MATRIX Mobile: A Multihop Address allocation and Any-To-Any Routing in Mobile 6LoWPAN

Bruno P. Santos
bruno.ps@dcc.ufmg.br

Olga Goussevskaia
olga@dcc.ufmg.br

Luiz F. M. Vieira
lfvieira@dcc.ufmg.br

Marcos A. M. Vieira
mmvieira@dcc.ufmg.br

Antonio A.F. Loureiro
loureiro@dcc.ufmg.br

Computer Science Department, Universidade Federal de Minas Gerais, Brazil

ABSTRACT

CCS Concepts

•Networks → Network protocol design; Network layer protocols;

Keywords

Mobility, 6LoWPAN; IPv6; CTP; RPL; any-to-any routing; fault tolerance

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. It has defined protocols, including encapsulation and header compression mechanisms, that allows IPv6 packets to be sent and received over low-power devices. These protocols, such as CTP (Collection Tree Protocol [1]) and RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks [2]), build a structure to collect data, such as a tree or bottom-up data flow. However, they do not handle any-to-any communication neither mobility.

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

MATRIX (Multihop Address allocation and dynamic any-To-any Routing for 6LoWPAN) [3] 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 bidirectional routing (upwards and down towards the sink).

MSWiM '17, November 21-25, 2017, Miami Beach, USA
© 2016 ACM. ISBN 978-1-4503-4502-6/16/11...$15.00
DOI: http://dx.doi.org/

In this paper, we present MATRIX Mobile, a solution for handling mobility in 6LoWPAN built upon the MATRIX protocol. It provides the benefits from the MATRIX protocol, including any-to-any routing, memory efficiency, reliability, communication efficiency, hardware independence while dealing with mobility in an efficient way. It enables MATRIX to be used in scenarios and application where mobility is present.

In addition to the MATRIX Mobile protocol, which includes the treatment and message exchange for mobility, it also provides a mechanism to detect passively the devices' mobility, therefore triggering the necessary mechanisms to make the communication work properly.

Previous mobile protocols do not consider hierarchical IPv6 address allocation, only flat address, which incurs in more memory consumption to store the bidirectional routes.

Our main contributions are: a passive mobility detection mechanism that captures change in topology without requiring additional hardware, (e.g. accelerometers or compass), the MATRIX Mobile protocol (a solution to handle mobility in MATRIX), its evaluation with analytic and experiments validating the protocol.

The rest of this paper is organized as follows. Section 2 presents the design of MATRIX Mobile. Section 3 shows the complexity analysis of MATRIX Mobile. Section 4 evaluates our protocol. Section 5 discusses the related work.Finally, Section 6 presents the final thoughts and future work.

2. DESIGN OVERVIEW

The objective of MATRIX Mobile is enabling any-to-any communication for static and mobile nodes into 6LoWPANs. MATRIX Mobile preserves positive aspects of MATRIX protocol, such as memory and message efficiency, as well as, adaptability to topological dynamics changes, while providing ways to manage mobile nodes. MATRIX Mobile works at the network layer together with an underlying data collection protocol, such as CTP and RPL. Figure 1 presents the protocol's architecture, which is divided into two planes: i) **Data plane** is responsible for querying the route tables and packet forwarding; ii) **Control plane** is responsible for address space partitioning and distribution, manage the route tables, and relocate mobile nodes. Following, we review important MATRIX's components and then turn our attention to the *Mobile Engine*, which is the focus of this work.

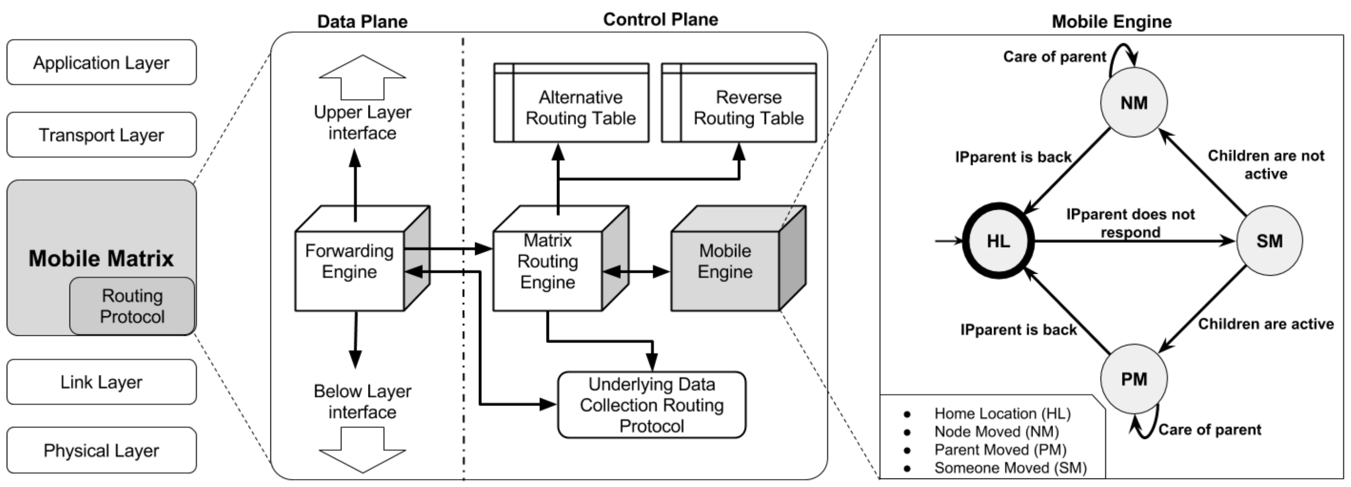

Figure 1: Mobile MATRIX protocol's architecture.

The MATRIX Mobile protocol operation consists of the following phases[1]:

1) **Collection tree initialization (Ctree):** a collection tree is built by an underlying routing protocol (e.g CTP or RPL);

2) **Descendants convergecast, IPv6 tree broadcast:** once the collection tree is stable, the address hierarchy tree (IPtree) is built using MHCL [3, 4]. The resulting address hierarchy is stored in the distributed IPtree, which initially has the same topology as Ctree, but in reverse, top-down, direction.

3) **Standard routing:** bottom-up routing follows the Ctree built in phase 1. By another hand, top-down routing follows the IPtree. Any-to-any routing is done by combining both previous schemes, i.e., a packet can be forwarded bottom-up fashion until a Least Common Ancestor (LCA) between the sender and receiver, and then forwarded top-down fashion until the ultimate destination.

4) **Mobility management:** mobility...

2.1 Control Plane

2.1.1 Distributed Tree Structures

Three distributed trees are maintained on 6LoWPAN running Mobile MATRIX protocol: i) **Ctree**: a collection tree built by the underlying collection protocol; ii) **IPtree**: an IPv6 hierarchical tree built by MATRIX initialization and extensible if new nodes join the network; iii) **RCtree**: reverse collection tree reflecting the topology changes caused by node mobility or dynamic links.

Initially, $IPtree = Ctree^R$ and $RCtree = \emptyset$ (see Figure 2(a) and 2(b)). Whenever a change occurs in one link in Ctree, the new link is added into $RCtree$ and maintained as long as the changes remain, therefore $RCtree = Ctree^R \setminus IPtree$ (see Figures 2(c) and 2(d)). Each node n_i keeps the following information in order to build and maintain theses trees:

[1]For more details please consult [3].

- $CTparent_i$: the ID of the current parent in the dynamic collection tree;
- $IPparent_i$: : the ID of the node that assigned n_i its IPv6 range initially $CTparent_i = IPparent_i$;
- $IPchildren$: the standard (top-down) routing table with IP ranges for each one-hop descendant of n_i in IPtree;
- $RChildren$: the alternative (top-down) routing table, with IP and possibly ranges in RCtree;
- $LCparent$: the IPv6 address of n_i's the very last care of parent in $Ctree$.

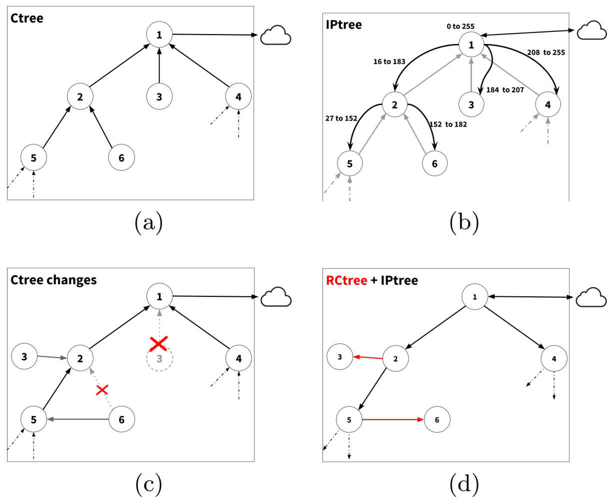

Figure 2: Trees maintained by Mobile MATRIX protocol: Ctree, IPtree, and RCtree.

Any n_i node stores only one-hop neighborhood information, so the memory requirements is $O(k)$, where k is the number of node's children. This is an optimal, considering that literature top-down routing mechanism, e.g. RPL, would need at least 1 routing entry for every child in n_i subtree.

2.1.2 *IPv6 multihop host configuration*

Mobile MATRIX lies in an underlying collection routing protocol to build the Ctree. Once the Ctree is stable[2], the address space available to the border router of the 6LoWPAN, for instance, 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 in order to distribute IP ranges in a fairly way. As result of this procedure is obtained the IPtree.

Figure 3 illustrate the process. First, the Ctree is built (upwards arrows), and then, after the Ctree stabilization, the convergecast phase occurs, which allows the nodes be aware of the size of its sub-tree (the percentage next to each node). Finally, the root starts the IP distribution by auto-setting its IP (e.g. the first IP of the available range), reserving a portion of the range for late nodes, and partitioning its range fairly between its children. Each node repeats the IP distribution process.

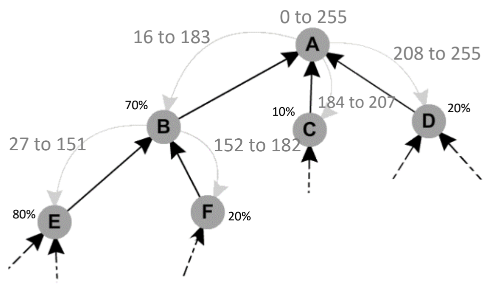

Figure 3: Simplified hierarchical address assignment with 8-bit available address space at the root and 6.25 % of address reserve for delayed connections at each node. ADD NEW IMG WITH MOBILITY TO EXPLAIN THE ROUTING

2.1.3 *Mobility Management*

ACTIVE AND PASSIVE MOBILITY DISCOVERING

Once host configuration was already done, the Mobile Engine starts working on allowing nodes to move around the 6LoWPAN. The MATRIX Mobile Engine uses a finite-state machine (Figure 3 leftmost) with 4 states: Home Location (HL), Node Moved (NM), Parent Moved (PM), and Someone Moved (SM). Each node can be in one of these states depending on its previous condition and on the present knowledge about the node neighborhood. The following, we describe which condition must be satisfied in order a node reaches each state.

1) **Home Location (HL)**: when a node receives an IP from its IPparent (Ctree stable node parent), then it starts the state machine on the HL state. This also starts a Reverse Trickle Timer (described in Section 2.3) in order to check if its IPparent stay awake. Once the that IPparent does not answer back, then the node goes to the next state.

2) **Someone Moved (SM)**: on that state, the node is aware that someone moved, but it does not know who moved (if it was itself or its current parent). This information is essential to take the right decision towards mobile node management.

   There are, at last, two ways to a node automatically find out who moved. The first one, a node in SM state actively query its children (if it has anyone), if no one answer then the node moved, otherwise the parent moved. The second one, a node in SM state waits for a period (e.g. two times the maximum period of reverse Trickle Timer) to receives queries (heartbeats) from its children and then infer who moved. As soon as the node identify who moved, it moves towards NM or PM states according to if the node or the parent moved respectively.

3) **Node Moved (NM)**: if a node M moved within the network, then it goes to NM state, where some actions are taken. First, the node cleans the temporary routing table. Second, M trigger the underlying collection protocol to discover a new route. Third, M must restart the Reverse Trickle Timer as done in HL, but now it sends heartbeats to $M.c$ (its care of parent in the Ctree). Fourth, M sends periodically a control packet to its IPparent in order to keep the IPtree updated about its location. Such control packet contains only its IP and must travel from M new position upwards to the $LCA_{(M.p,M)}$ and then downwards to $M.p$. Such control message allows router nodes to allocate temporary route entry for correctly routing packets to M.

   If M keeps moving and is attached to a different care of parent, then it must also send a control packet to $M.lc$ (its last care of parent) in order to rapidly remove inconsistent temporary routes[3]. If M comes back home, i. e., it is attached again to its IPparent, then a control message must be sent to its $M.lc$ to remove inconsistent route states.

4) **Parent Moved (PM)**: a node σ reaches PM state when it identifies that its IPparent moved. In this case, σ must trigger the underlying route discovery mechanism and sends periodically a control packet towards its grandparent (or early parent in the IPtree) in order to update the reverse route to reach itself and its sub-tree. The control packet contains σ's IP and range. Such packets must travel from σ upwards to $LCA_{(\sigma,(\sigma.p).p)}$ and then downwards to $(\sigma.p).p$.

   σ remains in PM state while its IPparent does not come back. Once that $\sigma.p$ comes back to home, then σ sends a control packet through $M.lc$, then upwards to $LCA_{((\sigma.p).p,\sigma.lc)}$ and finally downwards to $(\sigma.p).p$ in order to remove the inconsistent temporary routes.

loop...

[2]In the Ctree, a node n_i is stable if it reaches c times the maximum maintenance beacon period of Ctree protocol without changing its parent. We used a Trickle algorithm-based [5] as adaptive beacon scheme.

[3]Note this is optional given that temporary routes will be removed in a future time, but as long as temporary route remains wrong routing can occur.

2.2 Data Plane: any-to-any routing

TODO

2.3 Reverse Trickle Timer

Trickle timer [5] is an algorithm that works as follow: it starts by firing beacons with a short interval in order to quickly react to network topology dynamics during the construction of Ctree. While the network remains stable, the interval between two consecutive beacons increases, thus reducing the overhead of control packets. But, if some inconsistency is detected (e.g. loops, link failures, reconfiguration, etc), then the timer resets to its initial value and the algorithm restarts.

The standard Trickle timer is an adaptative algorithm in terms of control message overhead. However, the algorithm lacks in agility and efficiency to detect changes in a highly dynamic network with mobile nodes. To support mobile nodes, we propose a Reverse Trickle Timer that operates similarly to the standard algorithm, but in reverse order.

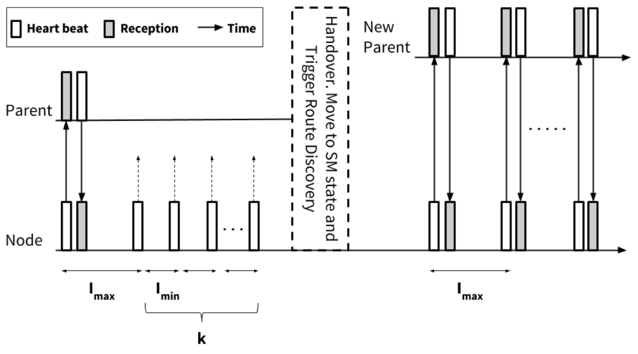

Figure 4: Reserve Trickle timer operation.

Figure 4 illustrates the reverse Trickle timer procedure. Lets denote I_{min} and I_{max} as the minimum and maximum interval between two consecutive beacons. First, the reverse Trickle starts with I_{max}. Then it goes to I_{min} if it does not receive an acknowledgment from a beacon. After k unsuccessful tries, the node knows that someone moved. Thus, the node can take actions to properly perform a handover to another parent. When the underlying collection protocol finds a new parent, then the reverse Trickle timer restarts the procedure.

In [6], the authors argue that a common alteration to support mobility is to change the control message periodicity. The typical approach uses a simple periodic timer or the standardized Trickle timer. While reverse Trickle waits for $I_{max} + k \times I_{min}$ 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. COMPLEXITY ANALYSIS

In this section, we assume a synchronous communication model with point-to-point message passing. In this model, 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 at time-slot t, it must arrive at u before time-slot $t+1$. Also, we divided intro three parts: i) **Memory footprint** ; ii) **Control message overhead**; iii) **Path distortion**;

Table 1: Terms

Term	Meaning
σ	A standby node
μ	A mobile node
$\mu.p$	M's IPparent before the movement
$\mu.c$	M's care of parent
$\mu.lc$	M's last care of parent
$d(v, u)$	Distance between nodes v and u
$LCA_{v,u}$	Local Common Ancestor of nodes v and u
Δ_v	Time that a node v spends away from home
δ	Period to send a temporary route info beacon

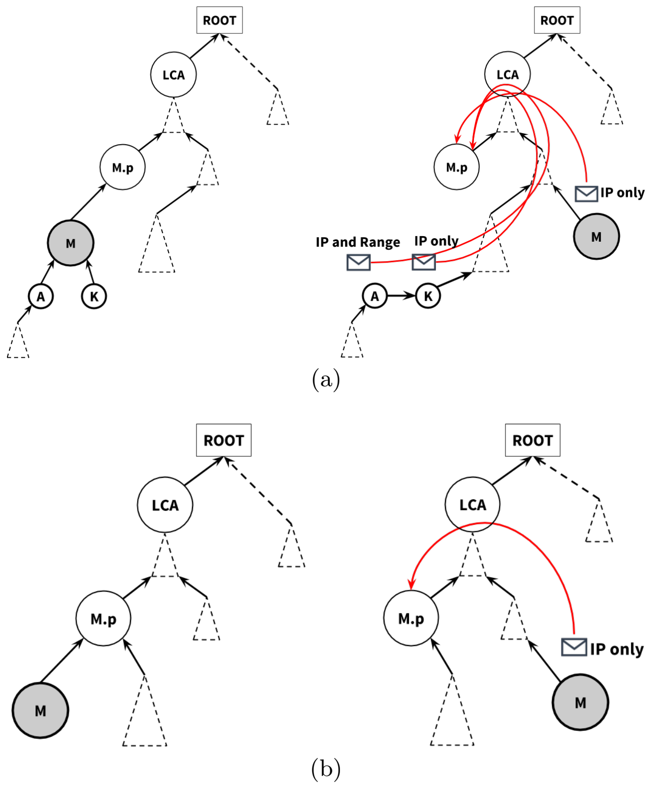

Figure 5: rename it

We performed the message and time complexity analyses of the Mobile MATRIX. Note that Mobile MATRIX requires that an underlying acyclic topology (Ctree) has been constructed by the network before its starts, i.e., every node knows who its parent in Ctree is. It is also required that IPtree has been already built. In [3] [complexity Ctree], the authors demonstrate that Ctree and IPtree construction are bounded by $O(n)$ in terms of messages, and IPtree takes $O(depth(Ctree))$ time-slots to build the hierarchical tree. Following, it is presented definitions to guide the message and time complexity analyses.

Definition 1. If a leaf node σ is on PM state, then when the underlying acyclic protocol found a new parent (care of parent), the node must send 1 control message towards its grandparent in IPtree to reconfigure the network. The message contains only the node IP.

Definition 2. If a non-leaf node σ is on PM state, then σ must act as the previous definition, but the message contains the node IP and its IP range address.

Definition 3. If a node μ is on NM state, then when the underlying acyclic protocol found a care of parent, the node must send 1 control message towards its IPparent in order to reconfigure the network. Also, if the node is moving around the deployment area and it was attached in some care of parents, then the node must send 1 control message towards its IPparent and for each last carry of parent in its journey. The message contains only the node IP.

Definition 4. If a node μ returns to its home location from NM state. Then, the node must send a 1 control message to its very last care of parent in order to remove previous the network states.

Definition 5. If a node σ on PM state is again attached to its IPparent. Then, the node must send a 1 control message through its very last care of parent in order to remove previous the network states.

Theorem 1. *Consider a network running MATRIX Mobile protocol after its stable setup, one non leaf node μ with K one-hop children in the Ctree. Moreover, μ remains away from its home location for Δ time-slots, and the periodic time for sending a control message is δ time-slots. If μ was attached to a care of parent in the Ctree, the complexity of message and time to keep the IPtree updated are $Msg(MobileMATRIX(Ctree, root)) = \Theta(|K| \times \frac{\Delta}{\delta})$ and $Time(MobileMATRIX(depth(T, root))) = O(d_1 + d_2)$, where $d_1 = d(\mu, \mu.p)$ and $d_2 = max(d(k_1, \mu.p), ..., d(k_{|K|}, \mu.p)))$.* ***This message and time complexity is asymptotically optimal.***

Proof. For simplicity, suppose a network with $n-1$ static nodes, a non leaf node μ moves to another point into the network and returns to its home location after Δ time-slots. Also, μ has K one-hop children. By **definitions** 1, 2, and 3, μ itself and every one-hop μ's[4] child must send regularly control messages at a frequency of one for each δ time-slots, while μ is away for Δ time-slots. Then, by **definitions** 4 and 5, when μ returns to home, μ and its children must send another control message in order to remove the network previous states, thus the total required control messages is bounded by $(|K|+1) \times (2 + \frac{\Delta}{\delta}) = \Theta(|K| \times \frac{\Delta}{\delta})$. Moreover, the time to perform the IPtree update for μ is bounded by $d(\mu, \mu.p) \leq d(\mu, LCA_{\mu,\mu.p}) + d(LCA_{\mu,\mu.p}, \mu.p)$ time-slots and the IPtree update for μ's children is bounded by $max(d(k_1, \mu.p), ..., d(k_{|K|}, \mu.p)),$ where $d(k_i, \mu.p) \leq d(k_i, LCA_{(k_i,\mu.p)}) + d(LCA_{(k_i,\mu.p)}, \mu.p)$ time-slots, thus the time-slots needed to update the IPtree is $O(d_1 + d_2)$.

$\square$

Theorem 2. *Consider the same setting of the Theorem 1, but now μ is a leaf node. The complexity of message is $\Theta(1)$ and time is $\Theta(d(\mu, \mu.p))$, respectively.* ***This message and time complexity is asymptotically optimal.***

Proof. By **definitions** 3, μ must regularly send 1 control message for each δ time-slots, while μ remains away from its home location for Δ time-slots. Then, by **definition** 4, when μ returns to home, μ sends 1 control message to remove the network previous state, thus the total required control messages is bounded by $2 + (1 \times \frac{\Delta}{\delta}) = \Theta(1)$. Moreover, the number of time-slots required to update the IPtree is $d(\mu, LCA_{(\mu,\mu.p)}) + d(LCA_{(\mu,\mu.p)}, \mu.p) = \Theta(d(\mu, \mu.p))$.

[4]Children before μ moves.

3.1 Discussion

talk about worse case

$\square$

4. EVALUATION

Evaluation...

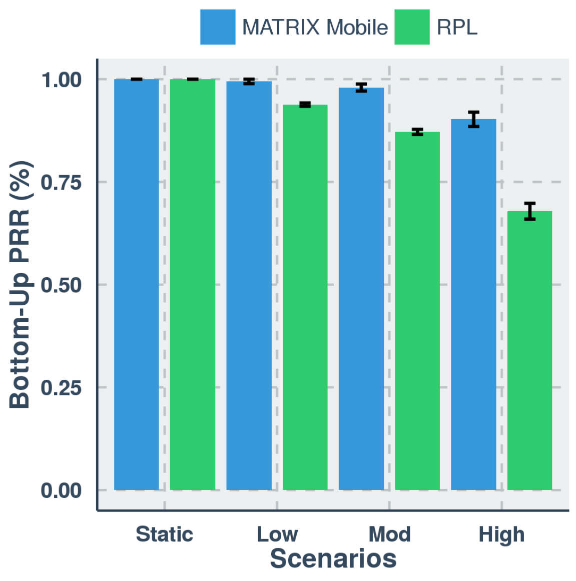

Figure 6: Bottom-Up routing success rate.

4.1 Office scenario

Table 2: Mobility Metrics

Mobility Metrics	Low Mob. sec.	Mod. Mob. sec.	High Mob. sec.
Avg. Degee	3.97	4.26	4.85
Avg. time to link break	238.7	255.5	207.9
Link Breaks	942	2011	4680
Avg. Link duration	1174.6	654.0	328.30
avg. number of link changes	1.7	1.8	2.2

The office scenario models 100 persons attending a work office, a class room, or a similar activity where entities can move around and return to a predefined position. The office scenario intends to present low mobility, thus in our set up $k\,\%$ of the nodes are moving at any moment in time, where $k \in \{5\%, 10\%, 15\%\}$ namely low, moderate and high mobility scenarios respectively. Table 2 present some mobility metrics [7] for each setting.

5. RELATED WORK

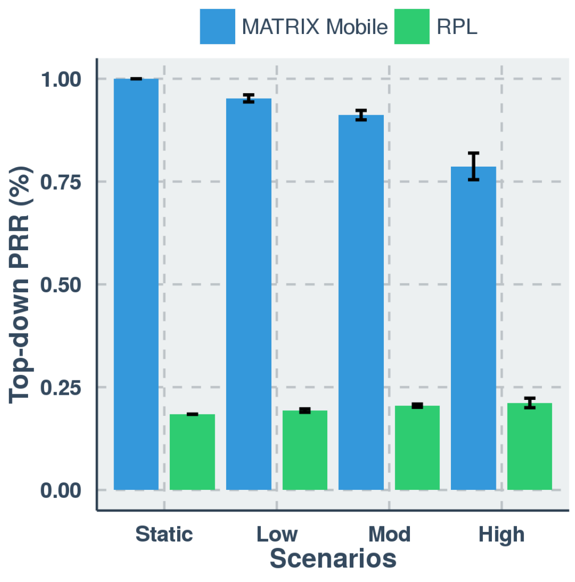

Figure 7: Top-Down routing success rate.

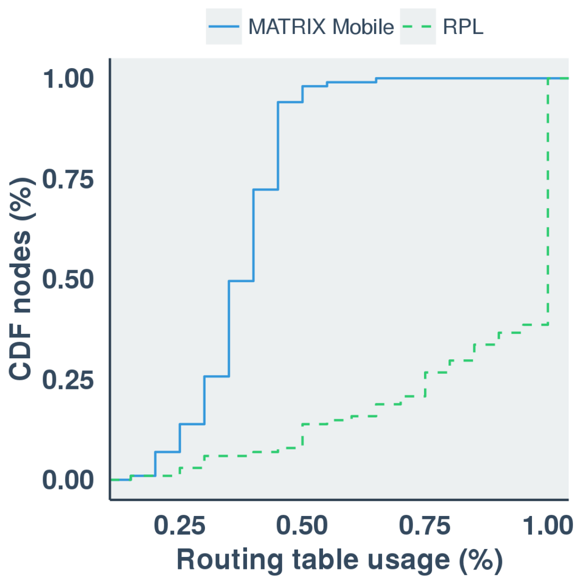

Figure 9: Downward routing table usage CDF. Maximum table size = 20.

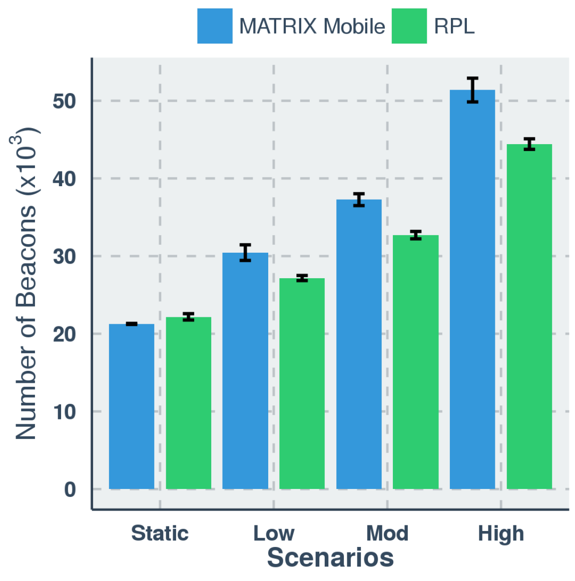

Figure 8: Number of control packets.

In the world of tiny (embedded systems, wireless sensor networks, Internet of Things) several mobile-enabled routing protocols have been proposed. Following, we first highlight MATRIX Mobile features against its original proposal. Then, we survey recent protocols in such context and put them in perspective with MATRIX Mobile.

MATRIX originally does not support node mobility. Indeed, if some node moves, the hierarchical IP distribution will be broken, and downward routing will present poor reliability. In the other hand, MATRIX Mobile enable nodes to move in the network, but they must to return to its home location in the IPtree in a future moment. MATRIX Mobile support mobility without remove any feature of the original protocol. To quickly responsiveness to mobile nodes, we implement the Reverse Trickle and control it through a finite-machine state. Also, MATRIX Mobile uses the same memory resources ($RChildren$ table) as original one to support mobility feature, i.e., we do not add more memory requirements.

RPL [2] is the standard protocol for 6LoWPANS. Although RPL is the current standard, it presents limitations, for example, in mobility scenarios, scalability issues, reliability and robustness for point-to-multipoint traffic [8, 3]. Most recent mobile-enabled routing protocol are RPL extensions. They deal with mobile issues, but they do not handle all RPL drawbacks.

Co-RPL [9] aims to provide mobility support to RPL and reliability in end-to-end communications. Co-RPL ignores Trickle mechanism in order to handle mobility, this turns Co-RPL more responsive, but a higher overheard. Co-RPL keeps RPL modes to build and repair downwards routes, which are inefficient in terms of memory. MATRIX Mobile in both by behaving as MATRIX and performing local repairs when a node moves.

In [10], the authors enhances RPL to support mobile management (MMRPL). They also proposed alterations on the beacon periodicity by replacing the Trickle mechanism by a Reverse Trickle-Like. The main difference is that Reverse Trickle-Like decays exponentially, while our approach quickly goes to the minimum after an unacknowledged beacon. MMRPL also need some static nodes, while MATRIX Mobile does not. MATRIX Mobile also is fault tolerant bypassing like failures by using local broadcast [3].

MRPL-V [11] was proposed in the vehicular context. The protocol also ignores the Trickle mechanism by replacing it by a periodic fixed timer to fire beacons. The authors do not propose more alterations in RPL. RPL is not built to support highly dynamic scenarios like vehicular ones, so even using periodic beaconing the topology changes to fast that RPL global repair can be to slow to catch the changes. Thus downwards can present poor reliability. MATRIX Mobile was not proposed to vehicular context.

In [12], the authors propose ME-RPL handling mobile nodes differently of static nodes. In the route discovery phase, when some node is choosing the preferred parent, static nodes have higher priority than mobile ones. Another alteration is in the route repair phase, when a node is detached to a parent, it sends DIS messages in dynamic intervals, which depends on the number of times that a node changes it parent. Indeed, such alterations turns ME-RPL responsive to mobile nodes rejoin the DODAG and indentify mobile nodes, but is required some fixed nodes. Also the memory requirement to downward route still prohibitive, when compared with MATRIX Mobile.

mRPL [13] propose a hand-off mechanism for mobile nodes in RPL. The authors separates the nodes into two sets: mobile nodes (MN) and serving access point nodes (AP). AP nodes are preferred parents to MN nodes. Then they use smart-HOP algorithm on MN nodes to perform hand-off between AP nodes. Again, the main difference between MATRIX Mobile and mRPL are the need to static nodes, IPv6 allocation (MATRIX configures the nodes in the 6LoWPAN), fault tolerance, and the memory efficiency.

DMR [14] protocol enhances the RPL to support data transfer in mobile nodes. They remove some features of RPL like Top-Down and Any-to-Any traffic pattern. DMR also store mode routing information to choose best parent to reliably deliver data. MATRIX Mobile support more traffic patterns than DMR, and provide memory efficiency to downwards routes.

XCTP [15] extend the Collection Tree Protocol (CTP) to supports bidirectional traffic (upwards and downwards), also it support node/link failures bypassing them. However, XCTP does not support IPv6 addressing and any-to-any traffic. Hydro fills the gap of any-to-any traffic, but Hydro requires static nodes with a large memory to perform the routing and support mobility nodes.

Table 3 summarizes the most important properties of the related protocols. We use a check mark if the protocol has the feature or empty otherwise. Such properties provide an overview of features protocol-enabled. Ten features have been considered, which are related to traffic pattern, addressing, memory, reliability, and limitations.

Acknowledgments

This work was supported in part by CAPES, CNPq and FAPEMIG.

6. CONCLUSIONS

Conclusions...

7. REFERENCES

[1] 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 Transactions on Sensor Networks (TOSN) 10 (3).

[2] 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 (Proposed Standard) (2012).

[3] 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: Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '16, ACM, New York, NY, USA, 2016, pp. 302–309.

[4] B. S. Peres, O. Goussevskaia, MHCL: ipv6 multihop host configuration for low-power wireless networks, CoRR abs/1606.02674.
URL http://arxiv.org/abs/1606.02674

[5] P. Levis, N. Patel, D. Culler, S. Shenker, Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks, in: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1, NSDI'04, 2004, pp. 2–2.

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

[7] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, M. Schwamborn, Bonnmotion: a mobility scenario generation and analysis tool, in: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, p. 51.

[8] O. Iova, G. P. Picco, T. Istomin, C. Kiraly, Rpl, the routing standard for the internet of things... or is it?, IEEE Communications Magazine 17.

[9] 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.

[10] C. Cobarzan, J. Montavont, T. Noel, Analysis and performance evaluation of rpl under mobility, in: Computers and Communication (ISCC), 2014 IEEE Symposium on, IEEE, 2014, pp. 1–6.

[11] K. C. Lee, R. Sudhaakar, L. Dai, S. Addepalli, M. Gerla, Rpl under mobility, in: Consumer Communications and Networking Conference (CCNC), 2012 IEEE, IEEE, 2012, pp. 300–304.

[12] I. El Korbi, M. B. Brahim, C. Adjih, L. A. Saidane, Mobility enhanced rpl for wireless sensor networks, in: Network of the Future (NOF), 2012 Third International Conference on the, IEEE, 2012, pp. 1–8.

[13] H. Fotouhi, D. Moreira, M. Alves, mrpl: Boosting mobility in the internet of things, Ad Hoc Networks 26 (2015) 17–35.

[14] K.-S. Hong, L. Choi, Dag-based multipath routing for mobile sensor networks, in: ICT Convergence (ICTC), 2011 International Conference on, IEEE, 2011, pp. 261–266.

[15] B. P. Santos, M. A. Vieira, L. F. Vieira, extend

Table 3: Routing protocol properties

Protocol / Feature	Bottom-Up Traffic	Top-Down Traffic	Any-to-Any Traffic	Address Allocation	IPv6 support	Memory efficiency	Fault Tollerance	Local Repair	Topological changes	Constraints
MATRIX Mobile	✓	✓	✓	✓	✓	✓	✓	✓	Reverse Trickle	Nodes must return to home location
RPL	✓	✓	✓		✓				Trickle	
Co-RPL	✓	✓	✓		✓				Periodic fixed	
MMRPL	✓	✓	✓		✓				Reverse Trickle-like	Need static nodes
MRPL-V	✓	✓	✓		✓				Periodic fixed	Not specified
ME-RPL	✓	✓	✓		✓				Trickle	Need static nodes
mRPL	✓	✓	✓		✓				Trickle	Need static nodes
DMR	✓				✓				Trickle	Need static nodes
Hydro	✓	✓	✓						Periodic fixed	Need static nodes
XCTP	✓	✓					✓		Trickle	

collection tree protocol, in: Wireless Communications and Networking Conference (WCNC), 2015 IEEE, IEEE, 2015, pp. 1512–1517.