simulation and protocols parameters. In each plot, the bars or points represent the average, and the error bars indicate the confidence interval of 95 %. The curves are the maximum table usage for a given mobility scenario.

For static and CRWP scenarios, we executed an application on top of the network layer, in which each node sends 20 data packets to the border router at a rate of 1 packet per minute. Upon receiving a data packet, the border router confirms to the sender with an ack packet that has the size of a data packet. The application waits for 10 min for protocols initialization and stabilization before it starts sending data. The nodes start sending their data in a simulation time randomly chosen in $(10, 20]$ min. The mobility traces were configured to start after the stabilization time. Additionally, we generate 10 mobility traces for each scenario. Each trace and the static scenario were run 10 times, totaling 3010 executions.

[7] `https://bps90.github.io/mmatrix-code/`

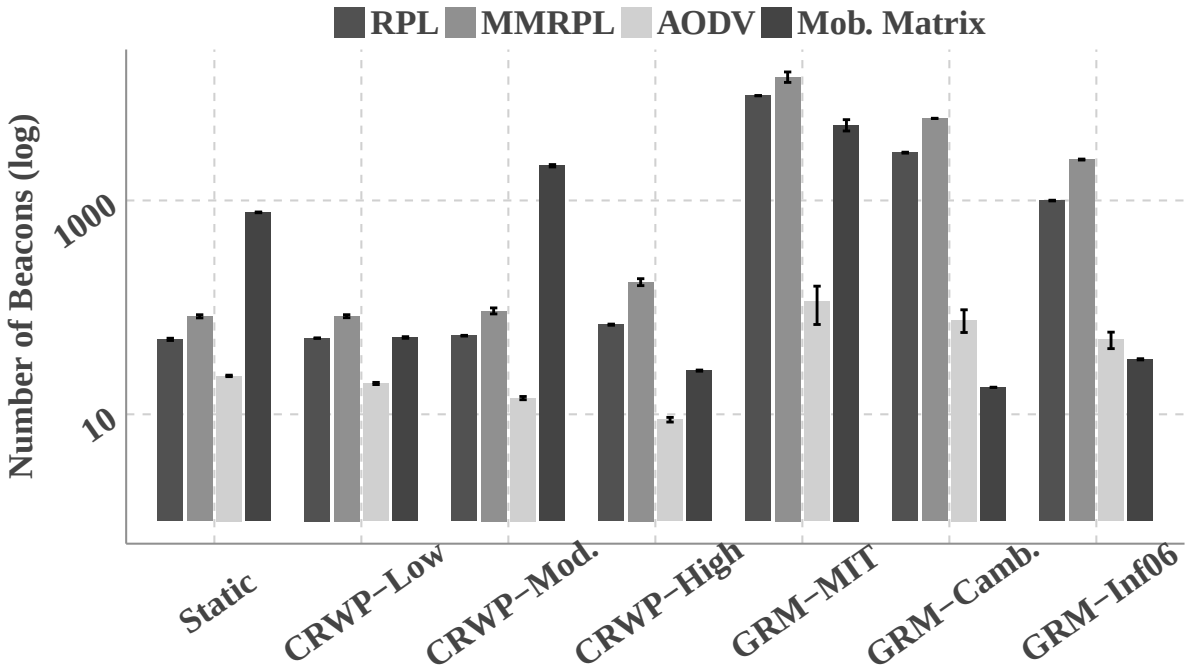


Figure 10: Number of control packets.

We ran a similar application for GRM scenarios except that the nodes, after sending their data packets, reschedule to send the data packets again in a simulation time randomly chosen in next [1, 12] hours. This process repeats indefinitely until the maximum GRM simulation time is achieved. For each trace, we run 5 times, totaling 15 executions.

In Figure 9, we show the Cumulative Distribution Functions (CDFs) of the percentage of downward routing table usage among nodes, for each scenario. In the static scenario, all μMatrix nodes use up to 25% of available downwards route entries, while in RPL and MMRPL $\geq$ 75% of nodes use $\geq$ 25% of entries. MMRPL and RPL present almost identical routing memory usage, since the only different between MMRPL RPL is the mobility detection mechanism, which does not affect memory usage. In AODV, the devices flood the network with route queries to find the packet's destination, then the devices opportunistically fill all available route entries. Therefore, AODV $\approx$ 100% of nodes use 100% of route entries. Indeed, for some RPL nodes and almost AODV nodes, 100% of table entries are used. Usually, these nodes that use more memory are near the border router, and they play a fundamental role in top-down routing. If they overflow their downward routing table, then the traffic pattern top-down suffers from poor reliability, and some nodes may be unreachable.

μMatrix also presents more efficient memory footprint than other protocols, in non-human mobility scenarios (CRWP). The difference among the protocols grows up as the nodes mobility increase. Figure 9 shows CRWP-{Low, Mod., High} memory usage. In these scenarios, μMatrix uses up to 65% of the downward routing table, while > 15% of RPL devices present full table usage as well as almost all devices running AODV.

For human mobility scenarios (GRM), we highlight two of them: GRM-MIT and GRM-Camb.. The first one, it presents higher mobility, larger area, number of nodes and duration than other scenarios. On the other hand, GRM-Camb. presents fewer nodes and mobility than MIT. Figure 9 also shows the protocols under GRM mobility pattern. In GRM-MIT, μMatrix presents lower downward routing table than RPL and AODV. For RPL and AODV, almost all route entries are used causing poor reliability in top-down and any-to-any routing. Therefore, μMatrix is more efficient

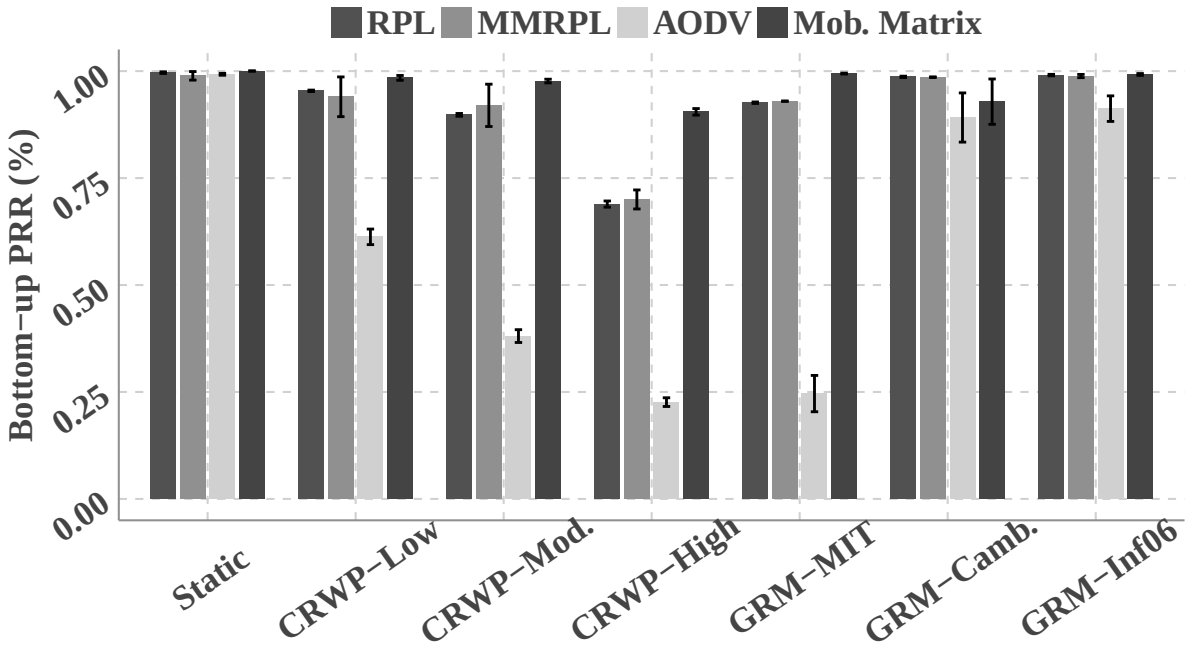


Figure 11: Bottom-up routing success rate.

in the most dynamic scenario. In the GRM-Camb. scenario, RPL and AODV seem to presented better results, but GRM-Camb. has only 54 nodes and the routing table size is only 20, and yet some nodes consume all routing memory available. In GRM-Camb., μMatrix uses $\leq$ 65% of downwards route entries.

Another relevant analysis is the control messages (or beacons) overhead of each routing protocol. μMatrix sends beacons to react to topology changes quickly by using the RevTT algorithm and the underlying collection protocol beaconing scheme. The protocol AODV sends route queries to find routes between the sender and receiver. RPL sends controls beacons to build and maintain its routing structures. μMatrix allows tuning the RevTT fire rate to reduce the sending beacons, but note that the reverse trickle adjustment faces a trade-off between quick mobility discovery and control overhead. In Table 5, we set I_{max} of RPL and μMatrix evenly and close to data packet rate, which gives to the protocols the fair opportunity to identify topology changes and react to them.

Figure 10 shows the amount of control traffic overhead of the protocols (the total number of beacons sent during the entire simulation). AODV is a reactive routing protocol (create routes on demand); therefore it sends fewer control packets than μMatrix, MMRPL, and RPL, which are pro-active. However, AODV presents higher losses than others evaluated protocols as we show ahead. MMRPL, μMatrix, and RPL present close control overhead being μMatrix slightly more economical. The difference between RPL and μMatrix does not exceed 8.6%. The different of quantities in GRM traces to others is only due to the simulation time.

Figure 11 shows the Packet Reception Rate (PRR) in bottom-up data traffic. In all scenarios, μMatrix presents higher or equal PRR than RPL, MMRPL or AODV. Upon node mobility, μMatrix realizes that a topological change happened by using RevTT, it quickly triggers the underlying route discovery, and as a consequence, bottom-up routes are rapidly rebuilt, and the reliability increases. MMRPL and RPL also present high reliability on bottom-up data traffic overall evaluated scenarios but being slightly less reliable than μMatrix. AODV, in the static scenario, presents $\approx$ 100% PRR, however, in non-human mobile scenarios, its reliability decreases as the mobility increases. In human mobility, for example, GRM-MIT (higher mobile scenario), AODV also presents poor reliability in bottom-up routing.

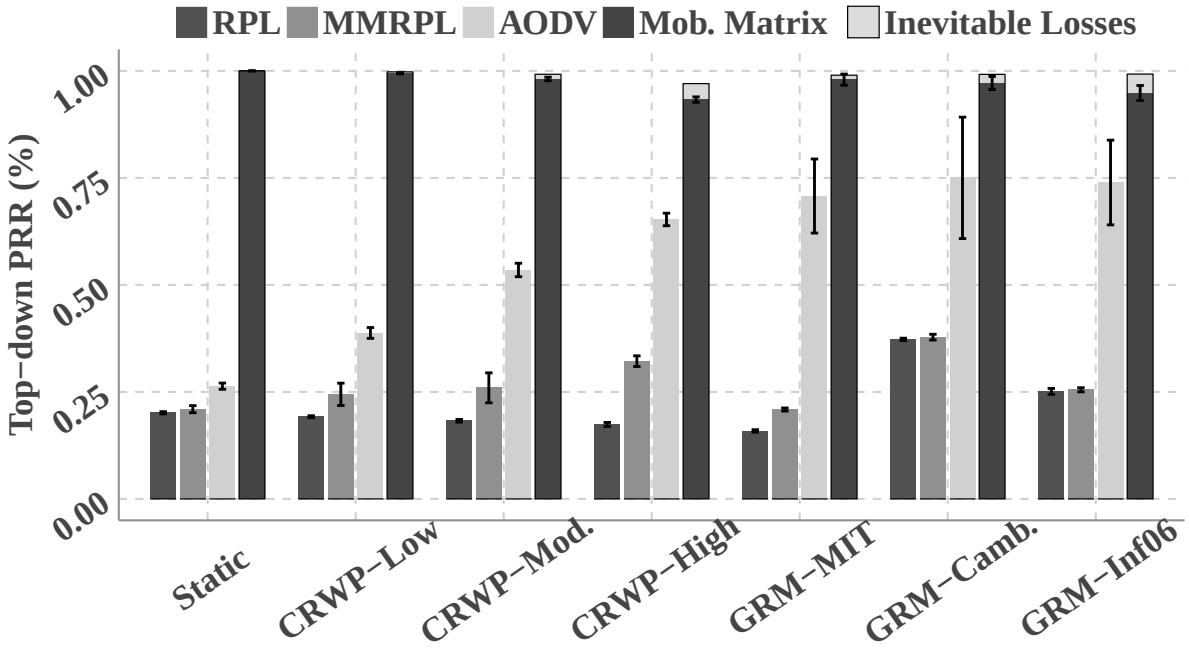


Figure 12: Top-down routing success rate. The transparent bar represents inevitable losses.

Figure 12 shows the PRR for top-down data traffic. In the plot, "inevitable losses" are represented with a transparent bar, and they refer to the number of messages that were lost due to the node being in transit from one location to another. Meanwhile, messages were routed before the route updated mechanism (see Section 3), in which case, there was no valid path to the destination, and the packet loss is inevitable.

It is possible to see that in all scenarios μMatrix presents higher PRR than other protocols in top-down data traffic routing. Under no mobility, μMatrix presented 99.9% of success rate, while RPL and MMRPL presented $< 21\%$, and AODV presented $\approx 26\%$. In non-human mobility scenarios, μMatrix PRR decreases slowly when more mobility is allowed. In the harshest mobility scenario, CRWP-High, μMatrix shows PRR of 95% while AODV has 68% and RPL has 17%. RPL, MMRPL, and AODV presented top-down PRR 19%, 28%, and 68% rescpectivelly. μMatrix, in GRM scenarios, presented at least 97% of top-down PRR, and RPL exhibited the lowest PRRs ranging from 16% to 35% followed closely by MMRPL. AODV presented PRR up to 75%, but it delivery rate with acknowledgment is low as we show ahead.

RPL, MMRPL and AODV suffer from poor reliability because of the lack of memory (see Figure 9) to store top-down routes, while μMatrix is more efficient in the memory usage, as we have shown in complexity analysis in Section 4.

Figure 13 shows the trade-off between control message overhead and successful delivery rate. Figure 13 is composed of four graphics. The two top graphics are for bottom-up traffic, and the two lower graphics are for top-down traffic. The two left graphics are for CRWP mobility model, and the two right graphics are for GRM mobility model. In each graphic, it is desirable a high delivery rate with a low number of beacons (upper-left region). However, to identify mobility passively and quickly, usually, it requires more control messages. Therefore, protocols that balance this trade-off are fundamental in the IoT, IoMT or SIoT context, especially when the devices have energy constraints.

Note that in all scenarios and traffic patterns, μMatrix presents higher delivery rate and a balanced number of

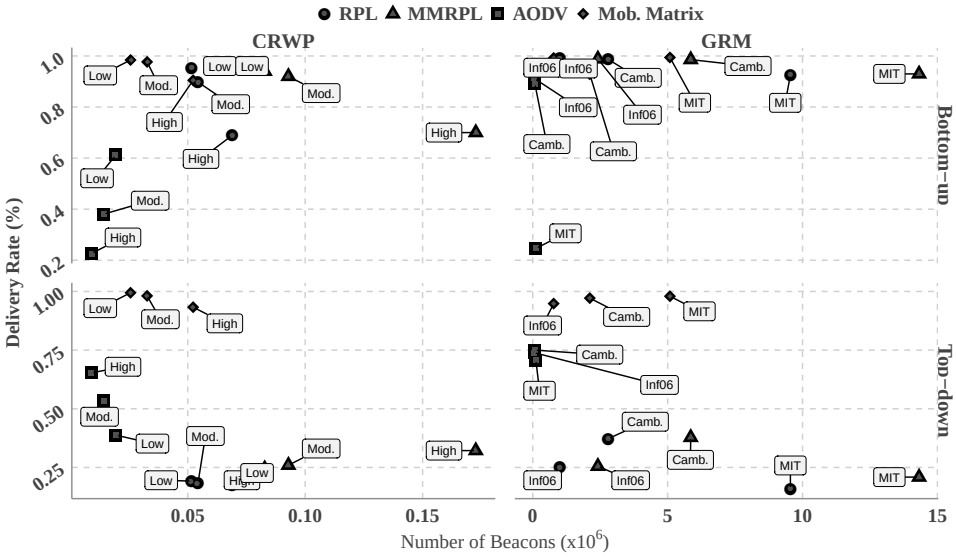


Figure 13: The trade-off between control message overhead and successful delivery rate.

beacons. AODV is the most economical concerning message overhead, but it suffers from reliability, especially in high mobility scenarios and both data traffic pattern. RPL shows the lower delivery rate and higher control message overhead. Also, the MMRPL and RPL beacons go from the node to the border router to recreate routes downwards. μMatrix reduces this costs by preserving route updates locality as discussed in Section 3 and analyzed in Section 4. In general, MMRPL presented better delivery rate than RPL at the cost of more beacons.

Figure 14 depicts the trade-off between delivery with acknowledgment rate and round trip time (RTT). Note that only round-trip messages were considered. We plot a graphic for each of the seven mobility scenarios. RPL, MMRPL, and AODV suffer from losses in the top-down traffic (as explained in Figure 12). μMatrix presents the higher delivery with acknowledgment rate. The mean RTT is similar between the four protocols except for CRWP-High scenario. In the CRWP-High scenario, μMatrix presents higher RTT because, when there is no valid route between the sender and the receiver, CTP [5] (underlying routing protocol employed) keeps some messages in a buffer for a while, then the forwarding engine (see Section 3) eventually sends the messages. Therefore, in high mobile scenarios where the topology constantly changes, some messages will be delivered with some delay; thus μMatrix also presents higher delivery with acknowledgment rate.

7. Conclusions

In this work, we have designed, analyzed and evaluated the Mobile Matrix, a mobile routing protocol with a hierarchical addressing scheme for resource-constrained devices largely employed in IoT, IoMT, and Social IoT. In the new IoT context, the "things" are able to move and do social ties; thus μMatrix represents a step towards this new mobile cyber-physical environment by allowing the devices to move around while providing device mobility transparency to upper layers in the network stack. The protocol has low memory footprint, adjustable control message overhead, optimal routing path distortion, and provides any-to-any communication. We provide a formal analysis of μMatrix memory footprint, control message overhead, and the routing path distortion. We also introduced the CRWP, a non-human mobility model suited for scenarios with mobile devices that have cyclical movement patterns.

We evaluated the routing protocols under human and non-human mobility patterns. Our μMatrix implementation offers $\geq 95\%$ of top-down PRR in highly dynamic and mobile scenarios, while other protocols $\leq 75\%$. This difference is a consequence of the downwards routing table usage, in which the devices running μMatrix protocol use up to 65% of routing entries available while for RPL, MMRPL and AODV several devices presented full routing table implying

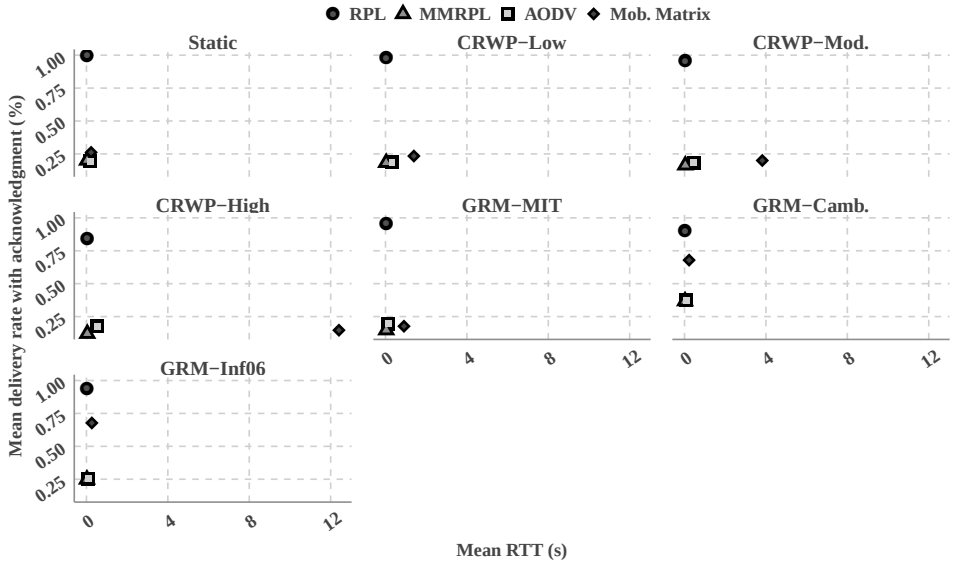


Figure 14: The trade-off between delivery rate with acknowledgment and round trip time (RTT).

in poor top-down and any-to-any reliability. It was also shown that existing routing protocols have poor delivery with acknowledgment rate.

We have shown that existing routing protocols do not meet several IoMT and SIoT requirements such as mobility management, memory, and energy efficiency in routing. Thus, efforts in that direction, e.g., the μMatrix, enable several opportunities for future research in IoMT and SIoT network stack support. For instance, it would be interesting to evaluate: i) how μMatrix and others ready-to-go protocols perform under IoMT or SIoT applications in a large-scale network with thousands of mobile devices; ii) how to passively detect mobility, the efficacy of this technique may lead to better energy efficiency of routing protocols for IoMT and SIoT. It is also worth mentioning the possibility of extending μMatrix to allow devices to move between different network domains.

Acknowledgements

We thank the research agencies CAPES, CNPq and FAPEMIG.

References

[1] B. Afzal, M. Umair, G. A. Shah, E. Ahmed, Enabling iot platforms for social iot applications: Vision, feature mapping, and challenges, Future Generation Computer Systems.

[2] L. Atzori, A. Iera, G. Morabito, M. Nitti, The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization, Computer networks 56 (16) (2012) 3594–3608.

[3] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications, IEEE Internet of Things Journal 4 (5) (2017) 1125–1142.

[4] K. Nahrstedt, H. Li, P. Nguyen, S. Chang, L. Vu, Internet of mobile things: Mobility-driven challenges, designs and implementations, in: Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on, IEEE, 2016, pp. 25–36.

[5] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, P. Levis, Ctp: An efficient, robust, and reliable collection tree protocol for wireless sensor networks, ACM TOSN 10 (1) (2013) 16.

[6] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, R. Alexander, RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (2012).

[7] O. Iova, P. Picco, T. Istomin, C. Kiraly, RPL: The Routing Standard for the Internet of Things... Or Is It?, IEEE Communications Magazine 54 (2016) 16–22.

[8] B. S. Peres, O. A. d. O. Souza, B. P. Santos, E. R. A. Junior, O. Goussevskaia, M. A. M. Vieira, L. F. M. Vieira, A. A. F. Loureiro, Matrix: Multihop Address Allocation and Dynamic Any-to-Any Routing for 6LoWPAN, in: ACM MSWiM, 2016, pp. 302–309.

[9] M. Weiser, The computer for the 21st century., Mobile Computing and Communications Review 3 (3) (1999) 3–11.

[10] A. Oliveira, T. Vazão, Low-power and lossy networks under mobility: A survey, Computer Networks 107 (2016) 339–352.

[11] O. Gaddour, A. Koubâa, R. Rangarajan, O. Cheikhrouhou, E. Tovar, M. Abid, Co-RPL: RPL routing for mobile low power wireless sensor networks using Corona mechanism, in: Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on, IEEE, 2014, pp. 200–209.

[12] C. Cobarzan, J. Montavont, T. Noel, Analysis and performance evaluation of RPL under mobility, in: IEEE ISCC, 2014, pp. 1–6.

[13] K. C. Lee, R. Sudhaakar, L. Dai, S. Addepalli, M. Gerla, RPL under mobility, in: IEEE CCNC, IEEE, 2012, pp. 300–304.

[14] I. El Korbi, M. B. Brahim, C. Adjih, L. A. Saidane, Mobility enhanced RPL for wireless sensor networks, in: IEEE ICUFN, 2012, pp. 1–8.

[15] H. Fotouhi, D. Moreira, M. Alves, mRPL: Boosting mobility in the Internet of Things (2015).

[16] K.-S. Hong, L. Choi, DAG-based multipath routing for mobile sensor networks, in: IEEE ICT, 2011, pp. 261–266.

[17] B. P. Santos, M. A. M. Vieira, L. F. M. Vieira, eXtend collection tree protocol, in: IEEE WCNC, 2015, pp. 1512–1517.

[18] S. Dawson-Haggerty, A. Tavakoli, D. Culler, Hydro: A hybrid routing protocol for low-power and lossy networks, in: IEEE SmartGridComm, 2010, pp. 268–273.

[19] C. Perkins, D. Johnson, J. Arkko, Mobility support in IPv6 (2011).

[20] L. Bellier, K. E. Malki, C. Castelluccia, H. Soliman, Hierarchical Mobile IPv6 (HMIPv6) Mobility Management, RFC 5380 (2008).

[21] C. Perkins, E. Belding-Royer, S. Das, Ad hoc on-demand distance vector (AODV), RFC 3561 (2003).

[22] T. Clausen, P. Jacquet, Optimized link state routing protocol (olsr), RFC 3626 (2003).

[23] K. Kim, S. D. Park, G. Montenegro, S. Yoo, N. Kushalnagar, 6lowpan ad hoc on-demand distance vector routing (load), Network WG Internet Draft 19.

[24] K. Kim, G. Montenegro, S. Park, I. Chakeres, C. Perkins, Dynamic manet on-demand for 6lowpan (dymo-low) routing, Internet Engineering Task Force.

[25] B. P. Santos, O. Goussevskaia, L. F. Vieira, M. A. Vieira, A. A. Loureiro, Mobile Matrix: A Multihop Address Allocation and Any-to-Any Routing in Mobile 6LoWPAN, in: Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Q2SWinet '17, 2017, pp. 65–72.

[26] P. Levis, N. Patel, D. Culler, S. Shenker, Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks, in: USENIX NSDI, 2004, pp. 2–2.

[27] I. O. Nunes, C. Celes, M. D. Silva, P. O. Vaz de Melo, A. A. Loureiro, GRM: Group Regularity Mobility Model, in: Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '17, ACM, New York, NY, USA, 2017, pp. 85–89.

[28] X. Hong, M. Gerla, G. Pei, C.-C. Chiang, A group mobility model for ad hoc wireless networks, in: Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, ACM, 1999, pp. 53–60.

[29] A. Mei, J. Stefa, SWIM: A Simple Model to Generate Small Mobile Worlds, in: IEEE INFOCOM 2009, 2009, pp. 2106–2113.

[30] A. Hess, K. A. Hummel, W. N. Gansterer, G. Haring, Data-driven human mobility modeling: A survey and engineering guidance for mobile networking, ACM Computing Surveys (CSUR) 48 (3) (2016) 38.

[31] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research, Wireless communications and mobile computing 2 (5) (2002) 483–502.

[32] V. F. Mota, F. D. Cunha, D. F. Macedo, J. M. Nogueira, A. A. Loureiro, Protocols, mobility models and tools in opportunistic networks: A survey, Computer Communications 48 (2014) 5–19.

[33] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, M. Schwamborn, BonnMotion: a mobility scenario generation and analysis tool, in: EAI ICST, 2010, p. 51.

[34] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, J. Scott, Pocket switched networks: Real-world mobility and its consequences for opportunistic forwarding, Tech. rep., University of Cambridge, Computer Laboratory (2005).

[35] P. Hui, J. Crowcroft, E. Yoneki, Bubble rap: Social-based forwarding in delay-tolerant networks, IEEE Transactions on Mobile Computing 10 (11) (2011) 1576–1589.

[36] N. Eagle, A. S. Pentland, Reality mining: sensing complex social systems, Personal and ubiquitous computing 10 (4) (2006) 255–268.

[37] D. Kotz, T. Henderson, Crawdad: A community resource for archiving wireless data at dartmouth, IEEE Pervasive Computing 4 (4) (2005) 12–14.

[38] J. Whitbeck, M. D. de Amorim, V. Conan, Plausible Mobility: Inferring Movement from Contacts, in: Proceedings of the Second International Workshop on Mobile Opportunistic Networking, MobiOpp '10, ACM, New York, NY, USA, 2010, pp. 110–117.

[39] F. Bai, A. Helmy, A survey of mobility models, Wireless Adhoc Networks.

[40] A. Dunkels, B. Gronvall, T. Voigt, Contiki-a lightweight and flexible operating system for tiny networked sensors, in: IEEE LCN, 2004, pp. 455–462.

[41] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, P. J. Marrón, COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks, in: Simutools'09, 2009, pp. 1–27.