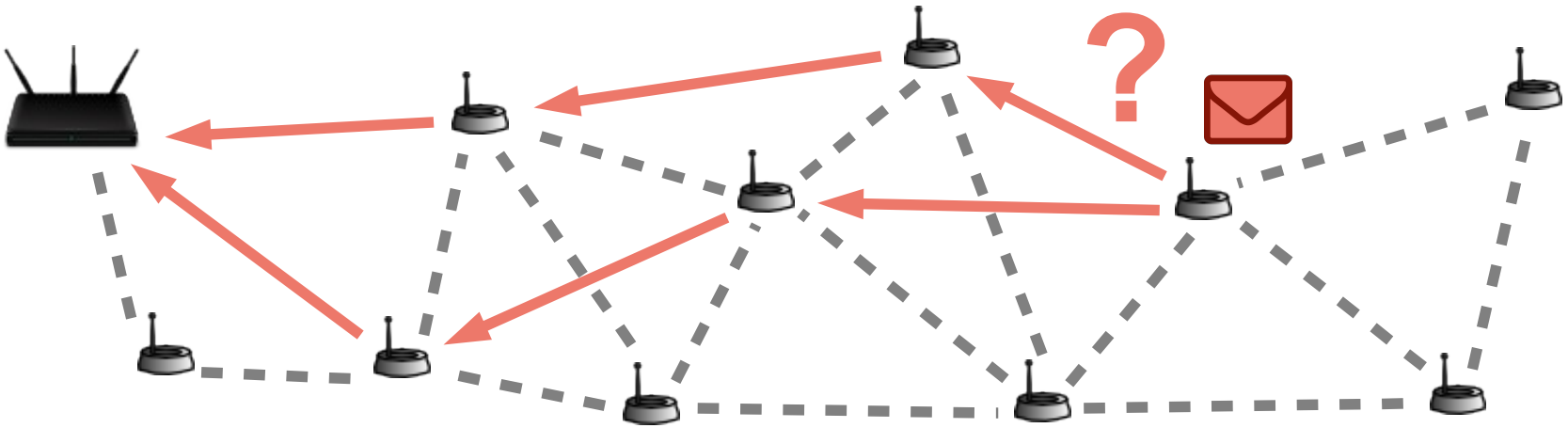
The IPv6 Routing Protocol for Low-power and Lossy Networks (RPL) under Network Partitions

Agnieszka Paszkowska, Konrad Iwanicki
University of Warsaw

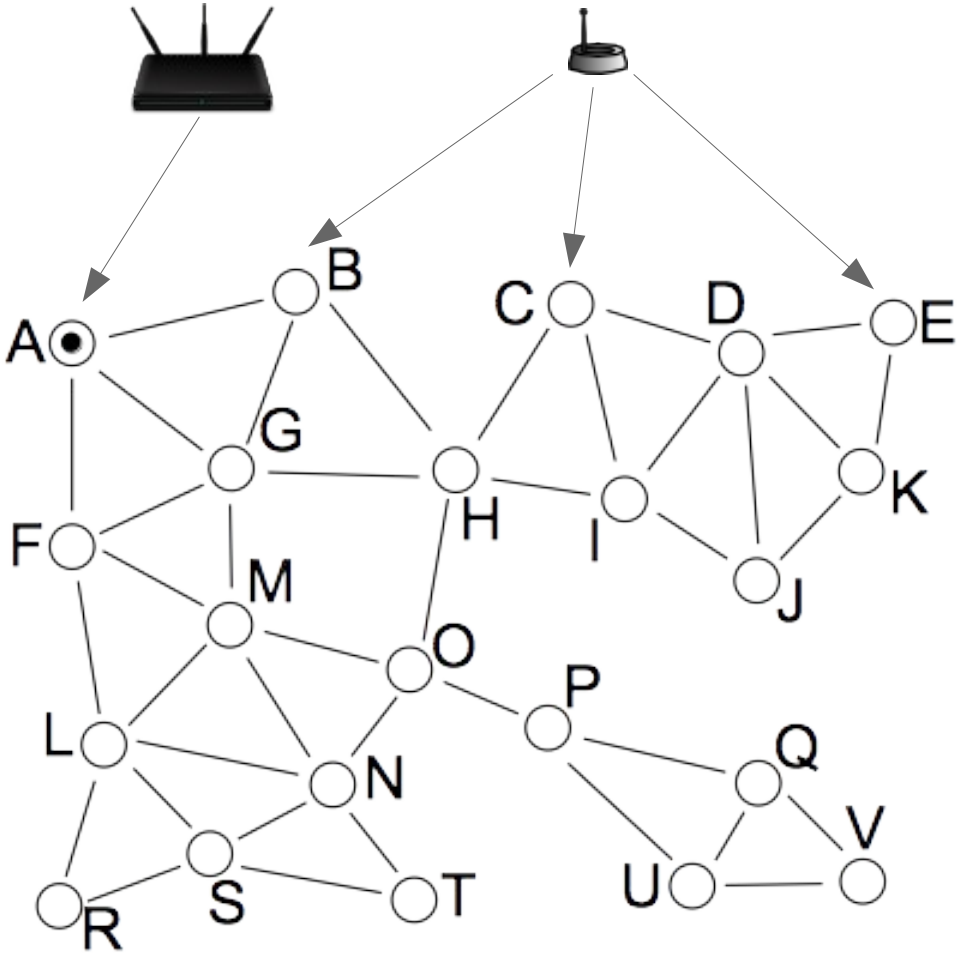
EWSN 2018, Madrid, Spain, February 16, 2018

Introduction

- A routing protocol finds paths in a network along which data can be sent.
- RPL [*RFC 6550 and several more*] is the current standard for routing in low-power wireless networks:
  - A lot of research attention.
  - Several implementations (e.g., ContikiRPL & TinyRPL).
- In industrial apps, routing protocols must be reliable.
- Our work aims at improving RPL's reliability.

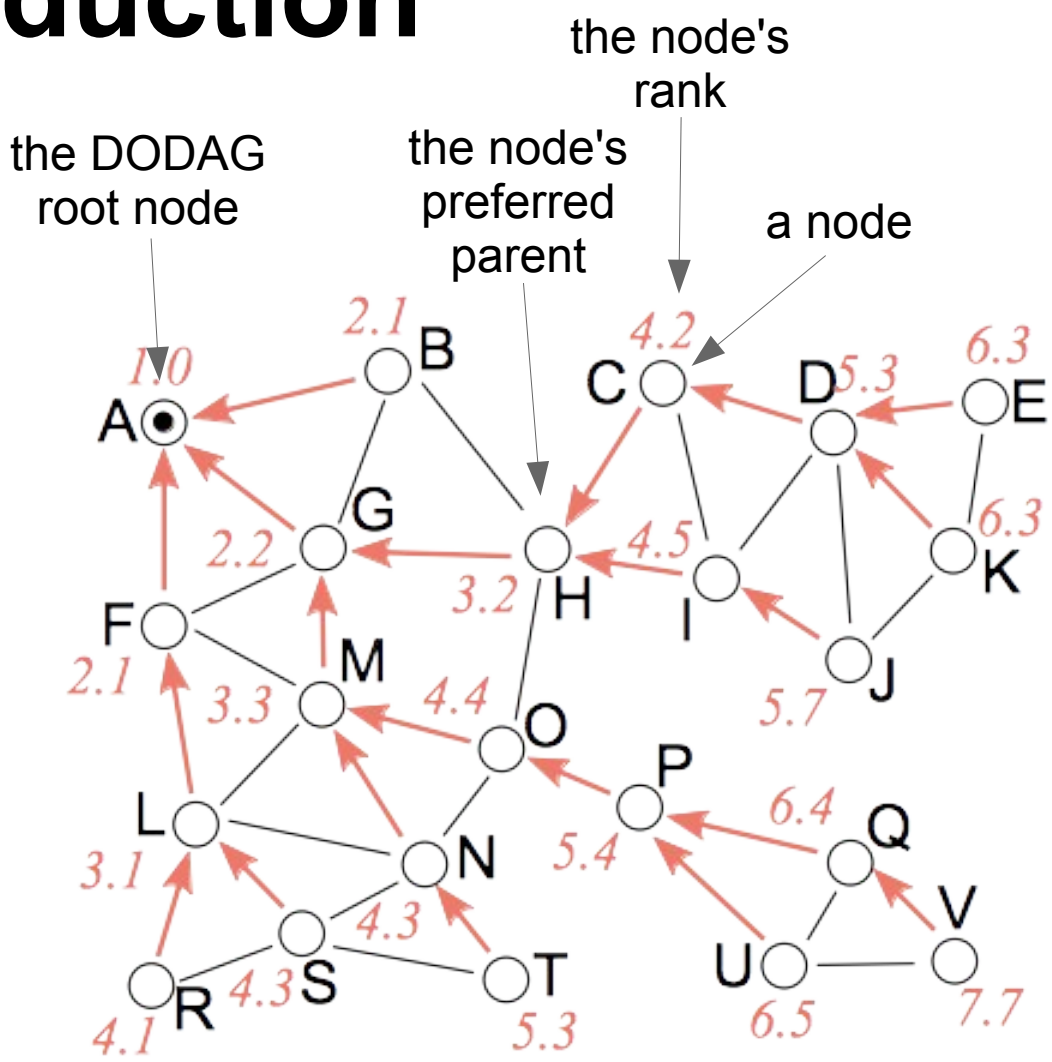


Introduction



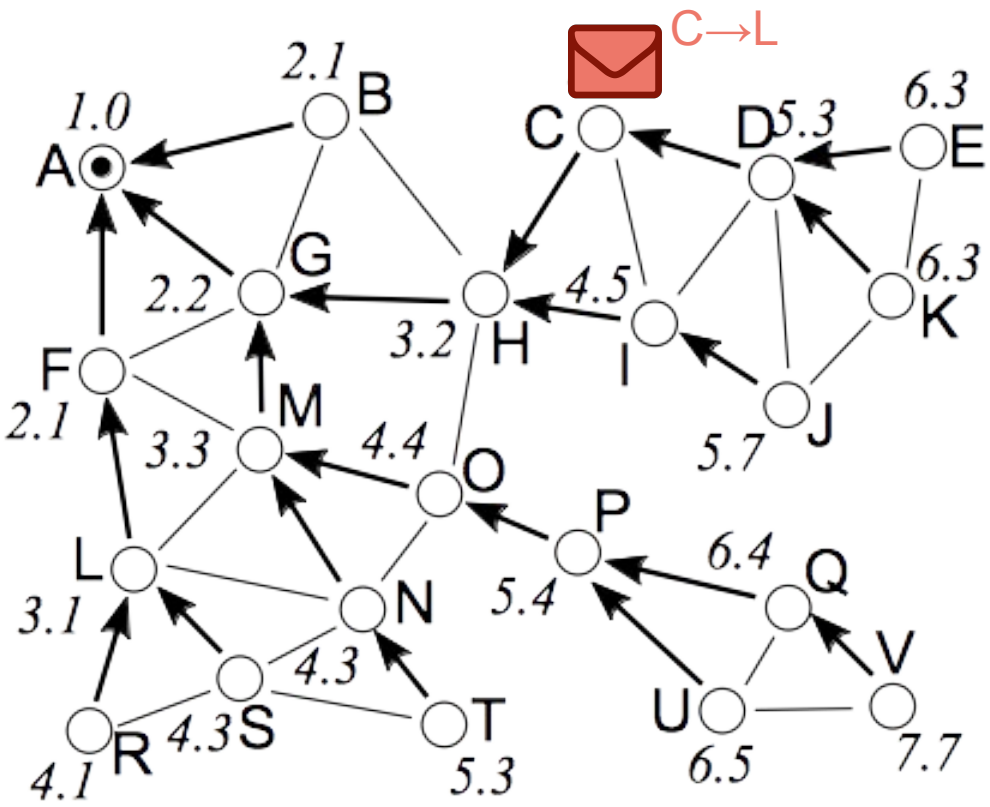
Introduction

- RPL routes in a so-called **DODAG**.



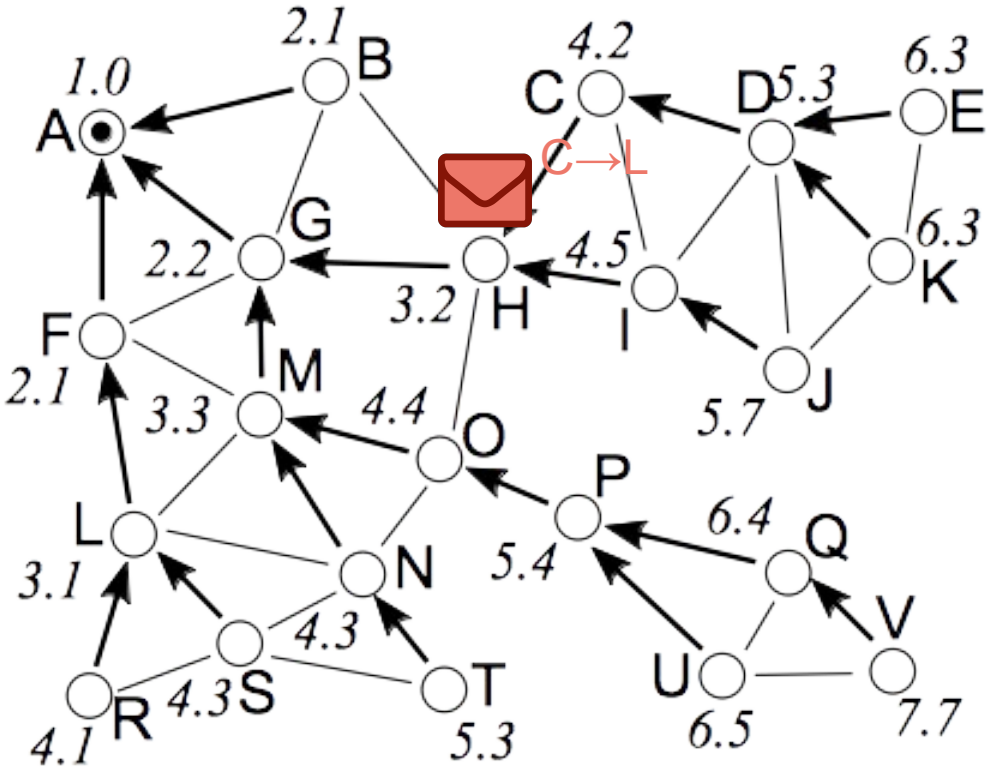
Introduction

- RPL routes in a so-called **DODAG**.



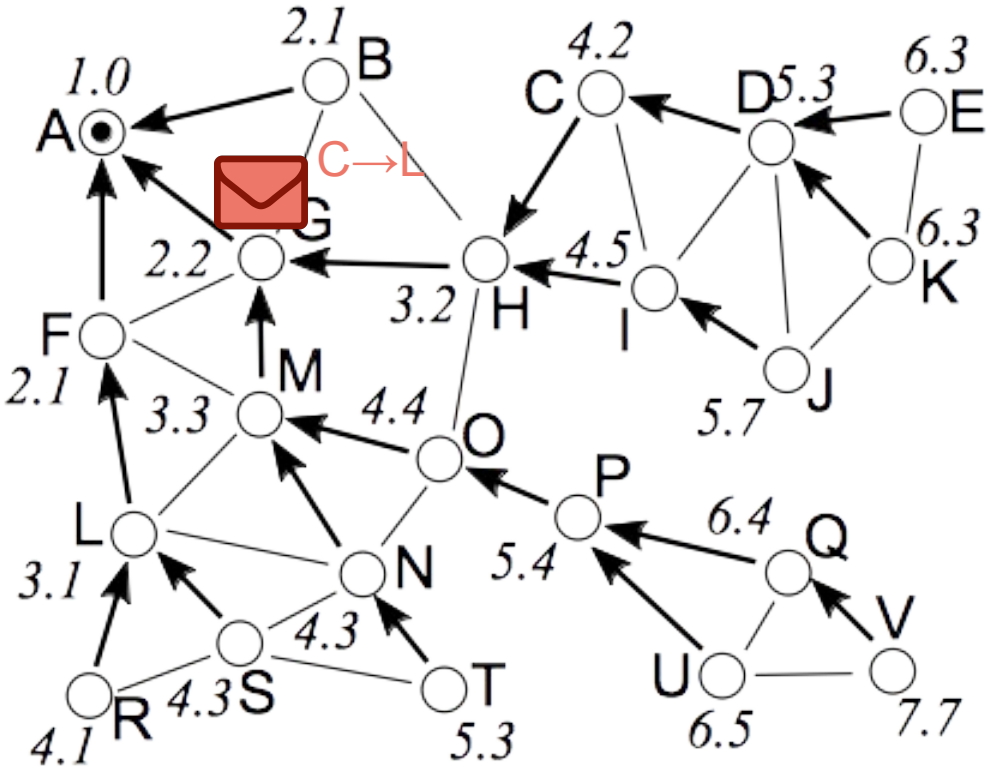
Introduction

- RPL routes in a so-called **DODAG**.



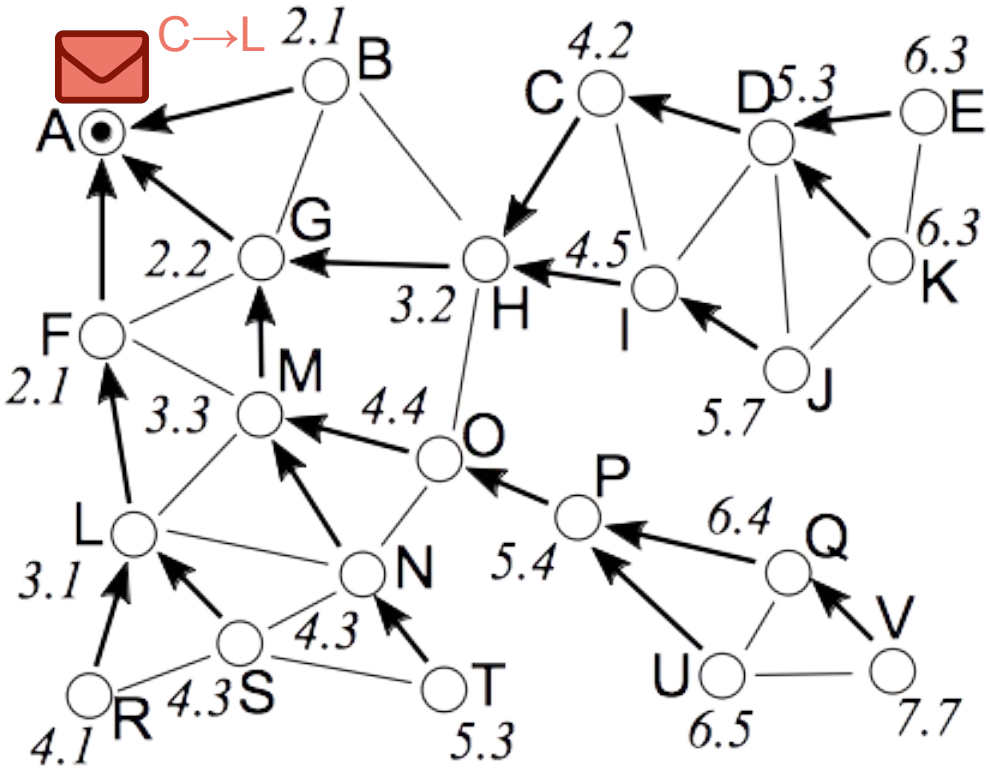
Introduction

- RPL routes in a so-called **DODAG**.



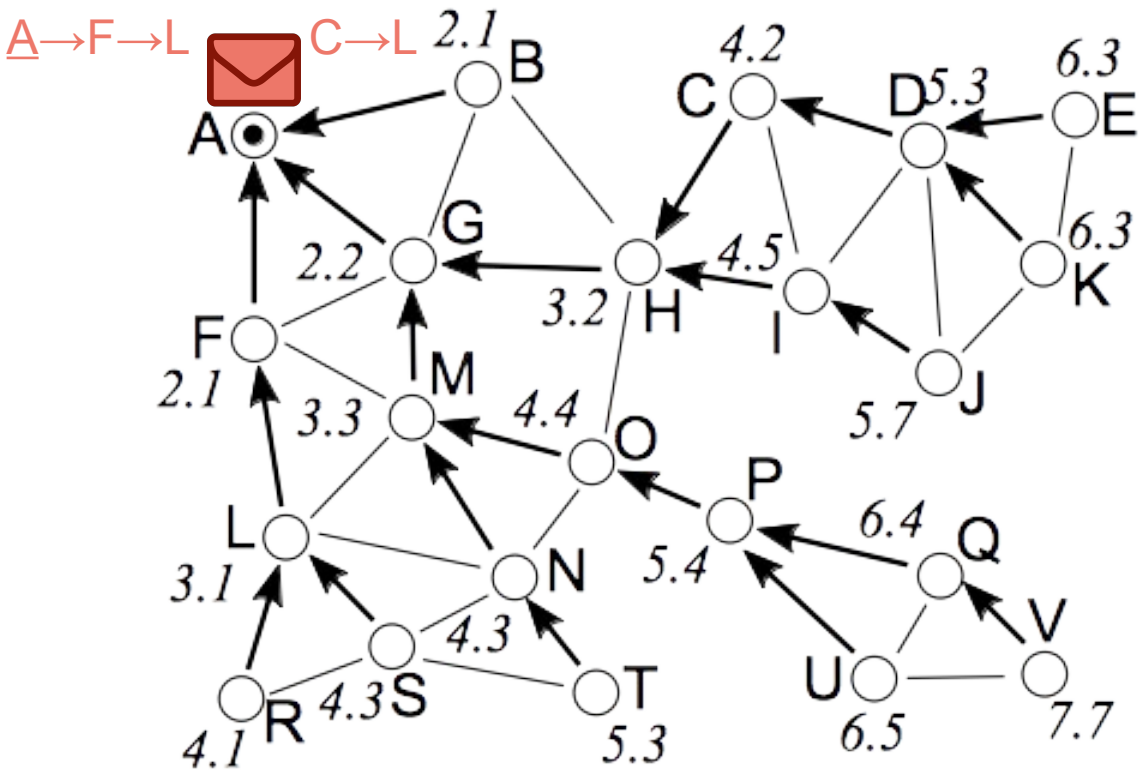
Introduction

- RPL routes in a so-called **DODAG**.



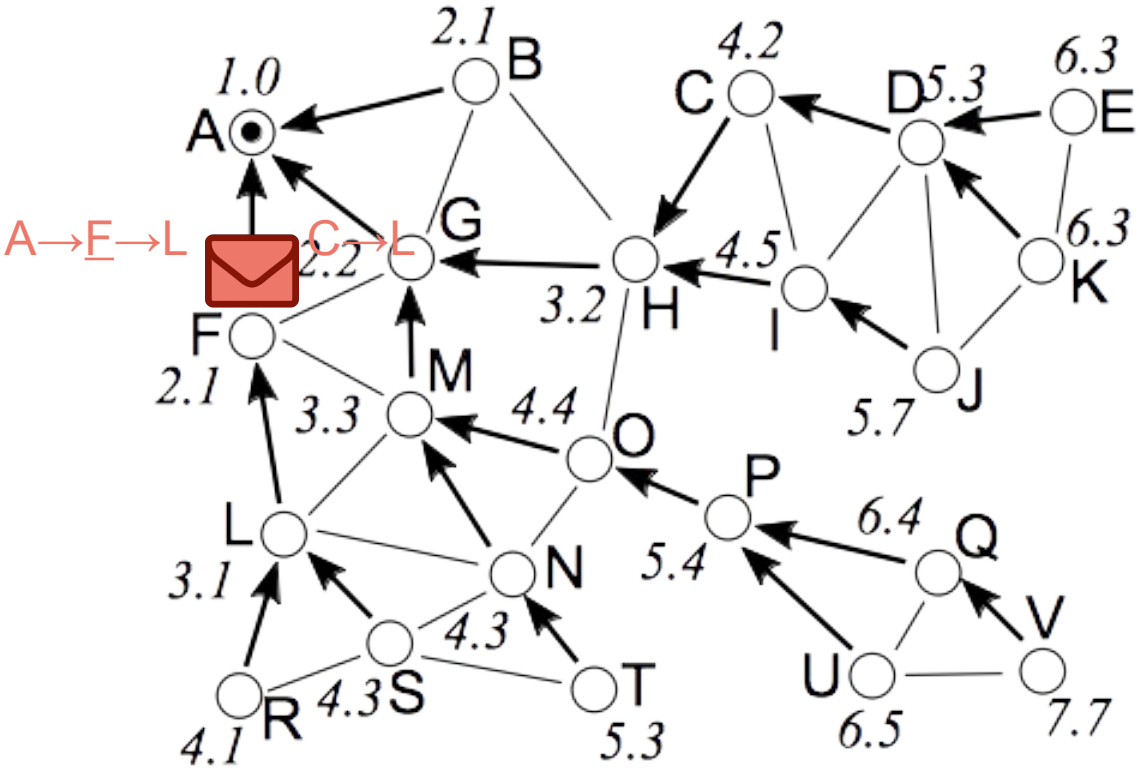
Introduction

- RPL routes in a so-called **DODAG**.



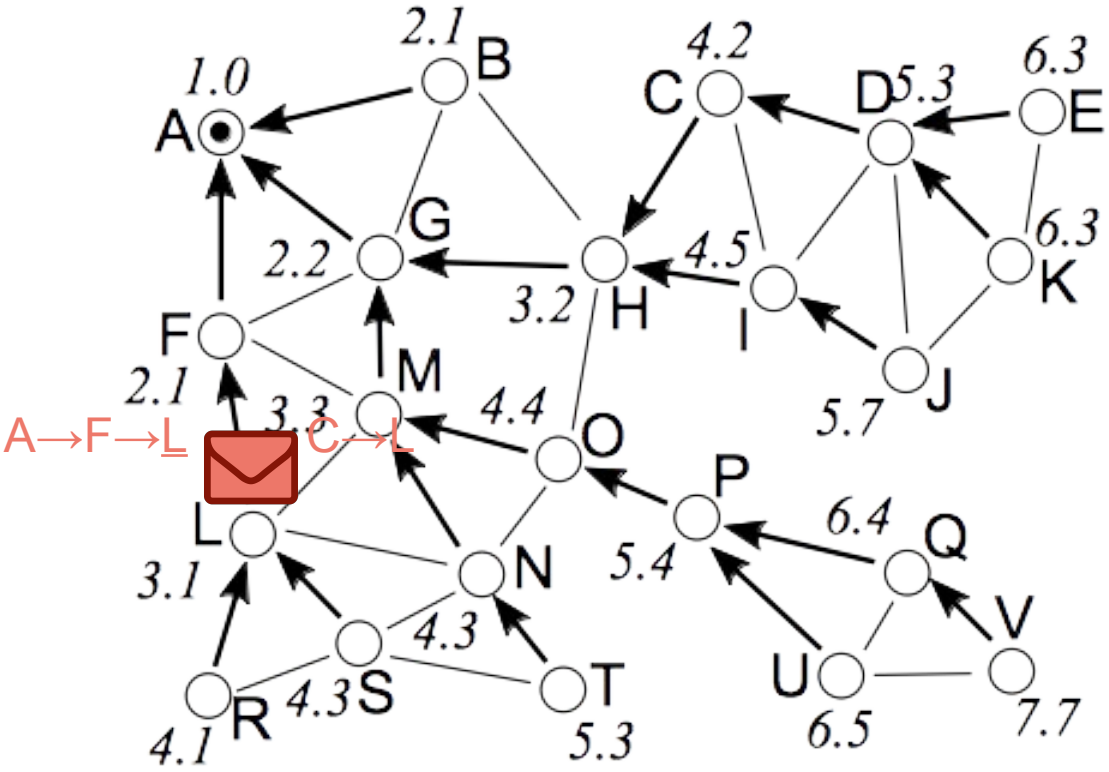
Introduction

- RPL routes in a so-called **DODAG**.



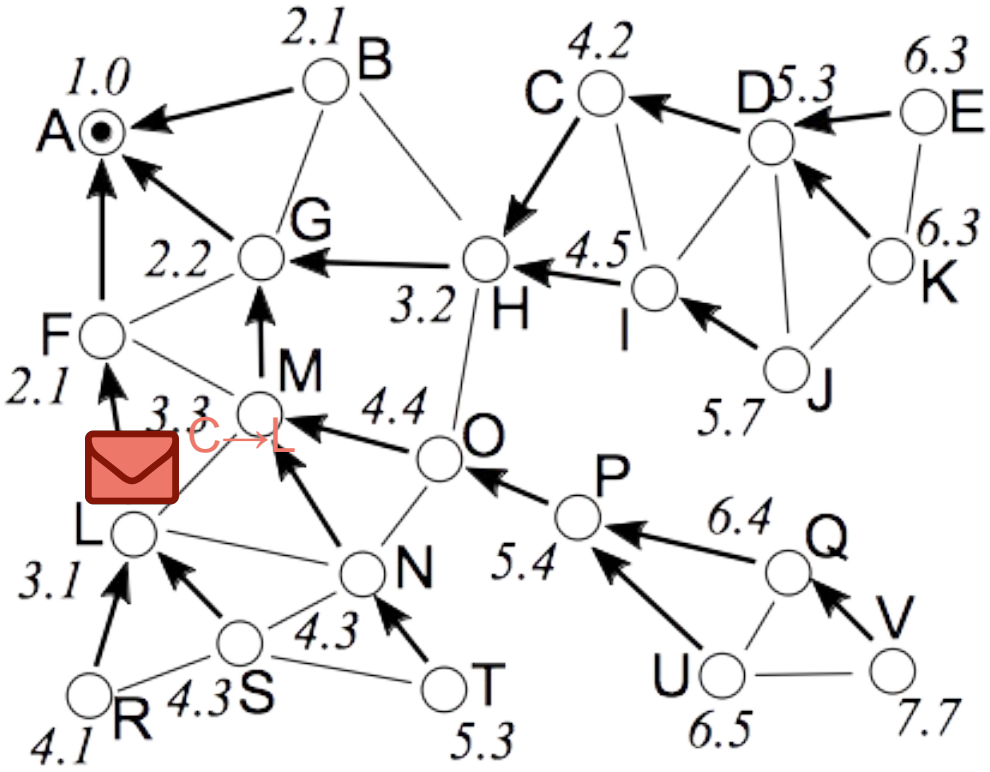
Introduction

- RPL routes in a so-called **DODAG**.



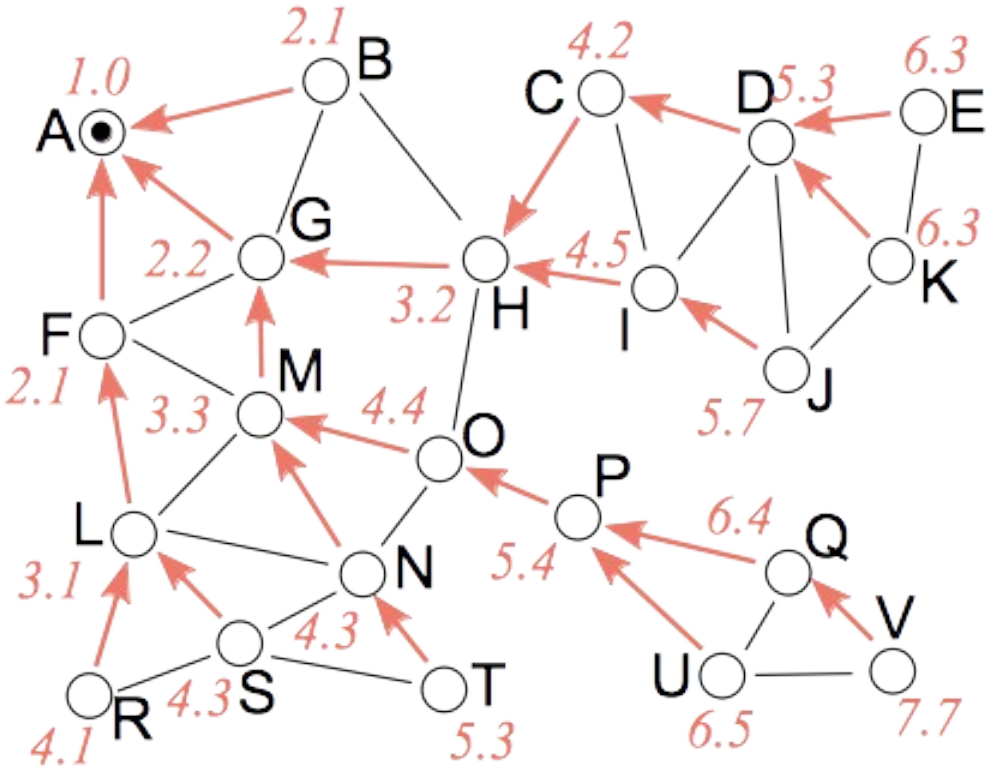
Introduction

- RPL routes in a so-called **DODAG**.



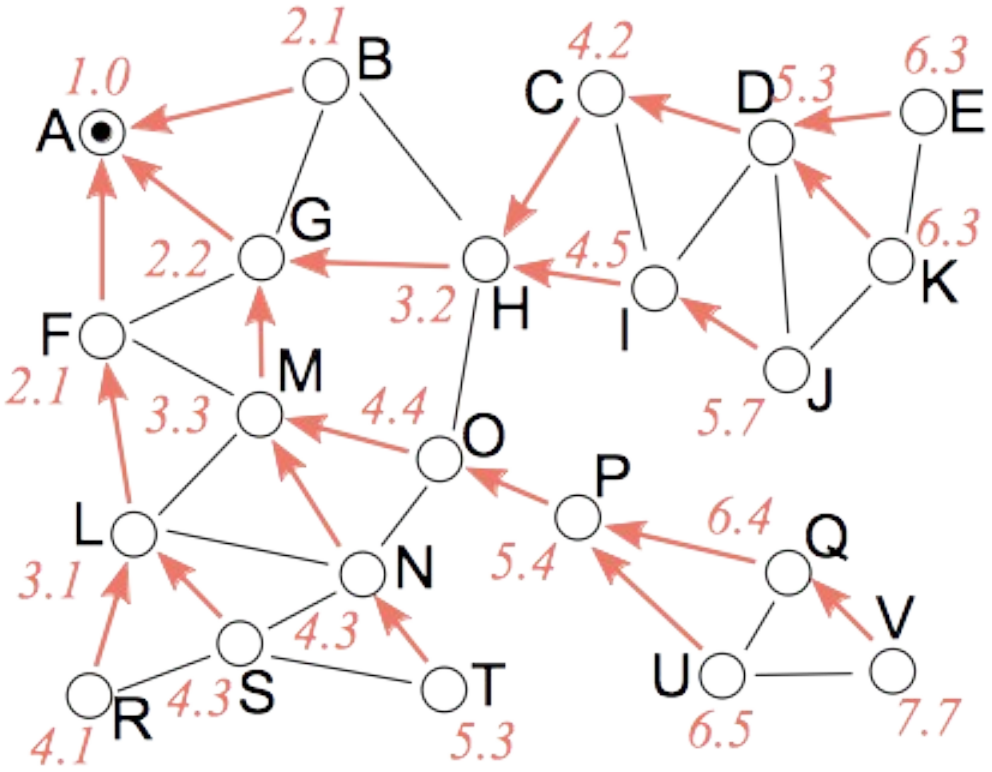
Introduction

- RPL routes in a so-called **DODAG**.
- The maintenance of the DODAG is RPL's major task:



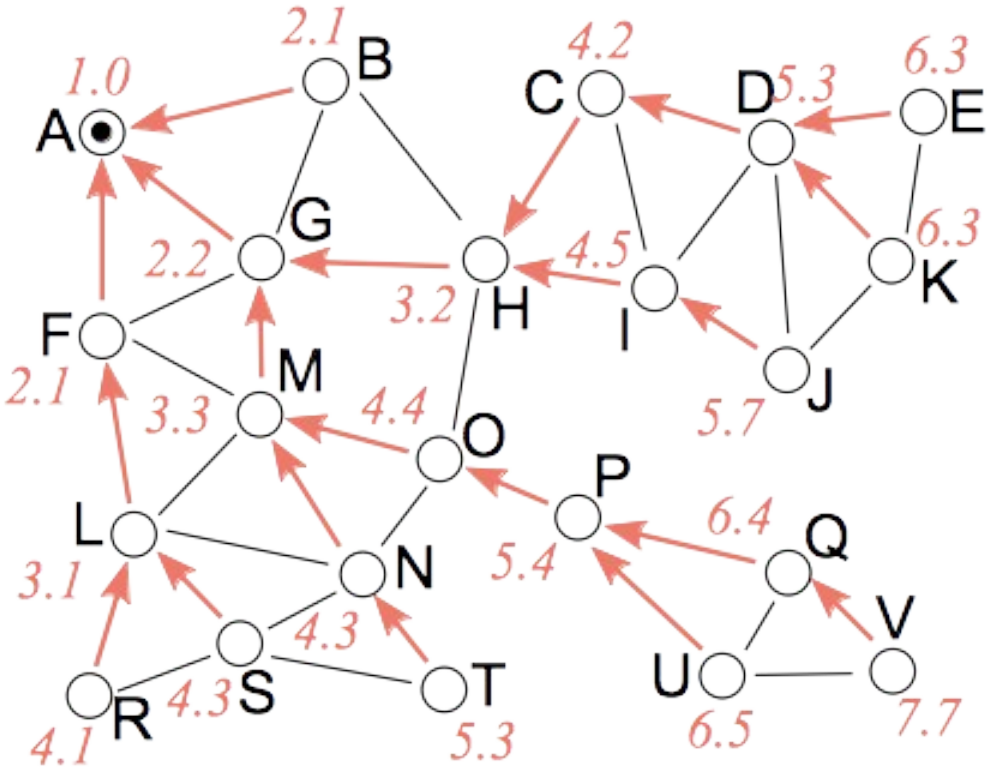
Introduction

- RPL routes in a so-called **DODAG**.
- The maintenance of the DODAG is RPL's major task:
  - The quality of the links changes continuously.



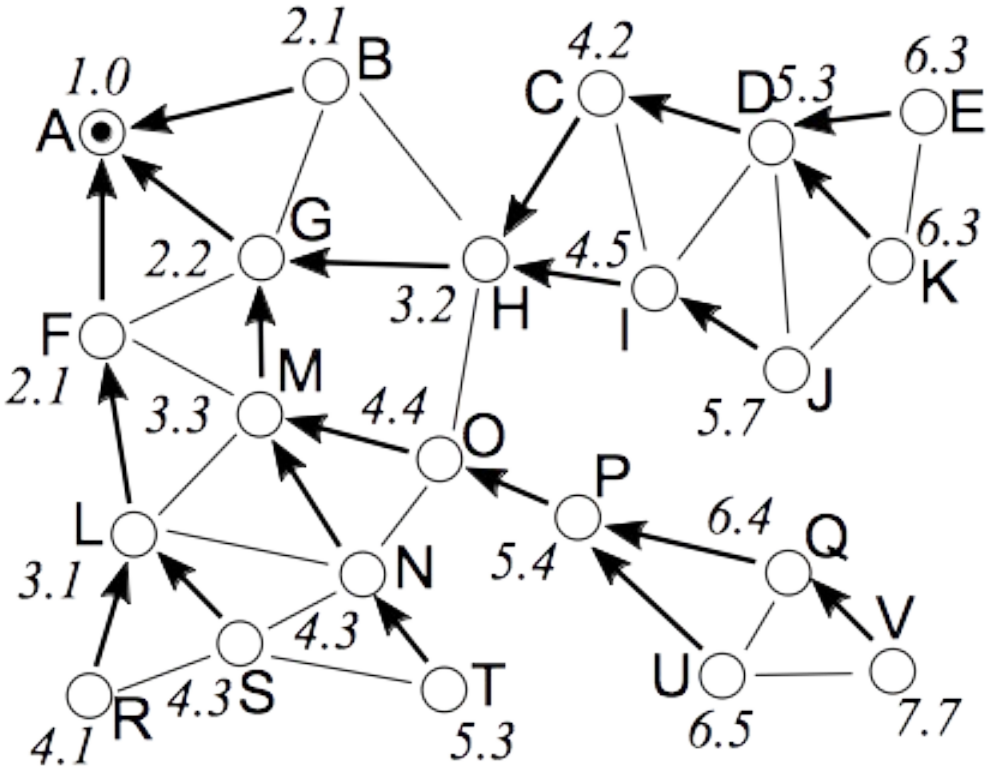
Introduction

- RPL routes in a so-called **DODAG**.
- The maintenance of the DODAG is RPL's major task:
  - The quality of the links changes continuously.
  - Nodes may fail and recover.



Introduction

- RPL routes in a so-called **DODAG**.
- The maintenance of the DODAG is RPL's major task:
  - The quality of the links changes continuously.
  - Nodes may fail and recover.
- We are concerned with hard-to-handle failures: **network partitions**.

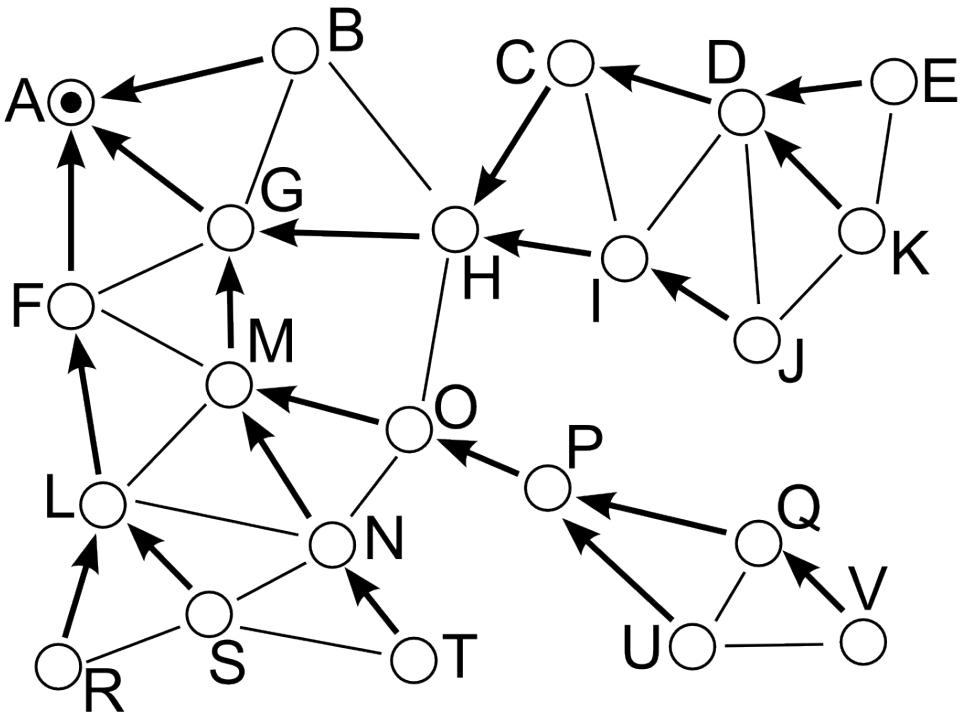


Problem

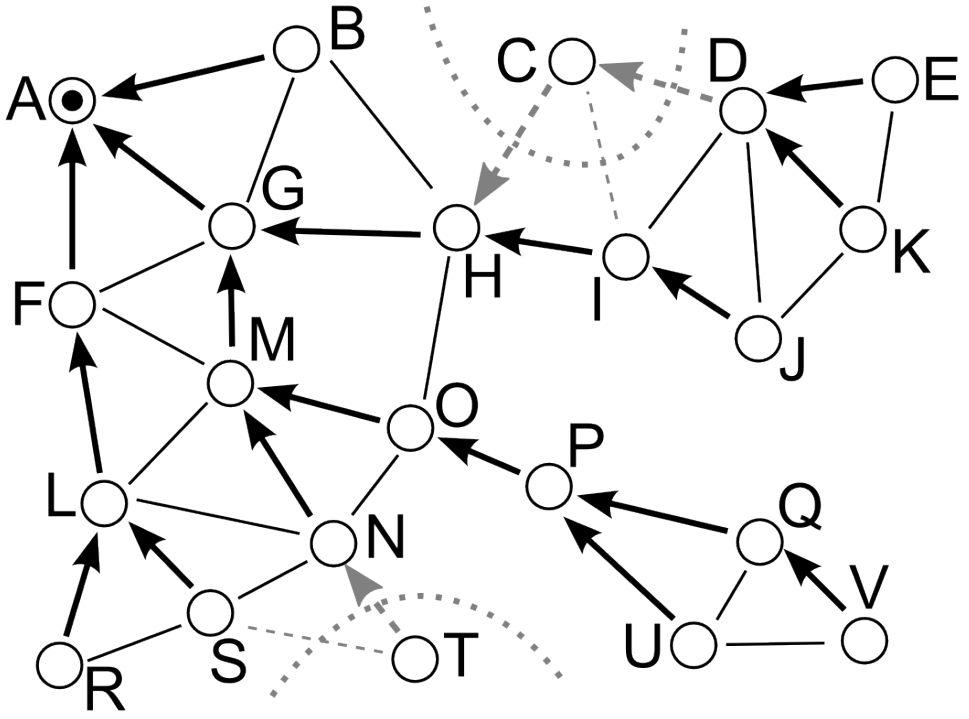
A network is **partitioned** / **split** / **disconnected** if and only if it contains two working nodes between which no path exists via the DODAG root.

Problem

A connected network.



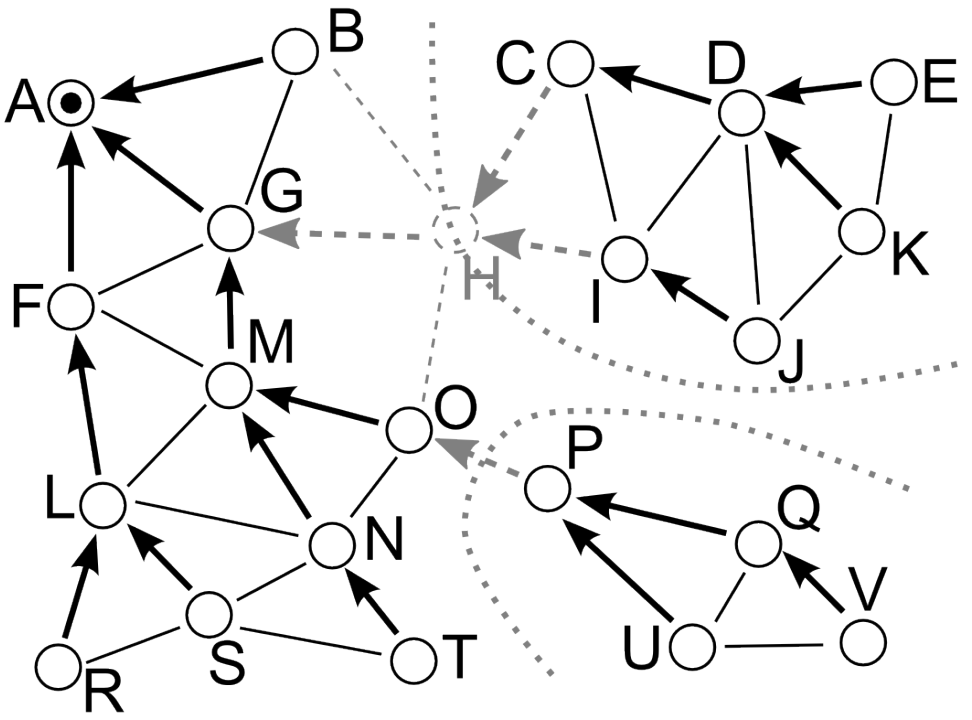
Problem



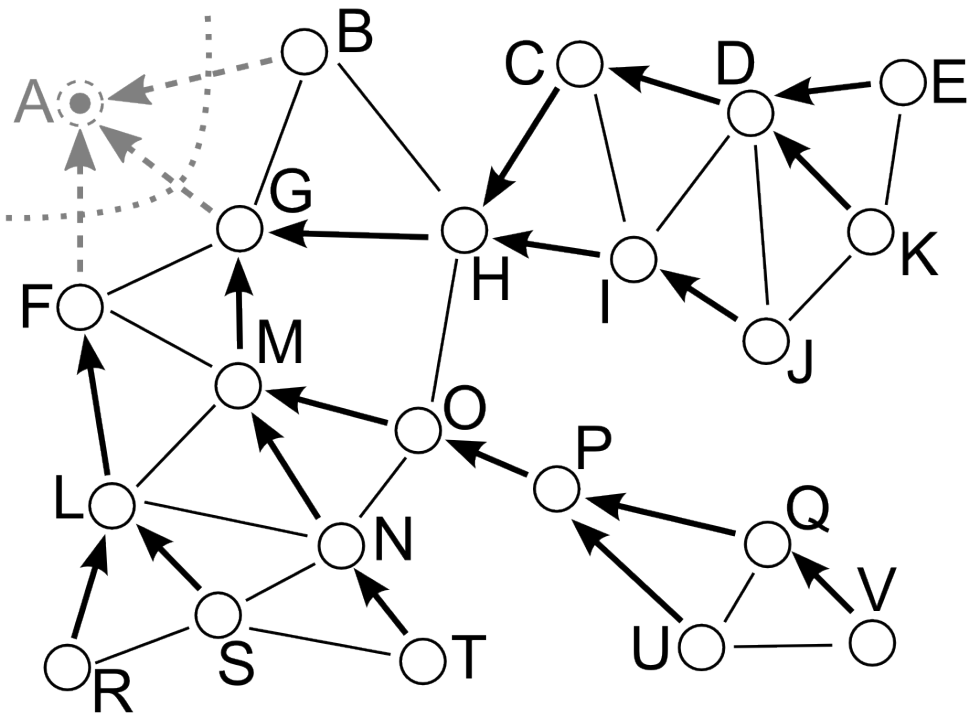
Isolated nodes.

Problem

Multi-node partitions.



Problem



The DODAG root's crash.

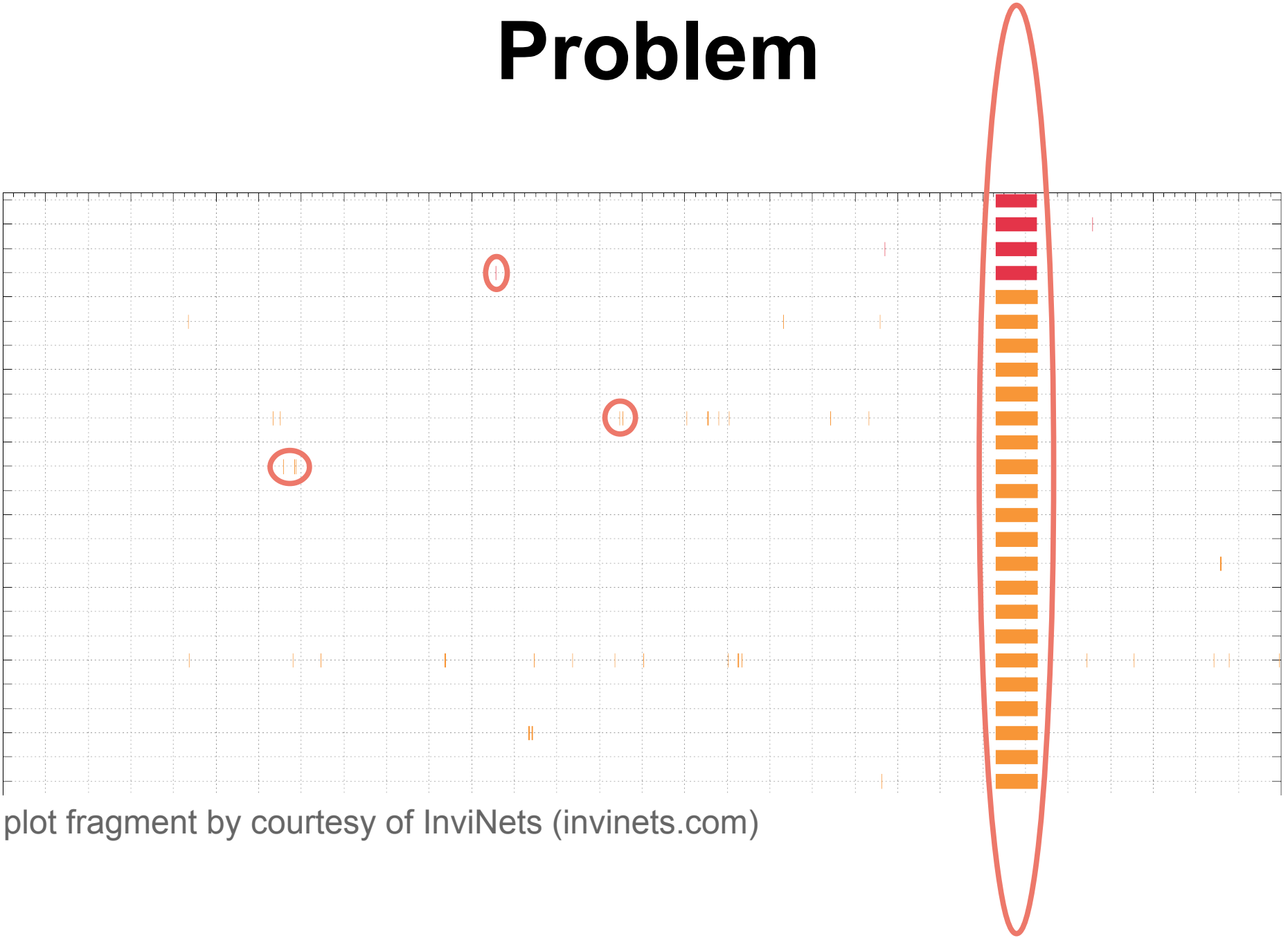
Problem

How relevant is the problem?

Problem

photos by courtesy of InviNets (invinets.com)

Problem



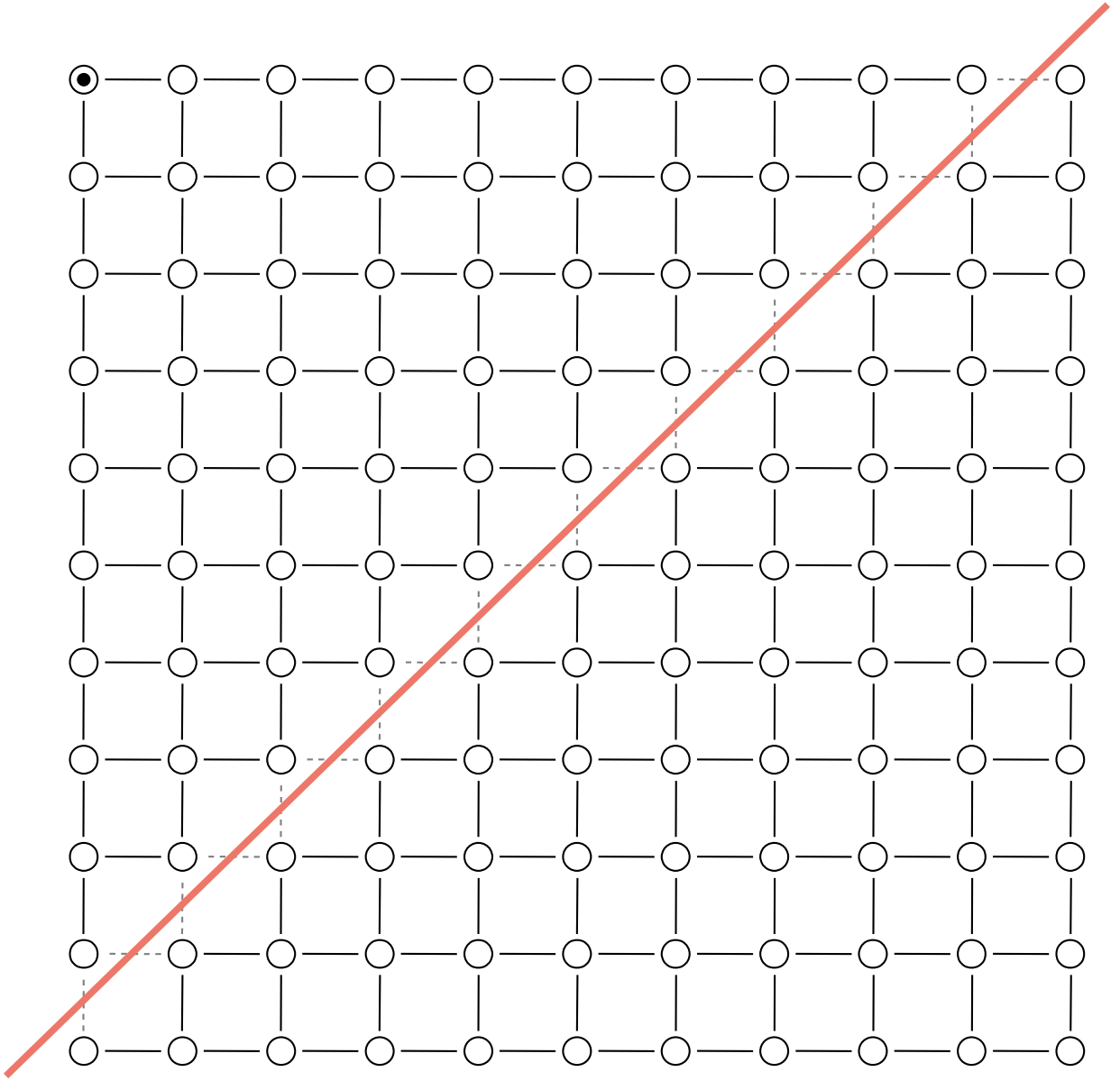
plot fragment by courtesy of InviNets (invinets.com)

RPL Implementations & Partitions

Let us illustrate the behavior of RPL's two popular implementations:

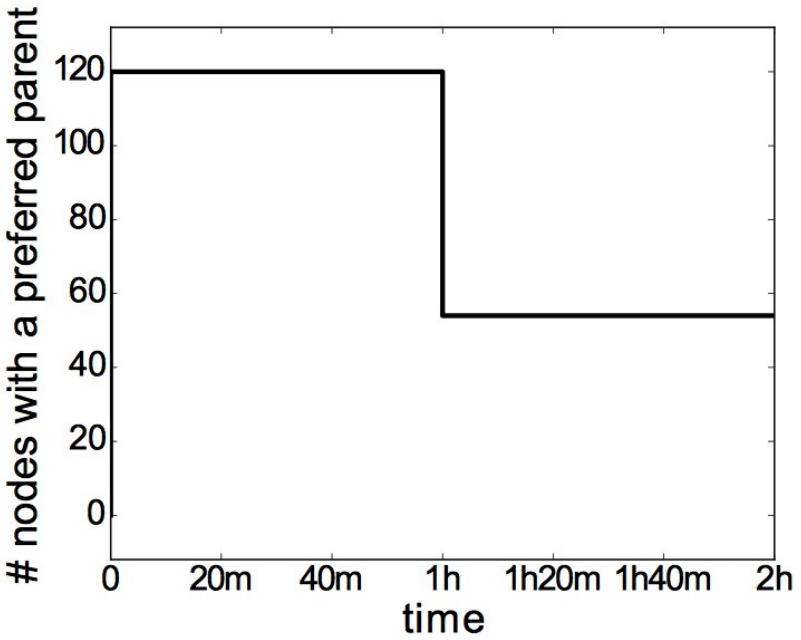
- ContikiRPL
- TinyRPL

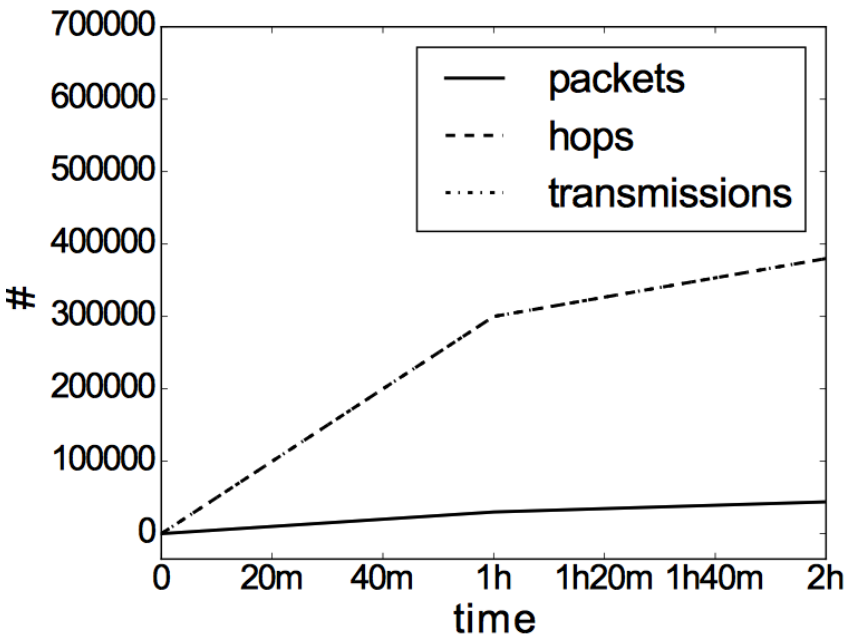
in a sample simulated network partition scenario.



RPL Implementations & Partitions

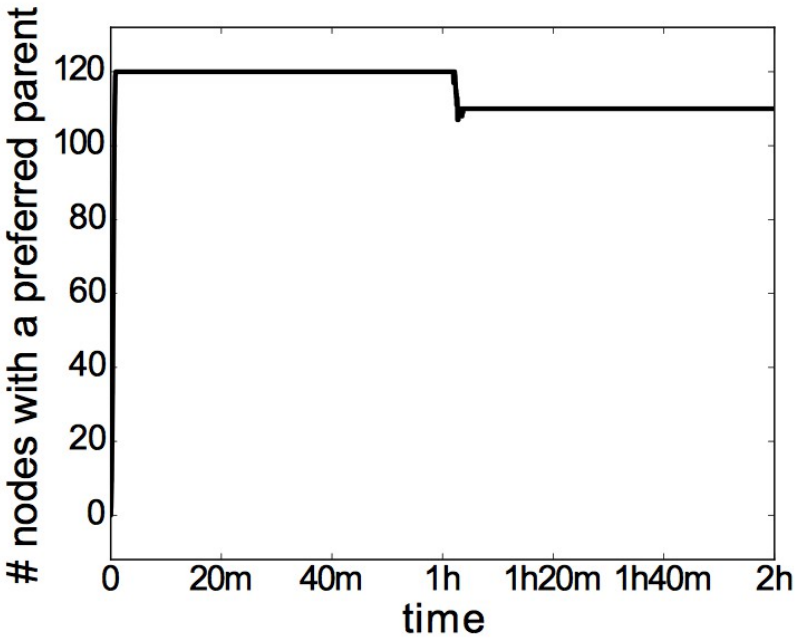
Expected behavior

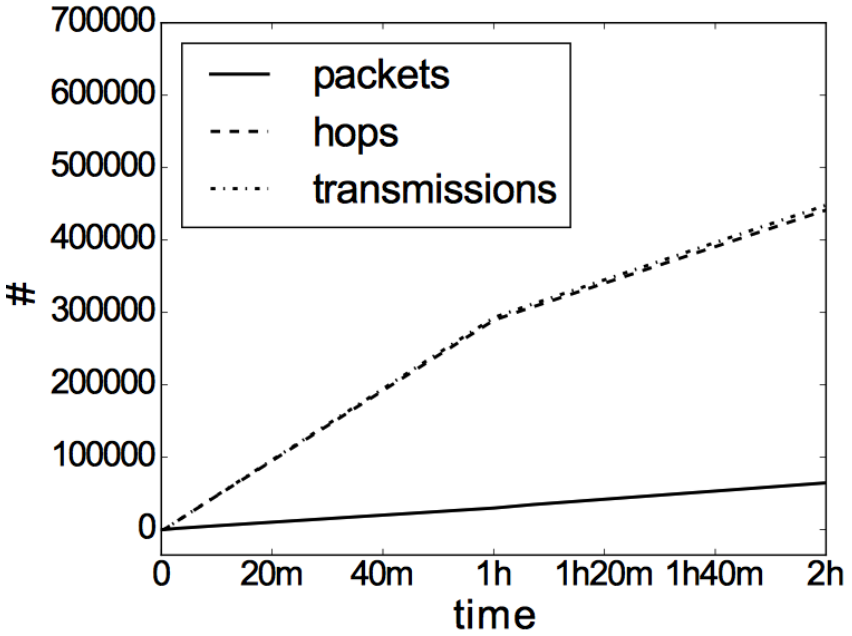




RPL Implementations & Partitions

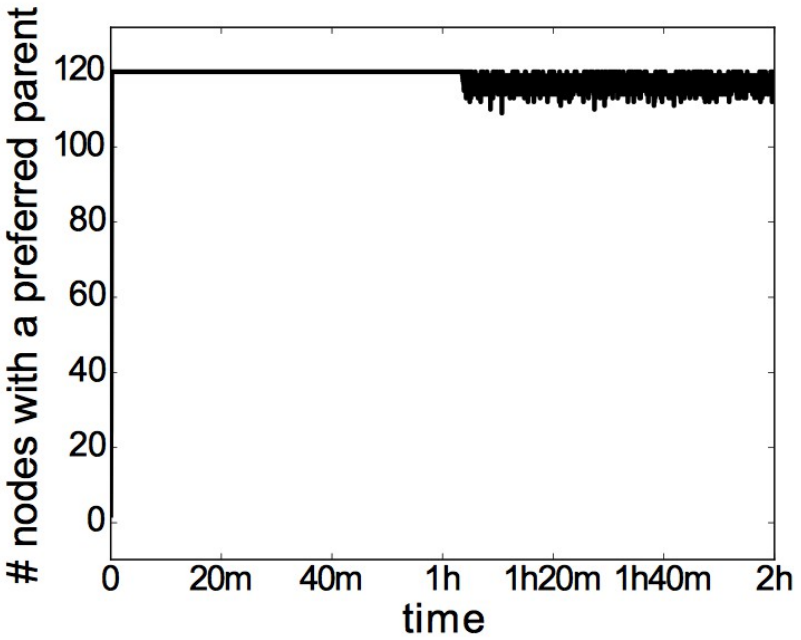
ContikiRPL's behavior

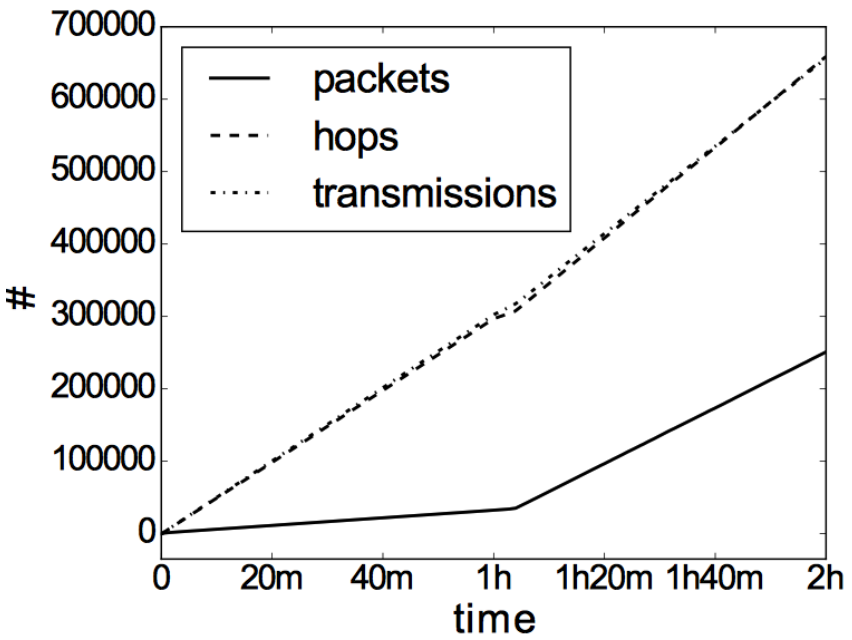




RPL Implementations & Partitions

TinyRPL's behavior





RPL Implementations & Partitions

Is this behavior of RPL's implementations due to:

- bugs in those implementations *OR*
- some features or RPL's design itself?

Analyzing RPL

- We developed a model:
  - using a LTL-like formalism;
  - based on RPL's specification;
  - in abstraction from the issues the specification leaves open to implementations;
  - captures RPL's dynamic behavior.
- Using the model we formulated a hypothesis that RPL is able to handle network partitions.
- We subsequently attempted to prove the hypothesis.

Analyzing RPL

Property examples:

RA1: Always, if a node's *NeighborSet* changes (i.e., entries are added or removed, or their fields are modified), then the node will eventually reselect its *PreferredParent* and *Rank*.

RA3: If a node's neighbor is eventually always non-adjacent, then an entry, *n*, with this neighbor's identifier as *n.Id* will eventually always either be absent from the node's *NeighborSet* or have its *n.Reachable* field set to *False*.

OF2: A node's *MinRank* is always equal to the minimal value of the node's *Rank* so far.

DIO2: If a node's neighbor is always eventually adjacent, then the node will always eventually receive a DIO message from this neighbor.

Analyzing RPL

Main hypothesis:

Hypothesis 1: If an always-eventually-live node is eventually always partitioned from the DODAG root, then eventually always, whenever the node is live, its *PreferredParent* and *Rank* will be equal to *Null* and *Infinity*, respectively.

Analyzing RPL

Proof scheme:

Analyzing RPL

Proof scheme:

1. Show that, from some moment in time, partitioned nodes will only select preferred parents among themselves (Lemma 1).

Analyzing RPL

Proof scheme:

1. Show that, from some moment in time, partitioned nodes will only select preferred parents among themselves (Lemma 1).
2. Show that, from some moment in time, the minimum over the ranks of the nodes in a partition will never decrease (Lemma 2).

Analyzing RPL

Proof scheme:

1. Show that, from some moment in time, partitioned nodes will only select preferred parents among themselves (Lemma 1).
2. Show that, from some moment in time, the minimum over the ranks of the nodes in a partition will never decrease (Lemma 2).
3. Show that, from some moment in time, the aforementioned minimum will repeatedly increase (Lemma 3).

Analyzing RPL

Proof scheme:

1. Show that, from some moment in time, partitioned nodes will only select preferred parents among themselves (Lemma 1).
2. Show that, from some moment in time, the minimum over the ranks of the nodes in a partition will never decrease (Lemma 2).
3. Show that, from some moment in time, the aforementioned minimum will repeatedly increase (Lemma 3).
4. Show that a node's rank cannot increase infinitely (Lemma 4).

Application of the Results

Application of the Results

RPL's implementations fail to handle partitions.

Application of the Results

RPL's design guarantees handling network partitions.

RPL's implementations fail to handle partitions.

Application of the Results

RPL's design guarantees handling network partitions.

RPL's implementations fail to handle partitions.

The implementations must be buggy:

- They must violate some of the properties we formulated.

Application of the Results

RPL's design guarantees handling network partitions.

RPL's implementations fail to handle partitions.

The implementations must be buggy:

- They must violate some of the properties we formulated.

We analyzed the two implementations for the violations of our properties.

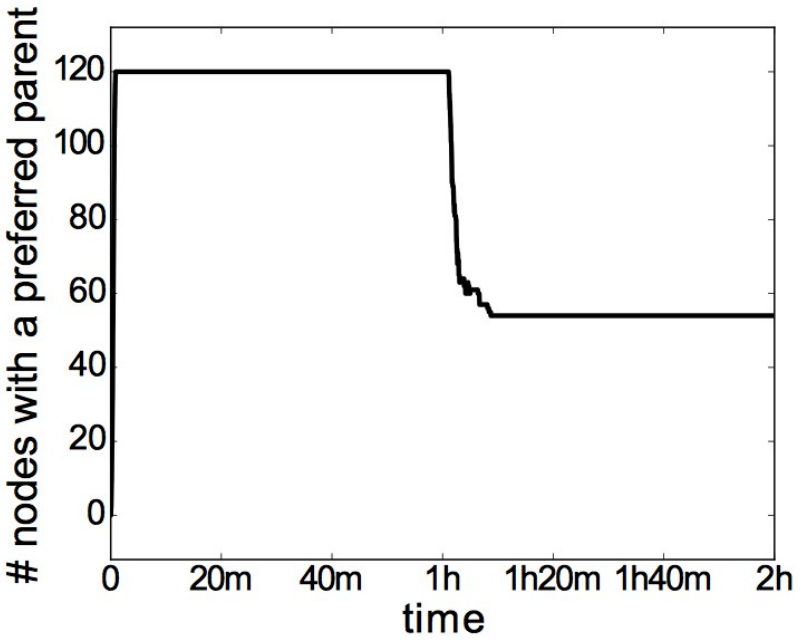
Application of the Results

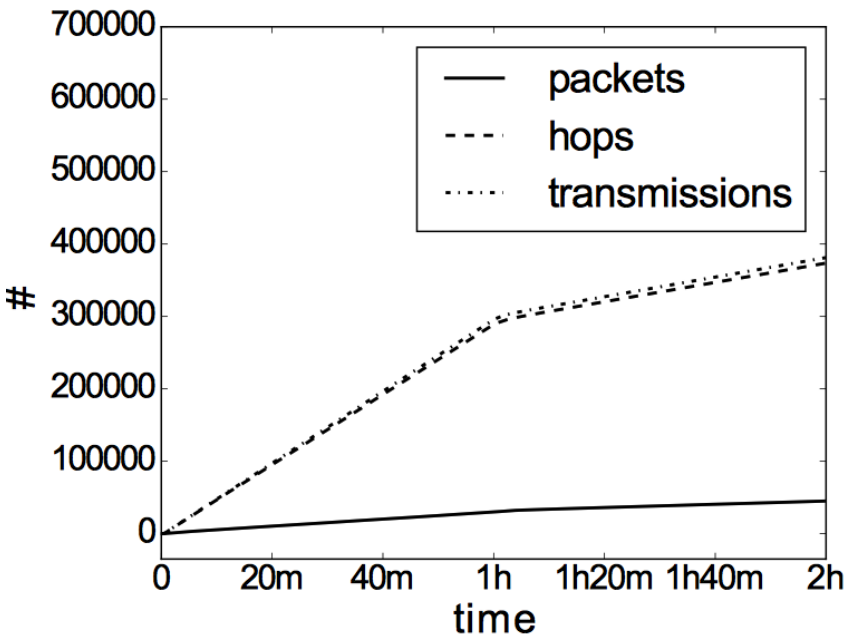
Property	Is property not violated in the source code?	
	ContikiRPL	TinyRPL
RA1	yes	NO
RA2	yes	yes
RA3	NO	yes
OF1	yes	yes
OF2	NO	NO
OF3	yes	yes
OF4	yes	yes
OF5	yes	yes
DIO1	yes	yes
DIO2	yes	NO
DIO3	yes	yes

We patched these bugs.

RPL's Behavior after the Fixes

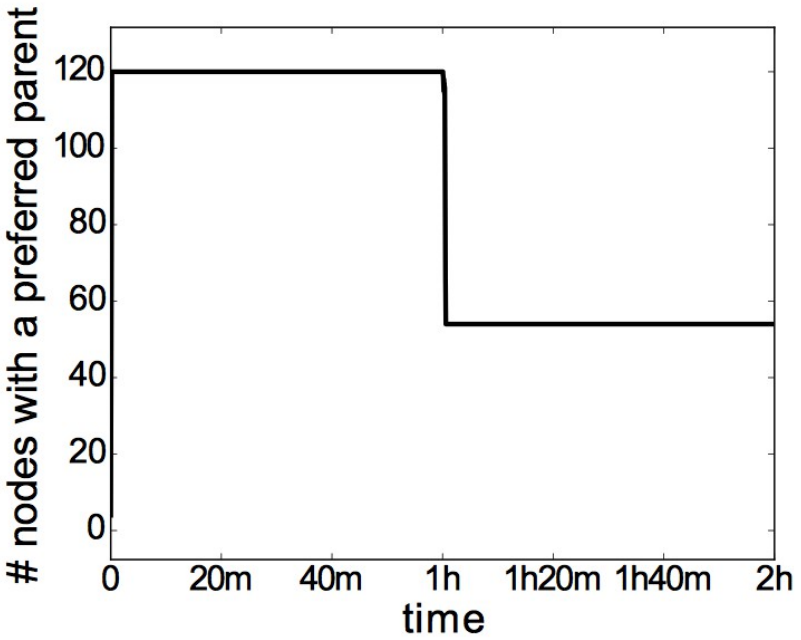
ContikiRPL's behavior

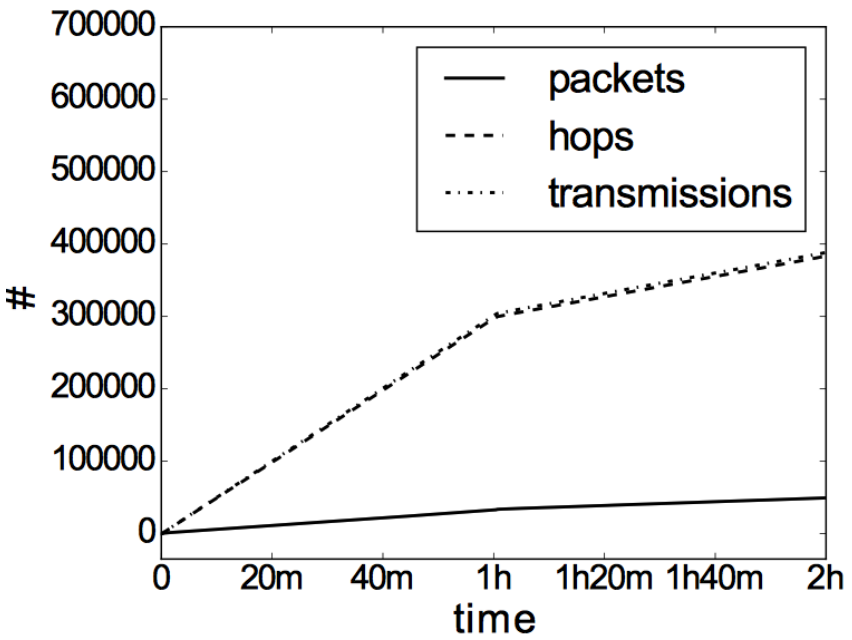




RPL's Behavior after the Fixes

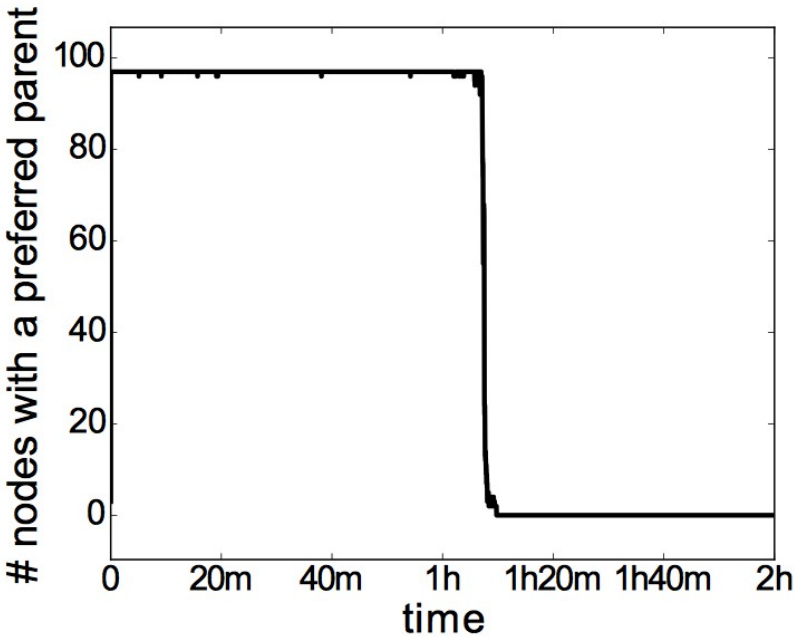
TinyRPL's behavior

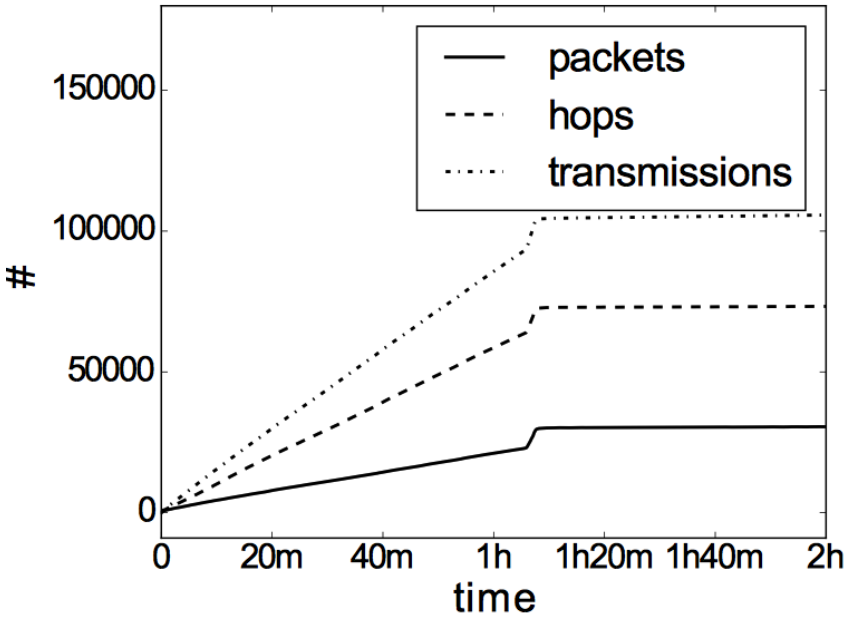




RPL's Behavior after the Fixes

TinyRPL on Indryia with the DODAG root's crash





Conclusions

Conclusions

- RPL's popular implementations do not work correctly under network partitions.

Conclusions

- RPL's popular implementations do not work correctly under network partitions.
- In contrast, the protocol itself should, by design, handle partitions as expected, which we have formally proved.

Conclusions

- RPL's popular implementations do not work correctly under network partitions.
- In contrast, the protocol itself should, by design, handle partitions as expected, which we have formally proved.
- Using the properties derived in the formal analyses, we have identified and fixed the flaws in the implementations.
  - Some properties required by RPL's design need to be maintained in many places in the code.
  - Others concerned issues that are left open in RPL's spec.

Conclusions

- RPL's popular implementations do not work correctly under network partitions.
- In contrast, the protocol itself should, by design, handle partitions as expected, which we have formally proved.
- Using the properties derived in the formal analyses, we have identified and fixed the flaws in the implementations.
  - Some properties required by RPL's design need to be maintained in many places in the code.
  - Others concerned issues that are left open in RPL's spec.
- The overall conclusion is that despite the seeming maturity of RPL's implementations, their reliability leaves room for improvement.

Thank you

Questions?

The presented research was supported by the National Center for Research and Development (NCBR) in Poland under grant no. LIDER/434/L-6/14/NCBR/2015.

The National Centre
for Research and Development

LIDER