Mobile Matrix: A Multihop Address Allocation and Any-To-Any Routing in Mobile 6LoWPAN

Bruno P. Santos, Olga Goussevskaia, Luiz F. M. Vieira, Marcos A. M. Vieira, Antonio A.F. Loureiro
Computer Science Department, Universidade Federal de Minas Gerais – Brazil
{bruno.ps,olga,lfvieira,mmvieira,loureiro}@dcc.ufmg.br

ABSTRACT

In this work, we present Mobile Matrix, a routing protocol for 6LoWPAN that uses hierarchical IPv6 address allocation to perform any-to-any routing and mobility management without changing a node's IPv6 address. In this way, device mobility is transparent to the application level. The protocol has low memory footprint, adjustable control message overhead and achieves optimal routing path distortion. Moreover, it does not rely on any particular hardware for mobility detection, such as an accelerometer. Instead, it provides a passive mechanism to detect that a device has moved. We present analytic proofs for the computational complexity and efficiency of Mobile Matrix, as well as an evaluation of the protocol through simulations. Finally, we propose a new mobility model, to which we refer as cyclical random waypoint mobility model, that we use to simulate mobility scenarios, where communication is carried out in environments with limited mobility, such as 6LoWPANs deployed in office buildings, university campuses, concert halls or sports stadiums. Results show that μMatrix deliveries 3x times more packets than RPL for top-down traffic over high mobility scenario.

CCS CONCEPTS

•Networks →Network protocol design; Network layer protocols;

KEYWORDS

Mobility; 6LoWPAN; IPv6; CTP; RPL; any-to-any routing;

1 INTRODUCTION

IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) is an IETF working group that defines standards for low-power devices to communicate with Internet Protocol. It can be applied even to the small devices to become part of the Internet of Things (IoT). It has defined protocols, including encapsulation and header compression mechanisms, which allow IPv6 packets to be sent and received over low-power devices. These protocols, such as CTP [12] and RPL [24], typically build an acyclic network topology to collect data, such as a tree or a directed acyclic graph. However, they do not handle any-to-any communication or mobility [13].

Mobility is a major factor present in everyday life. It makes life easier and turns applications more flexible. The usage of many devices for IoT can benefit from it, as is the case of today adoption of smartphones and tablets. By extending IoT protocols to handle mobility, IoT becomes even more ubiquitous.

Matrix (Multihop Address allocation and dynamic any-To-any Routing for 6LoWPAN) [20] is a platform-independent routing protocol for dynamic network topologies and fault-tolerant any-to-any data lows in 6LoWPAN. Matrix uses hierarchical IPv6 address allocation and preserves bi-directional routing.

We present Mobile Matrix (μMatrix), a solution for handling mobility in 6LoWPAN built upon the Matrix protocol. It provides the benefits from Matrix, including any-to-any routing, memory efficiency, reliability, communication efficiency, hardware independence while dealing with mobility efficiently. It enables Matrix to be used in scenarios and applications where mobility is present.

μMatrix handles mobility at the network layer, so the IPv6 address of each node is assigned once and kept unchanged despite mobility. In this way, routing and mobility management is transparent to the application level. The proposed communication protocol has low memory footprint, being suitable for low memory devices, such as wireless sensor networks and IoT. Since there is an intrinsic trade-off between the delay to detect that a node has moved and the number of control messages, μMatrix is able to tune the frequency of control messages according to the application or the mobility pattern. Moreover, μMatrix has optimal routing path distortion, i.e., messages addressed to a mobile node, from anywhere in the network, are sent along the shortest path from the source to its current location, using its original IPv6 address.

To the extent of our knowledge, previous mobile routing protocols for 6LoWPAN have not used hierarchical IPv6 address allocation, but a flat address structure, which incurs in more memory consumption to store the bi-directional routes. On the other hand, protocols for mobile ad hoc networks, like AODV [21] and OLSR [4], have high memory footprint and control message overhead, which makes them not suitable for low power devices or 6LoWPAN.

The main contributions of this paper can be summarized as follows. 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 has low memory footprint, adjustable control message overhead and achieves optimal routing path distortion. We provide analytic proofs for the computational complexity and efficiency of μMatrix, as well as an evaluation of the protocol through simulations. An essential building block of μMatrix is the passive mobility detection mechanism that captures changes in topology without requiring additional hardware (e.g. accelerometer, GPS or compass).

Q2SWinet'17, Miami, FL, USA
© 2017 ACM. 978-1-4503-5165-2/17/11…$15.00
DOI: 10.1145/3132114.3132126

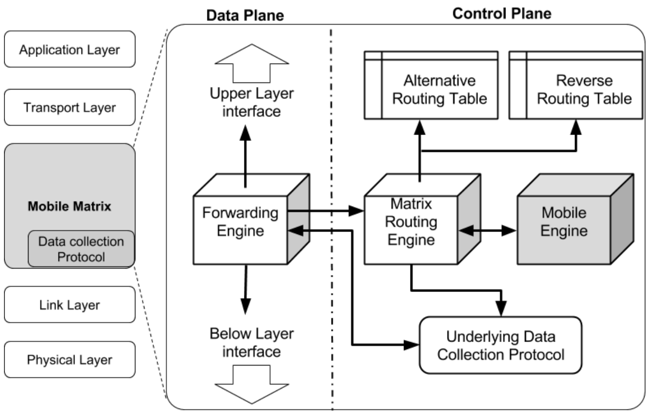


Figure 1: μMatrix protocol's architecture.

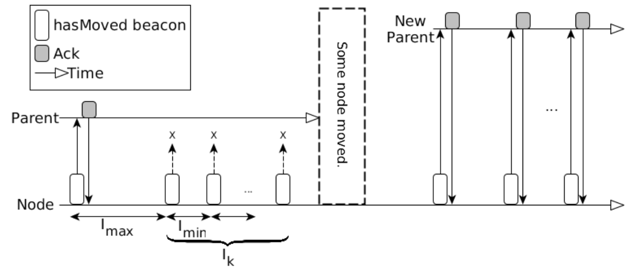


Figure 2: Reserve Trickle timer operation.

Moreover, we propose a new mobility model, to which we refer as *Cyclical Random Waypoint mobility model (CRWP)*. In CRWP, nodes are assigned to a home location and might make several moves in random directions, connecting to the 6LoWPAN at different attachment points, 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.

2 DESIGN OVERVIEW

μMatrix enables any-to-any communication for mobile and static nodes in 6LoWPANs. μMatrix manages mobile nodes without changing its IPv6 address. Also, the protocol preserves all features from previous implementation (such as memory efficiency and fault tolerance) [20]. Figure 1 presents the protocol's architecture. μMatrix lies at network layer with an underlying data collection protocol (such as CTP or RPL). μMatrix has two planes: i) **Control plane** able to split and distribute the available address space, manage route tables, and handle mobile nodes; ii) **Data plane** capable on querying route tables and forward data and control packets.

μMatrix operation consists of the following phases:

1. Collection tree initialization (Ctree): An underlying routing protocol (e.g CTP [12] or RPL [16]) creates a collection routing tree.
2. Descendants convergecast, IPv6 tree: once the collection tree is stable, μMatrix builds an address hierarchy tree (IPtree) by using MHCL algorithm [19, 20]. Initially, IPtree has the same topology as Ctree^R (top-down direction), but in runtime, they may differ.
3. Mobility management: μMatrix manages the RCtree structure, a tree that reflects the topology changes due to nodes mobility.
4. Standard routing: bottom-up routing follows the Ctree built in phase 1, while top-down the IPtree. Any-to-any routing combines both previous schemes, i.e., a packet flows bottom-up fashion until a Least Common Ancestor (LCA) between the sender and receiver and then it flows top-down until the destination.

2.1 Mobility detection

Mobility detection is a key issue to handle mobile nodes on μMatrix. If nodes by itself inform its motion (e.g. by using accelerometer or GPS) to the protocol, then we refer to as active motion, otherwise if the protocol infer the node movement, we refer to passive motion.

Trickle [17] algorithm passively detects topology changes. However, Trickle lacks in agility to detect changes in dynamic network and mobile nodes. We propose Reverse Trickle timer that operates similarly to the standard algorithm, but in reverse order.

Reverse Trickle introduce a control message and three parameters: i) `hasMoved` beacon; ii) I_{max} and I_{min} the maximum and minimum time interval to send a `hasMoved` beacon; iii) I_k the number of attempts to query a node before declaring a inconsistency. These parameters must defined by the network operator before.

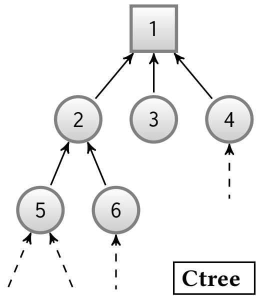
Figure 2 shows the Reverse Trickle procedure. First, a node starts sending unicast `hasMoved` beacons to its parent. I_{max} is the interval between two consecutive `hasMoved`. If the node did not receive an ack for a `hasMoved` beacon, then it sets the interval to I_{min}. After I_k unsuccessful attempts, the node knows that someone moved. 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 Reverse Trickle parameters, the network operator should consider the trade-off between delay to detect mobility and number of beacons. For a smaller delay to mobility detection, I_{max} must be tuned to small values at cost of more `hasMoved` beacons. In our experiments (Section 5) reverse trickle parameters were set according to application data rate (Table 1).

In [18], the authors argue that a common modification to support mobility is change the control message periodicity. The typical approach uses a simple periodic timer or the standardized Trickle timer. While reverse trickle waits 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 $2 \times I_{max}$.

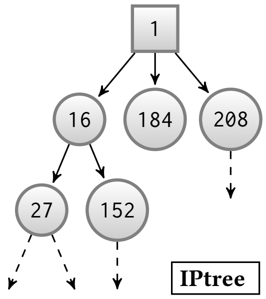
2.2 Control Plane

2.2.1 Routing data structures. μMatrix maintains three routing trees structures: i) **Ctree**: a collection tree built by the underlying collection protocol; ii) **IPtree**: an IPv6 hierarchical tree created by MATRIX initialization and kept static afterward, except when new nodes join the network; iii) **RCtree**: a tree that reflects the topology changes caused by node mobility.

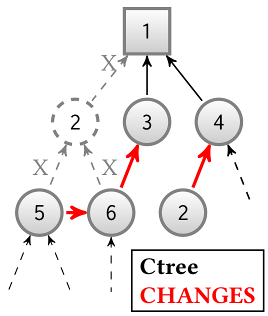
Initially, IPtree = Ctree^R and RCtree = $\emptyset$ (see Figures 3(a)(b)). Whenever a topology change occurs due to mobility in Ctree, the



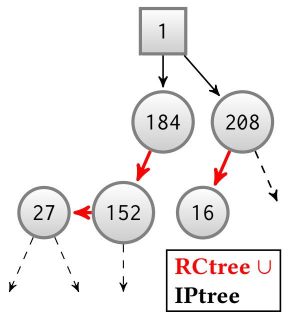
(a) Ctree structure, ID inside the nodes.



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



(c) Node 2 moves, then Ctree changes.



(d) $RCtree \cup IPtree$. Red and thicker links are in RCtree, but not in IPtree.

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

new 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 theses trees:

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

μMatrix introduce one routing packet and one parameter to control exchanging topology information and maintain the $Mtables$: i) `nodeInfo` routing frame has 7 fields: seqNum, IPv6 Node, IP range, IPv6 IPparent, CTparent, TTL, and Type. The fields are self-explanatory, except by the type field, which specifies if routing frame is a `keepRoute.{IP_ONLY or IP_AND_RANGE}` entry insertion, or `rmRoute` to remove a entry. In the following, we will use `keepRoute` for short; ii) δ parameter specifies the time between sending two consecutive.

In mobile scenarios, a node fills $Mtable$ upon receiving `keepRoute` beacons from mobile nodes. The node keeps $Mtable$ entries as long

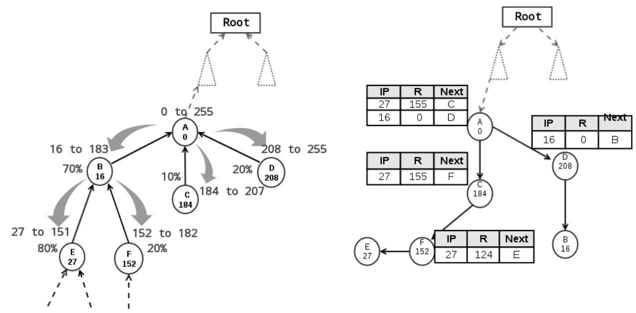


Figure 4: Simplified hierarchical address assignment with 8-bit available address space and 6.25 % of address reserve for delayed nodes. Inside the nodes its label and assigned IP, the % next the nodes express the approximate sub-tree size. Thick downwards arrows indicates the available IP range fairly distribution. In the rightmost, $Mtable$ after B moves.

as it receives `keepRoute`. Otherwise, it uses a THL mechanism to remove entries (see Sec 3.1 for memory footprint analysis). In static scenarios any node stores one-hop neighborhood information in $IPparent(\eta)$, this requires $O(k)$ entries, where k is the number of node's children. 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 for top-down routing fashion

2.2.2 IPv6 multihop host configuration. μMatrix relies on an underlying collection routing protocol to build the Ctree. Once the Ctree is stable[1], the address space available to the border router, e.g., the 64 least-significant bits of the IPv6 address (or a compressed 16-bit representation of the latter), is hierarchically partitioned among nodes in the Ctree. 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 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 most left shows the process. First, it is created the Ctree (upwards 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 root 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. Next, the node distributes the remaining range fairly between 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.

2.2.3 Mobility management. After host configuration, μMatrix starts the mobile engine allowing nodes to move around the 6LoWPAN. Mobile engine uses a finite state machine (Figure 5). Each node can be in one state depending on its previous condition and the knowledge about its neighborhood. The engine also uses Reverse Trickle to recognize mobility and transit among states. In the following, we discuss the actions taken in each of those states.

[1] A node is stable if it reaches k times the maximum maintenance beacon period of Ctree protocol without changing its parent. We use Trickle [17] as beacon scheme.

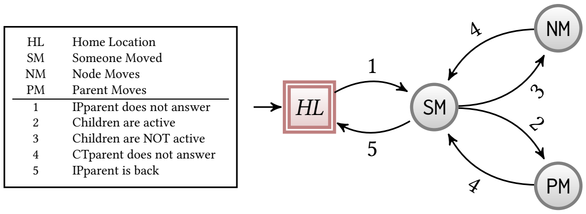


Figure 5: Mobile Engine state machine.

Each node begins at Home Location (HL) state. In HL, the nodes start the reverse trickle with its $CTparent(\eta)$ (initially, $CTparent(\eta) = Pparent(\eta)$ see Figure 6(a)). When reverse trickle identifies a mobility event, then the node transit to Someone Moved (SM) state.

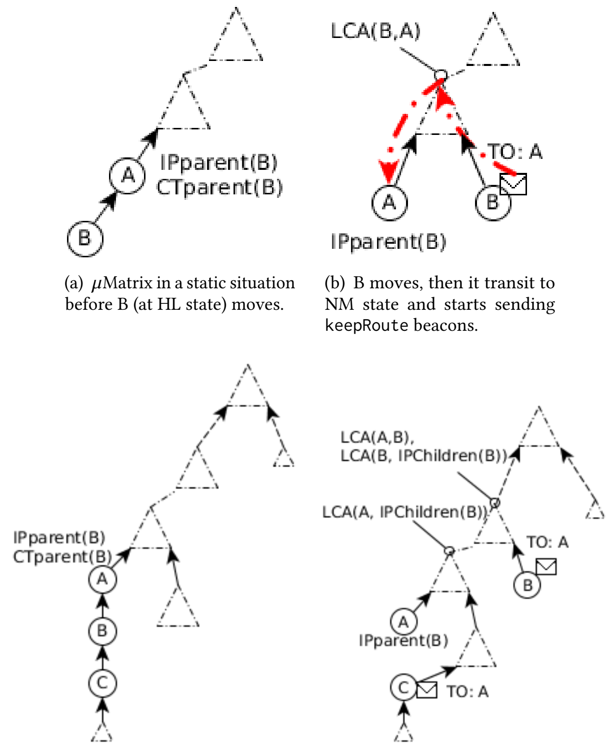
When a node is at SM state, it knows that someone moved, but it does not know if itself or its parent moved. There are at last two ways to automatically find out who moved. First, a node proactively queries its children ($IPChildren(\eta)$), if no one answer then the node moved; otherwise the parent moved. Second, a node must wait for a period (e.g. one I_{max}) 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 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 reverse trickle with new CTparent and begins sending `keepRoute.IP_ONLY` at a frequency of δ to its IPparent. Figure 6(a)(b) shows this situation. When B sends `keepRoute` beacons to A, when B moves and find a new CTparent. The beacons travel upward to the LCA(A, B) and then downwards to the node A.

When a node reaches PM, this means that its parent moved. Then, the node triggers the parent discovery mechanism. When it is attached again to Ctree, it restarts the reverse trickle and begins sending two `keepRoute` beacons (if it has children the beacon contains `IP_AND_RANGE`, otherwise `IP_ONLY`) at a frequency of δ to IPparent and its grand IPparent. Figure 6(c)(d) illustrate this situation. If B moves, then C eventually reaches PM state, and then C begins sending `keepRoute` beacons to its $grandIPparent = A$. The messages travel to LCA(A, C) and then to the ultimate destinations.

Eventually, nodes return to their home position being attached again to its IPparent in Ctree. This situation also triggers some actions. First, the node stops to sending `keepRoute` beacons and sends to its very previous CTparent = PRVparent a `rmRoute` containing its information to properly remove outdated routes nodes' *Mtable*. Also, the returned node restarts the reverse trickle with its IPparent.

Optional features are made to improve the mobile node management. Note that if a node is attached to a sequence of CTparent before returning to home location, then several states will be installed in the network. Although the *Mtable*' THL field exists to remove inconsistent entries, it is possible to send `rmRoute` beacons to each node' PRVparent to eliminate such inconsistency.



(a) μMatrix in a static situation before B (at HL state) moves.

(b) B moves, then it transit to NM state and starts sending `keepRoute` beacons.

(c) B a non leaf node before move.

(d) B moves, then C transits to PM state and start sending `keepRoute` beacons.

Figure 6: Mobile engine operation after mobility events.

Discussion: note that a sub-tree can move and nodes still hear each other. For instance, in Figure 6(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* needs updates above LCA. Figure 4 (rightmost) illustrates this situation. When node B moves, it transits to NM state, *Mtables* comprised in the path B to $A = IPparent(B) = LCA(B, IPparent(B))$ receive updates. B movement causes E and F to transit to PM state. Then, E and F find new routes and update the *Mtables* between them and its grandIPparent = A. Note that $Mtable(C)$ requires only one entry for both E and F sub-trees because their IP range are contiguous and an aggregation was made.

2.2.4 Loop avoidance. dynamic links and mobile nodes cause topology route information to become outdated, which causes routing loops [12]. μMatrix uses data path validation and adaptive beaconing to detect loops CTP and RPL [12, 16]. Besides that, if a node receives more than one unique control packet[2] in a short time, then this indicates an inconsistency in the tree, which triggers the

[2]Together the `keepRoute` fields (see Sec 2.2.1) denote a unique packet instance.

control packet suppression and the underlying protocol route update. Also, $Mtable$ and `keepRoute` beacon have respectively Time Has Lived (THL) and Time To Live (TTL) field, which it is used to remove inconsistent routes and messages from the network.

2.3 Data Plane: any-to-any routing

The Forwarding Engine (see Figure 1) is responsible for data forwarding. Any-to-any routing combines bottom-up forwarding, until the LCA between the sender and receiver, and then top-down forwarding 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 an entry $Mtable$. Third, if any $Mtable$ entries match with the target address, then the node checks if the packet destination falls within some range in $r \in IPchildren(\eta)$, if positive match, then the node forwards the packet downwards according. Finally, if all previous attempts fail, then the node sends the packet upwards using $CTparent(\eta)$.

3 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 5.

3.1 Memory footprint

As described in Section 2, 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 TTL_{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 routing of each mobile node $\mu \in CTree$.

Theorem 3.1. *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, i.e, $k(\mu) = |IPchildren(\mu)| > 0$, then each node $c \in IPchildre(\mu)$ will send a temporary (bi-directional) route request 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:

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

Theorem 3.1 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 $LCA(IPparent(\mu), CTparent(\mu))$.

3.2 Control message overhead

Control messages used by μMatrix are comprised of three types: (1) those used by Matrix to set up the initial IPtree and address allocation; (2) `hasMoved` beacons, defined in Section 2.1; and (3) `keepRoute` beacons, defined in Section 2.2.1.

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 [20]. Next we bound the number of control messages of type (2) and (3).

Theorem 3.2. *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` beacons, which are sent periodically by all nodes in order to detect mobility events. As described in Section 2.1, 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 in 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

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

Now, we bound the number of `keepRoute` beacons. As described in Section 2, mobile nodes send periodic `keepRoute` beacons at a frequency of δ to keep the $Mtables$ 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 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

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

Finally, the total control overhead is bounded by:

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

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 $LCA(IPparent(\mu), CTparent(\mu))$.

3.3 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. 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 3.3. *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. $\square$

4 CRWP MOBILITY MODEL

Here, we propose the Cyclical Random Waypoint Mobility Model (CRWP), a mobility model based on the Random Waypoint [2]. 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 people and their portable devices 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.

Table 1: Simulation parameters

Simulation parameter	Values		
Simulation time	1.5 h		
# Nodes	1 center root, 100 nodes in grid		
Mobility Model	CRWP		
Application data packets	20 pkt/node, Rate = 1 pkt/min		
Radio environment	50 m UDGM constant loss		
Area of deployment	400 m ×400 m		
Reverse Trickle	I_{max} = 60 s, I_{min} = 1 s, I_k = 3		
RPL Trickle	I_{max} = 60 s		
keepRoute beaconing period	δ = 60 s		
$Mtable$	TTL_{max} = 90 s, Size = 20 entries		
RPL downwards table	Size = 20 entries		
# mobility traces	10 traces/scenario		
Number of experiments	10 runs/trace		
Node Speed	constant 4 m/s		
T_{pause}	constant 300 s		
# node stops	Uniform Dist. in [1, 3] stops		
	Low	**Moderate**	**High**
PerMobNode	5%	10%	15%

5 SIMULATION RESULTS

We implement μMatrix as a subroutine of collection protocol available in ContikiOS [7] and the experiments were run on Cooja [9]. We compare μMatrix with ContikiOS' RPL implementation. We use the BonnMotion [1] to implement CRWP as well as to generate and analyze mobility traces. We simulated four different scenarios. The first scenario represents the static network, in which nodes do not move. The remaining represent mobility scenarios named low, moderate, and high with mobile nodes. Table 1 lists the default simulation parameters used for each scenario.

On top of the network layer, we ran an application, in which each node sends 20 data packets to the root. Upon receiving a data packet, the root 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 runs. In each plot, the bars represent the average, and the error bars the confidence interval of 95 %, and the curves are the maximum table usage for a given mobility scenario.
Mobility scenario: We simulated a scenario, where n = 100 people are assumed to be in an office and can move around and return to a predefined home position. This scenario is expected to present relatively low mobility, thus in our set up, k % of the nodes are moving at any moment in time, where $k \in \{5, 10, 15\}$. Table 2 presents some mobility metrics [1] for each scenario. We highlight that link breaks play a key role in the performance of the network

Table 2: Mobility Metrics

Mobility Metrics	Low Mob. sce.	Mod. Mob. sce.	High Mob. sce.
Avg. Link Breaks	1621	3057	4838
Avg. Link duration	761.90	457.4	345
Avg. Degree	4.12	4.36	4.44
Avg. Time to link break	227.6	216.1	204.5

protocol, note that high mobility scenario presents up to 20 % more topology changes than in low mobility. As expected, the average link duration decrease when $PerMobNode$ increases. The averages of node degrees and the time to a link break do not show much variability, they reflect the simulation parameters, where the node deployment is a grid and time for a link to break is less than T_{pause}.

5.1 Results

In Figure 7, we show the Cumulative Distribution Functions (CDFs) of the percentage of downward routing table usage among nodes for given mobility scenario. In static scenarios, all μMatrix nodes use up to 25% of available downwards route entries, while RPL < 75% of nodes use up to 25% of entries. Indeed, for some RPL nodes, 100% of table entries are used. Usually, those nodes that use more memory are near to the root, and they play a fundamental role in top-down routing. If they have a full downward routing table, then the traffic pattern top-down suffers from poor reliability, and some nodes may be unreachable. In mobility scenarios, μMatrix also presents more efficient memory footprint, and the difference grows up in high mobility scenarios, where > 50% of RPL nodes have all table entries busy, while μMatrix nodes use at most 70% of downwards available routes.

Figure 8(a) shows the amount of control traffic overhead of the protocols (the total number of beacons sent during the entire simulation). RPL sends fewer control packets than μMatrix, but the difference between them does not exceed 7.4%. μMatrix sends more beacons to react to topology changes quickly. Reverse Trickle is responsible for firing most of μMatrix beacons. μMatrix allows tuning the Reverse Trickle fire rate to reduce the sending beacons, but note that the adjustment reverse trickle faces a trade-off between quick mobility discovery and control overhead. In Table 1, 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.

Packets Reception Rate (PRR) is a metric of network reliability. It computes the number of packets received successfully over all packets sent. Figure 8(b) shows the PRR in bottom-up data traffic. In all scenarios, μMatrix presents higher PRR rate than RPL. When μMatrix realizes that a topological change happened, it quickly triggers the underlying route discovery, and as a consequence, bottom-up routes are rapidly rebuilt, and the reliability increases.

Figure 8(c) shows the PRR for top-down data traffic. We can see that, when there is no mobility, μMatrix presents 99.9% of success rate. In mobility scenarios, μMatrix PRR decreases slowly when more mobility is allowed. In the harshest mobility scenario, the PRR > 75%. RPL, on the other hand, suffer from poor reliability, delivering < 21.1% in all simulated scenarios, which occurs due to the lack of memory (see Figure 7) to store top-down routes.

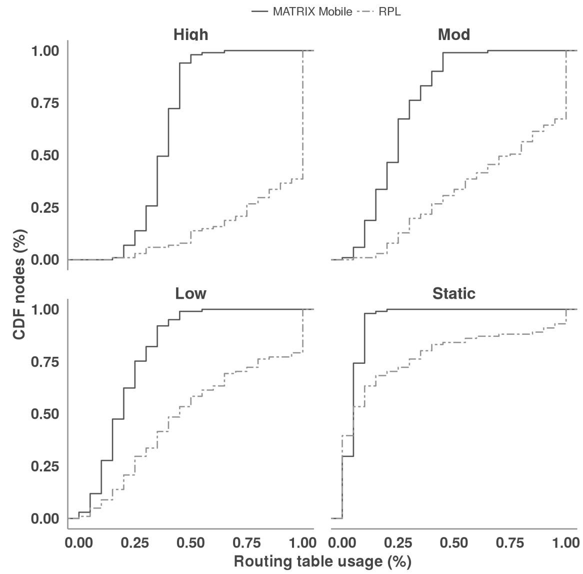


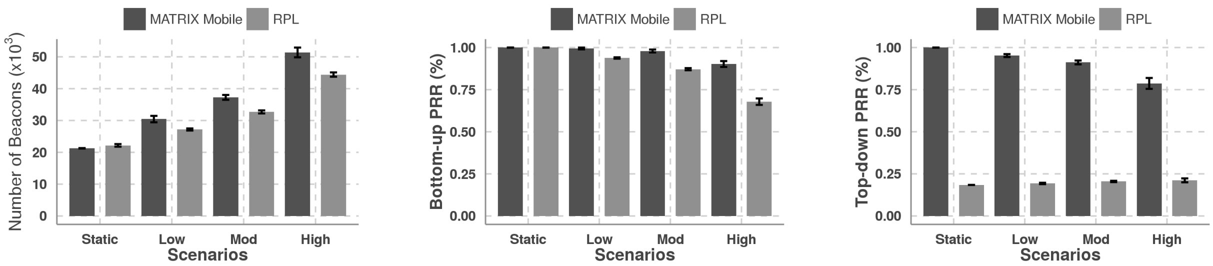
Figure 7: CDF of routing table usage. For μMatrix $Mtable$ + $IPchildren$, for RPL only downwards routing table. The maximum table size is 20.

6 RELATED WORK

In the world of tiny (IoT) several mobility-enabling routing protocols have been proposed. Firstly we highlight μMatrix's features against its original static version [20]. Then, we survey recent protocols in the context of 6LoWPAN and put them in perspective with μMatrix.

Matrix was originally proposed without support for mobility[20]. If a node moved from its home location, the hierarchical IPv6 address allocation would become invalid and compromise downward routing.Although RPL [24] is the standard protocol for 6LoWPANs, it presents limitations, for example, in mobility scenarios, scalability issues, reliability and robustness for point-to-multipoint traffic [13, 20]. Most recent mobile-enabled routing protocols are RPL extensions. They deal with mobile issues, but they do not handle RPL drawbacks. Co-RPL [11] provides mobility support to RPL but without Trickle. This turns Co-RPL more responsive but has higher overhead. MMRPL [5] modifies the RPL beacon periodicity by replacing the Trickle mechanism with a Reverse Trickle-Like. Their Reverse Trickle decays exponentially, while our approach quickly goes to the minimum after an unacknowledged beacon. MMRPL also needs some static nodes. In ME-RPL [8], static nodes have higher priority than mobile ones. ME-RPL requires some fixed nodes. The memory requirement to downward routes is still prohibitive. mRPL [10] proposes a hand-off mechanism for mobile nodes in RPL by separating nodes into mobile (MN) or serving access point (AP). They use smart-HOP algorithm on MN nodes to perform hand-off between AP.

XCTP [23] extends CTP to support bidirectional traffic. XCTP does not support IPv6 addressing and any-to-any traffic. Hydro [6] fills the gap of any-to-any traffic, but it requires static nodes with a large memory to perform the routing and support mobility nodes.



(a) Number of control packets

(b) Bottom-up routing success rate.

(c) Top-down routing success rate.

Figure 8: Simulation experiments

Table 3: Routing protocol properties

Feature	μMatrix	RPL	Co-RPL	MMRPL	ME-RPL	mRPL	DMR	Hydro	XCTP
Bottom-p	✔	✔	✔	✔	✔	✔	✔	✔	✔
Top-down	✔	✔	✔	✔	✔	✔		✔	✔
Any-to-any	✔	✔	✔	✔	✔	✔		✔	
Address Allocation	✔								
IPv6 support	✔	✔	✔	✔	✔	✔	✔		
Memory efficiency	✔								
Fault Tollerance	✔								✔
Local Repair	✔								
Topological changes	Reverse Trickle	Trickle	Periodic fixed	Reverse Trickle-like	Trickle	Trickle	Trickle	Periodic fixed	Trickle
Constraints	Nodes should return to home location			Need static nodes	Need static nodes	Need static nodes	Need static nodes	Need static nodes	

Mobile IP [22] and Hierarchical Mobile IPv6 (HMIPv6) Mobility Management [3] are standards for IPv6 networks for handling local mobility. However, they are not designed for 6LoWPANs, they do not present a mobility detection or adjustable timers. LOAD [15] and DYMO-Low [14] are 6LoWPAN protocols inspired in AODV and DYMO, but they are not suitable for mobile networks.

Table 3 summarizes properties of the related protocols.

7 CONCLUSIONS

In this work, we presented μMatrix: a memory efficient routing protocol for 6LoWPAN that performs any-to-any routing, hierarchical address allocation, and mobility management. As a building block of μMatrix, we proposed a passive mobility detection mechanism that captures topological changes without requiring additional hardware. Finally, we introduced the CRWP, a mobility model suited for scenarios with mobile nodes that have cyclical movement patterns. As future work, we plan to run experiments with physical devices and extend experimental evaluation to more mobile models, such as faulty communications scenarios.

ACKNOWLEDGEMENTS

We thank the research agencies CAPES, CNPq and FAPEMIG.

REFERENCES

[1] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla, and Matthias Schwamborn. 2010. BonnMotion: a mobility scenario generation and analysis tool. In *EAI ICST*. 51.

[2] Fan Bai and Ahmed Helmy. 2004. A survey of mobility models. *Wireless Adhoc Networks* (2004).

[3] Ludovic Bellier, Karim El Malki, Claude Castelluccia, and Hesham Soliman. 2008. Hierarchical Mobile IPv6 (HMIPv6) Mobility Management. RFC 5380. (2008).

[4] Thomas Clausen and Philippe Jacquet. 2003. Optimized link state routing protocol (OLSR). RFC 3626. (2003).

[5] Cosmin Cobarzan, Julien Montavont, and Thomas Noel. 2014. Analysis and performance evaluation of RPL under mobility. In *IEEE ISCC*. 1–6.

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

[7] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. 2004. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *IEEE LCN*. 455–462.

[8] Inès El Korbi, Mohamed Ben Brahim, Cedric Adjih, and Leila Azouz Saidane. 2012. Mobility enhanced RPL for wireless sensor networks. In *IEEE ICUFN*. 1–8.

[9] Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt, Robert Sauter, and Pedro José Marrón. 2009. COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks. In *Simutools'09*. 1–27.

[10] Hossein Fotouhi, Daniel Moreira, and Mário Alves. 2015. mRPL: Boosting mobility in the Internet of Things. (2015), 17–35 pages.

[11] Olfa Gaddour, Anis Koubâa, Raghuraman Rangarajan, Omar Cheikhrouhou, Eduardo Tovar, and Mohamed Abid. 2014. Co-RPL: RPL routing for mobile low power wireless sensor networks using Corona mechanism. In *IEEE SIES*.

[12] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, Maria Kazandjieva, David Moss, and Philip Levis. 2013. CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks. *ACM TOSN* 10, 1 (2013), 16.

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

[14] K Kim, G Montenegro, S Park, I Chakeres, and C Perkins. 2007. Dynamic MANET On-demand for 6LoWPAN (DYMO-low) Routing. *Internet Engineering Task Force* (2007).

[15] K Kim, S Daniel Park, G Montenegro, S Yoo, and N Kushalnagar. 2007. 6LoWPAN ad hoc on-demand distance vector routing (LOAD). *Network WG Internet Draft* 19 (2007).

[16] Kevin C Lee, Raghuram Sudhaakar, Lillian Dai, Sateesh Addepalli, and Mario Gerla. 2012. RPL under mobility. In *IEEE CCNC*. IEEE, 300–304.

[17] Philip Levis, Neil Patel, David Culler, and Scott Shenker. 2004. Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *USENIX NSDI*. 2–2.

[18] Afonso Oliveira and Teresa Vazão. 2016. Low-power and lossy networks under mobility: A survey. *Computer Networks* (2016).

[19] Bruna Peres and Olga Goussevskaia. 2016. MHCL: IPv6 Multihop Host Configuration for Low-Power Wireless Networks. *arXiv:1606.02674* (2016).

[20] Bruna S. Peres, Otavio A. de O. Souza, Bruno P. Santos, Edson R. Araujo Junior, Olga Goussevskaia, Marcos A. M. Vieira, Luiz F. M. Vieira, and Antonio A. F. Loureiro. 2016. Matrix: Multihop Address Allocation and Dynamic Any-to-Any Routing for 6LoWPAN. In *ACM MSWiM*. 302–309.

[21] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. 2003. Ad hoc on-demand distance vector (AODV). RFC 3561. (2003).

[22] Charles Perkins, David Johnson, and Jari Arkko. 2011. Mobility support in IPv6. (2011).

[23] Bruno P Santos, Marcos AM Vieira, and Luiz FM Vieira. 2015. eXtend collection tree protocol. In *IEEE WCNC*. 1512–1517.

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