Design and Analysis of Routing Protocol for IPv6 Wireless Sensor Networks

Elias Wendm Atalay

Supervisor
Prof. Enzo Mingozzi

Supervisor
Prof. Giuseppe Anastasi

Co-Supervisor
Dott. Carlo Vallati

A thesis presented for the degree of Master of Science in Embedded Computing Systems

Department of Engineering
Scuola Superiore Sant’anna and University of Pisa
Italy

Academic year 2014/2015
CONTENTS

List of Figures ............................................................................................................. 4

Chapter ONE ................................................................................................................. 5

Introduction .................................................................................................................... 5

Chapter TWO ................................................................................................................ 7

Background ................................................................................................................... 7

2.1 RPL ......................................................................................................................... 7

2.2 Topology Formation ............................................................................................... 8

2.3 Routing in RPL ....................................................................................................... 8

2.3.1. Routing upward ............................................................................................... 8

2.3.2 Downward routing ............................................................................................. 9

2.4 RPL Messages ....................................................................................................... 9

2.4.1 DODAG Information Object (DIO) ................................................................ 10

2.4.2 DODAG Information Solicitations (DIS) ......................................................... 10

2.4.3 DODAG Destination Advertisement Object (DAO) ......................................... 10

2.5 Routing metrics .................................................................................................... 10

2.6 Routing Constraint ............................................................................................... 11

2.7 Objective Function ............................................................................................... 11

2.8 Trickle Algorithm ................................................................................................. 12

2.8.1 Trickle Algorithm Overview .......................................................................... 12

2.8.2 Trickle Parameters and Variables .................................................................. 13

2.8.3 TRICKLE Algorithm ...................................................................................... 14

2.9 Trickle-S ................................................................................................................ 16

2.10 Trickle-F .............................................................................................................. 17

Chapter THREE .......................................................................................................... 20

Methods ........................................................................................................................ 20

3.1 Contiki Operating System .................................................................................... 21

3.2 Cooja Simulator .................................................................................................... 21
Chapter FOUR .......................................................... 24
Evaluation ...................................................................... 24
4.1 Performance metrics .................................................. 24
4.2 RPL parameters.......................................................... 25
   4.2.1 DIO Interval Minimum .......................................... 25
   4.2.2 DIO Interval Doublings .......................................... 25
   4.2.3 Redundancy Threshold ........................................... 26
4.3 Simulation and Network Setup ..................................... 26
   4.3.1 Link Failure Model .............................................. 26
   4.3.2 Network Setup .................................................. 27
      4.3.2.1 Network Setup Phase 1 .................................... 27
      4.3.2.2 Network Setup Phase 2 .................................... 28
4.4 Measuring the performance metrics .............................. 29
4.5 Phase1: Performance metrics evaluation ....................... 30
   4.5.1 DIO Interval Minimum .......................................... 30
   4.5.2 DIO Interval Doublings .......................................... 34
   4.5.3 Redundancy Threshold ........................................... 38
4.6 Phase2: Performance metrics evaluation ....................... 41
   4.6.1 DIO Interval Minimum .......................................... 41
   4.6.2 DIO Interval Doublings .......................................... 45
   4.6.3 Redundancy Threshold ........................................... 48
4.7 Discussion and Analysis ............................................. 51
Chapter FIVE .................................................................. 52
5.1 Trickle-Fair broadcast suppression and Synchronizing Trickle intervals ....................... 52
5.2 Performance evaluation of Trickle-FS ............................ 54
Chapter SIX .................................................................... 58
Discussion ...................................................................... 58
Chapter Seven .................................................................. 60
Conclusion ....................................................................... 60
Acknowledgment ............................................................. 61
References ....................................................................... 62
List of Figures

Fig 4. 1 Radio Medium by UDGM ................................................................. 27
Fig 4. 2 A network of 100 nodes ................................................................. 28
Fig 4. 3 A network of 500 nodes ................................................................. 29
Fig 4. 4 Network Formation Time ............................................................... 31
Fig 4. 5 Control Traffic overhead ............................................................... 32
Fig 4. 6 Energy Consumption ................................................................. 33
Fig 4. 7 Packet Delivery Ratio ................................................................. 34
Fig 4. 8 Network Formation Time ............................................................... 35
Fig 4. 9 Control Traffic Overhead ............................................................... 36
Fig 4. 10 Energy Consumption ................................................................. 37
Fig 4. 11 Packet Delivery Ratio ................................................................. 38
Fig 4. 12 Network Formation Time ............................................................... 39
Fig 4. 13 Control Traffic Overhead ............................................................... 39
Fig 4. 14 Energy Consumption ................................................................. 40
Fig 4. 15 Packet Delivery Ratio for K .......................................................... 41
Fig 4. 16 Network Formation Time ............................................................... 42
Fig 4. 17 Control Traffic Overhead ............................................................... 43
Fig 4. 18 Energy Consumption ................................................................. 44
Fig 4. 19 Packet Delivery Ratio ................................................................. 45
Fig 4. 20 Network Formation Time ............................................................... 46
Fig 4. 21 Control Traffic Overhead ............................................................... 46
Fig 4. 22 Energy Consumption ................................................................. 47
Fig 4. 23 Packet Delivery Ratio ................................................................. 48
Fig 4. 24 Network Formation Time ............................................................... 49
Fig 4. 25 Control Traffic Overhead ............................................................... 49
Fig 4. 26 Energy Consumption ................................................................. 50
Fig 4. 27 Packet Delivery Ratio ................................................................. 51

Fig 5. 1 Trickle-FS with K=2 ......................................................................... 54
Fig 5. 2 Energy Consumption .................................................................. 55
Fig 5. 3 Energy Consumption .................................................................. 55
Fig 5. 4 Packet Delivery Ratio of DIO interval Min ....................................... 56
Fig 5. 5 Packet Delivery Ratio with K .......................................................... 56
Chapter ONE

Introduction

The term The Internet of Things (IoT) is commonly used to name a set of objects (or things) that are directly connected to the Internet using the Internet Protocol (IP) stack. That is the main difference of wireless sensor networks (WSN) of previous generation where nodes were organized in a local network with special protocols like ZigBee[22] or WirelessHART[23]. Connection of objects to the global network in the IoT opens the opportunity for global data analysis. Typical applications for the IoT are home automation (e.g. smart home), personal health monitoring (e.g. measurements of heart rate, pulse or temperature), building automation (e.g. control heating, electrical and ventilation systems of the building), industrial automation (e.g. control of the electrical grids) and smart cities.

Routing is an essential service in the IoT, since it enables the exchange of information between Things, by efficiently directing and reliably delivering data on the network from their sources to their destinations. However, routing in the IoT is also challenging, due to the global scale of the IoT, the massive number of Things in the IoT, the dynamic topology of the IoT, and the resource constraints of the IoT devices.

The Internet Engineering Task Force (IETF) quickly recognized the need to form a new Working Group (WG) to standardize an IPv6-based routing solution for IP smart object networks, which led to the formation of a new Working Group called ROLL (Routing Over Low power and Lossy) networks in 2008: [20].

The ROLL Working Group conducted a detailed analysis of the routing requirements focusing on several applications: urban networks including smart grid, industrial automation, home, and building automation. This set of applications has been recognized to be sufficiently wide to cover most of the applications of the Internet of Things. The objective of the WG was to design a routing protocol for LLNs, supporting a variety of link layers, sharing the common characteristics of being low bandwidth, lossy and low power. The result of this Working Group was the “Ripple” routing protocol (RPL) specification, along with supporting specifications on routing metrics, objective functions and security [18].

Note that RPL operates at the IP layer according to the IP architecture, and thus allows for routing across multiple types of link layers, in contrast with other form of “routing” operating at lower layer (e.g. link layers).

Low-power and Lossy Networks (LLNs) are a class of network in which both the routes and their interconnect are constrained. LLN routers typically operate with constraints on processing power, memory, and energy (battery power). Their interconnect are characterized by high loss rates, low data rates, and instability. LLNs are comprised of anything from a few dozen to thousands of routers. Supported traffic flows include point-to-point (between devices inside the
LLN), point-to-multipoint (from a central control point to a subset of devices inside the LLN), and multipoint-to-point (from devices inside the LLN towards a central control point). The trickle algorithm allows nodes in a LLN to exchange information in a highly robust, energy efficient, simple, and scalable manner. Trickle uses two mechanisms to achieve efficient routing: (1) rate adaption where the nodes send control traffic less often when the network is stable, and (2) suppression where a node avoids sending control traffic if the information has already been recently sent by neighboring nodes.

In this work an extensive performance of the protocol is analyzed considering different Trickle parameters in order to capture their impact on Network Formation Time, Energy Consumption, Control Traffic Overhead, and Packet Delivery Ratio for three different trickle algorithms i.e. Trickle, Trickle with Synchronizing Timer intervals (Trickle-S), and Trickle with fair broadcast suppression (Trickle-F). And the results highlighted that the non-deterministic nature of Trickle can lead to sub-optimal route formation and high energy consumption. Trickle-F mitigates from sub-optimal route formation with the same energy consumption of Trickle. Trickle-S forms a better route with less power consumption than Trickle and Trickle-F, and mitigates from forming a sub-optimal route. Trickle-Fair broadcast suppression and Synchronizing Trickle intervals (Trickle-FS) is demonstrated to be effective in obtaining more efficient routes with the same power consumption of Trickle-S, this proposed solution is a new with respect to the existing literature.

The rest of the thesis is organized as follows: Chapter 2 presents the state of the art concerning the Routing Protocol for Lowpower and Lossy Networks; the Trickle algorithms are described in detail. Chapter 3 regards the software’s used to conduct the simulations. Chapter 4 describes the Evaluation of the Trickle algorithms and analyzed their effects on the performance of RPL. Chapter 5 describes the new proposed algorithm and the evaluation on the performance of RPL. Chapter 6 describes the Discussion, and Chapter 7 describes the Conclusion and Future work.
Chapter TWO

Background

2.1 RPL

A routing protocol is responsible for forwarding the packets from nodes. A proactive (Table Driven) is one of the routing protocols in WSN; in this approach the routes are computed in advance and stored in routing tables. Proactive protocols periodically exchanges control messages to find and propagate the routes in the network as soon as they start. Nodes send both local control messages to share local neighborhood information; and messages across the entire network for sharing the topology related information among all the nodes in the network.

Routing protocol for low power and lossy networks (RPL) is IPv6 routing protocol for low power and lossy networks designed by IETF routing over low power and lossy network (ROLL) group [16] as a proposed standard. RPL is distance vector protocol because linked state protocols require a significant amount of memory (links state database, LSDB) which is not suitable for the resource constrained LLNs. RPL is a proactive routing protocols and start finding the routes as soon as the RPL network is initialized.

RPL forms a tree like topology also called Directed Acyclic Graph (DAG). Each node in a RPL network has a preferred parent which acts like a gateway for that node. If a node does not have an entry in its routing table for a packet, the node simply forwards it to its preferred parent and so on until it either reaches the destination or a common parent which forwards it down the tree towards the destination. The nodes in a RPL network have routes for all the nodes down the tree. It means the nodes nearer to the root node have larger routing tables.

Path selection is an important factor for RPL and unlike traditional networks routing protocols, RPL uses more factors while computing best paths for example routing metrics, objective functions and routing constraints. These factors are described in detail in the following sections.

RPL uses TCP/IP for communication as it has become a global standard after the success of internet. IP is used in LLN to provide end to end connectivity. It also solves the problem of interoperability between these devices from different vendors [10], [13]. It also facilitates the development of applications and integrations in terms of data collection and configuration [11]. Experience has shown that IP is both lightweight enough to run on even severely resource constrained systems [7].

In the following sections we describe RPL topology, routing mechanism, RPL control messages, routing metrics, constraints, Objective functions, Trickle timer and Trickle algorithms.
2.2 Topology Formation

LLNs do not typically have predefined topologies, for example those imposed by point to point wires, so RPL has to discover links to form a topology first.

![RPL Network Topology](image)

Figure 2. 1 RPL Network Topology

RPL creates a tree like topology with a root at the top and leaves at the edges as shown in Figure 2.1. RPL uses "up" and "down" direction’s terminology regarding the movement of traffic. The up is from the leaves to the root and down from the root to the leaves.

2.3 Routing in RPL

RPL is a distance vector protocol and the operation of the protocol can be divided into two main phases.

2.3.1. Routing upward

RPL provides routes up the Destination Oriented DAG (DODAG) towards the DODAG Root, constructing a DODAG optimized according to an Objective Function. This is accomplished by sending DODAG Information Object (DIO) messages from the DODAG Roots down the DODAG, towards the leaf nodes. With the help of these messages the DODAGs are formed and maintained. A node is able to join a DODAG by discovering neighbours that are already part of the DODAG of interest and by having built a parent set. The parent set represents a subset of the candidate neighbour set (which contains the nodes reachable via link-local multicast). Any neighbour with a lower rank than the node itself is considered as a candidate parent. From the parent set, a preferred parent is elected which will be used as the next hop for all upward routes.
This allows for Multipoint-to-Point communication within a WSN that runs RPL. Regarding the Upward routing process, the construction of the DODAG is of great importance. The process of the DODAG construction is based on the Distributed Algorithm Operation and it involves assigning / configuring some nodes as DODAG Roots. These nodes send DIO messages (link-local multicast) to all the RPL nodes with the intention of advertising their “presence, affiliation with a DODAG, routing cost and related metrics”[18]. The nodes always listen for DIO messages and they use the information provided by these messages to join a new DODAG or maintain the existing one. (“Nodes provision routing table entries, for the destinations specified by the DIO message, via their DODAG parents in the DODAG Version. Nodes that decide to join a DODAG can provision one or more DODAG parents as the next-hop for the default route and a number of other external routes for the associated instance.”[18]).

### 2.3.2 Downward routing

RPL can also provide routes down the DODAG in order to reach certain nodes (that are not DODAG Roots). This is accomplished by sending DODAG Destination Advertisement Object (DAO) messages up the DODAG towards the DODAG Root. Usually downward routes are Point-to-Multipoint routes (from a DODAG Root towards leaf nodes), but with the help of the DAO messages, Point-to-Point routing can also be supported. (“Point-to-Point messages can flow toward a DODAG Root (or a common ancestor) through an upward route, then away from the DODAG Root to a destination through a downward route.”).

The downward routes can be maintained in two modes, called “storing” and “non-storing”. In “storing” mode, all non-root and non-leaf nodes store a routing table for the downward routes for their sub-DODAG. The routing tables are formed based on the information provided by the DAO messages. When a message is travelling on a downward route in “storing” mode, routing decisions (determine the packet’s next hop) are taken at every hop (based on each hops’ stored routing table).

In “non-storing” mode, the non-root nodes do not have to store a routing table. Only the DODAG Root has to store and maintain a routing table, thus making it responsible for all downward routing decisions. The messages are routed down the DODAG based on source routes provided by the DODAG Root. This mode is highly dependent on the DODAG Root node (if the Root node fails to store some information, then some of the destinations may not be reached). Point-to-Point routing can be supported by both “storing” and “non-storing” modes.

### 2.4 RPL Messages

RPL uses three types of control messages for creating and maintaining RPL topology and routing table.
These messages are: DODAG Information Object (DIO), DODAG Information Solicitation (DIS) and DODAG Destination Advertisement Object (DAO).

### 2.4.1 DODAG Information Object (DIO)
DIO messages are used by RPL to form, maintain and discover the DODAG. When a RPL network starts, the nodes start exchanging the information about the DODAG using DIO messages which contains information about the DODAG configuration and help the nodes to join the DODAG and select parents.

### 2.4.2 DODAG Information Solicitations (DIS)
The DIS is used by any node to explicitly solicit the DIO messages from the neighbor nodes. It is triggered by the node in case when it could not receive a DIO after a predefined time interval.

### 2.4.3 DODAG Destination Advertisement Object (DAO)
The DAO messages are used by RPL to propagate a node prefix to the ancestor nodes in support of downward traffic.

### 2.5 Routing metrics

A routing metric is a quantitative value used to find the cost of a path and helps in making the routing decision in case there are different routes available. In LLN a metric is a scalar used to find the best path according to the objective function.

Unlike traditional networks LLN also use node metric apart from link metrics. Therefore the metrics can be categorized as node metric and link metrics as stated below [9].

**Node metrics:** Node State Attribute (NSA), Node Energy, Hop count

**Link metrics:** Throughput, Latency, Link Quality Level, ETX, Link Color

**Hop Count:** This metric counts the number of hops from the source to the destination. A hop count of 3 means there are 3 intermediate links between the source and destination.

**Expected Transmission Count (ETX):** ETX of a link is the expected number of transmissions required to send a packet over that link. The path ETX is the sum of the ETX of all the links along the path. The ETX of a path with 3 links of 100% delivery ratio is 3, whereas the ETX of a path with 2 links of 50% delivery ratio is 4.

Since low power networks have vastly varying requirements and characteristics like lossy links, resources constraints, mobility, a vast variety of applications, therefore RPL does not define any specific metrics or forwarding polices [16] and these are described in other IETF Drafts.
Routing metrics are a critical component to the routing strategy. LLN has a wide variety applications and constraints which strongly appeal for dynamic metrics.

To better understand the need of dynamic metrics and difference between a metric and constraint for LLN, let's consider the following examples.

1. An application requires a quick delivery of packets using a short path and therefore the goal will be to use ETX metric for routing.

2. An application may require encrypted communication and therefore the goal will be to avoid non-encrypted links in the path.

3. A node may be energy constrained and the objective will be to minimize Energy Consumption by using as many mains connected nodes along the path as possible.

In the first example the ETX is a routing metric. In the second example in which the goal is to avoid non-encrypted links; can be considered as a constraint which is explained next. In the third example energy can be either a constraint in which case the path will not contain any battery operated node or it can be a metric in which case the path may contain the minimum number of battery operated nodes as compared to the alternate paths.

2.6 Routing Constraint

A constraint is used to either include or exclude links from the routing path that do not meet the criteria specified in the objective function.

2.7 Objective Function

A RPL Objective Function (OF) states the outcome of the process used by a RPL node to select and optimize routes within a RPL instance based on the Information Objects available. As a general concept, an OF is not an algorithm.

RPL forms Directed Acyclic Graphs (DAGs) as collections of Destination-Oriented DAGs (DODAGs) within instances of the protocol. Each instance is associated with a specialized Objective Function. A DODAG is periodically reconstructed as a new DODAG version to enable a global reoptimization of the graph.

An instance of RPL running on a device uses an Objective Function to help it determine which DODAG and which Version of that DODAG it should join. The OF is also used by the RPL Instance to select a number of routers within the DODAG current and subsequent versions to serve as parents or as feasible successors.

The RPL Instance uses the OF to compute a Rank for the device. This value represents an abstract distance to the root of the DODAG within the DODAG Version. The Rank is exchanged
between nodes using RPL and allows other RPL nodes to avoid loops and verify forward progression toward the destination. Regardless of the particular OF used by a node, Rank will always increase; thus, post convergence, loop-free paths are always formed.

2.8 Trickle Algorithm

The Trickle algorithm established a density-aware local communication primitive with an underlying consistency model that guides when a node transmits. When a node’s data does not agree with its neighbors, that node communicates quickly to resolve the inconsistency (e.g., in milliseconds). When nodes agree, they slow their communication rate exponentially, such that nodes send packets very infrequently (e.g., a few packets per hour). Instead of flooding a network with packets, the algorithm controls the send rate so each node hears a small trickle of packets, just enough to stay consistent. Furthermore, by relying only on local communication (e.g., broadcast or local multicast), Trickle handles network re-population; is robust to network transience, loss, and disconnection; is simple to implement; and requires very little state.

While Trickle was originally designed for reprogramming protocols (where the data is the code of the program being updated), experience has shown it to be a powerful mechanism that can be applied to a wide range of protocol design problems, including control traffic timing, multicast propagation, and route discovery.

In the next section we explained the Trickle algorithms Trickle, Trickle with Synchronizing Trickle intervals from now onwards Trickle-S, and Trickle with fair broadcast suppression from now onwards Trickle-F.

2.8.1 Trickle Algorithm Overview

Trickle’s basic primitive is simple: every so often, a node transmits data unless it hears a few other transmissions whose data suggest its own transmission is redundant. Example of such data includes routing state, software update versions, and the last heard multicast packet. This primitive allows Trickle to scale thousand-fold variations in network density, quickly propagate updates, distribute transmission load evenly, be robust to transient disconnections, handle network re-populations, and impose a very low maintenance overhead.

Trickle sends all messages to a local communication address. There are two possible results to a Trickle message: either every node that hears the message finds that the message data is consistent with its own state, or a recipient detects an inconsistency. Detection can be the result of either an out-of-date node hearing something new, or an updated node hearing something old. As long as every node communicates somehow—either receives or transmits—some node will detect the need for an update.
For example, consider a simple case where “up to date” is defined by version numbers (e.g., network configuration). If node A transmits that it has version X, but B has version X+1, then B knows that A needs an update. Similarly, if B transmits that it has version X+1, A knows that it needs an update. If B broadcasts or multicasts updates, then all of its neighbors can receive them without having even heard A’s transmission. In this example, it does not matter who first transmits — A or B; the inconsistency will be detected in either case.

The fact that Trickle communication can be either transmission or reception enables the Trickle algorithm to operate in sparse as well as dense networks. A single, disconnected node must transmit at the Trickle communication rate. In a lossless, single-hop network of size n, the Trickle communication rate at each node equals the sum of the Trickle transmission rates across all nodes. The Trickle algorithm balances the load in such a scenario, as each node’s Trickle transmission rate is 1/nth of the Trickle communication rate. Sparser networks require more transmissions per node, but the utilization of a given broadcast domain (e.g., radio channel over space, shared medium) will not increase. This is an important property in wireless networks and other shared media, where the channel is a valuable shared resource. Additionally, reducing transmissions in dense networks conserves system energy.

### 2.8.2 Trickle Parameters and Variables

A Trickle timer runs for a defined interval and has three configuration parameters: the minimum interval size $I_{\text{min}}$, the maximum interval size $I_{\text{max}}$, and a redundancy constant $k$:

**$I_{\text{min}}$**

This parameter gives the minimum amount of time between two DIOs. DIOs are transmitted periodically to reduce the redundant Control Traffic and use the limited resources more optimally. The transmission of DIO is controlled by a timer called trickle timer whose minimum value is $I_{\text{min}}$ and maximum value is $I_{\text{max}}$. The value of trickle timer starts from the lowest possible value $I_{\text{min}}$ and is doubled each time it is transmitted until it reaches its maximum possible value of $I_{\text{max}}$.

**$I_{\text{max}}$**

This parameter is used to limit the number of times the $I_{\text{min}}$ can be doubled.

**Redundancy constant ($k$)**

It is a natural number greater than 0 and is used to suppress the DIO transmission.

In addition to these three parameters, Trickle maintains three variables:

I, the current interval size,
t, a time within the current interval, and
c, a counter

Figure 2. 2 Trickle Algorithm

2.8.3 TRICKLE Algorithm

Algorithm Description

When the algorithm starts execution, it sets I to a value in the range of \([I_{min}, I_{max}]\) -- that is, greater than or equal to \(I_{min}\) and less than or equal to \(I_{max}\). The algorithm then begins the first interval.

When an interval begins, Trickle resets c to 0 and sets t to a random point in the interval, taken from the range \([I/2, I)\), that is, values greater than or equal to \(I/2\) and less than \(I\). The interval ends at \(I\).

Whenever Trickle hears a transmission that is "consistent", it increments the counter c.

At time t, Trickle transmits if and only if the counter c is less than the redundancy constant k.

When the interval I expires, Trickle doubles the interval length. If this new interval length would be longer than the time specified by \(I_{max}\), Trickle sets the interval length I to be the time specified by \(I_{max}\).

If Trickle hears a transmission that is "inconsistent" and I is greater than \(I_{min}\), it resets the Trickle timer. To reset the timer, Trickle sets I to \(I_{min}\) and starts a new interval as in step 2. If I is equal to \(I_{min}\) when Trickle hears an "inconsistent" transmission, Trickle does nothing. Trickle can also reset its timer in response to external "events".
RPL specifies the conditions determining inconsistent events which cause the reset of the Trickle timer. In particular, a reset is mandatory when the DODAG version number of a DIO message is newer than the current one, i.e., when a global repair procedure has been triggered, or when a loop in data forwarding is detected.

Fig 2.2 illustrates the Trickle operation with DIO transmission suppression in the case of four nodes and \( k = 1 \). Node N1 generates the shortest time \( t \) so it transmits its DIO and increments the counter \( C \) in the first interval and the other nodes N2, N3, N4 suppressed their transmission because the redundancy threshold \( k \) is not greater than counter \( c \), in the second interval Node N3 generates the shortest time \( t \) so it transmits its DIO and increments the counter \( C \) in the second interval and the other nodes N1, N2, and N4 suppressed their transmission because the redundancy threshold \( k \) is not greater than counter \( c \).

Pseudo-code of Trickle algorithm

**Algorithm 1: Trickle**

1. **Function initialization()**
   
   \[ I \leftarrow I_{\text{min}} \]

2. **Function Begin()**
   
   \[ T \leftarrow I \]
   \[ C \leftarrow 0 \]
   \[ t \leftarrow [T/2, T] \]

3. **Function consistent transmission Received()**
   
   \[ C \leftarrow C + 1 \]

4. **Function TimerExpire()**
   
   *If \( k > C \) then*
   
   Transmit DIO
   
   *Else suppress DIO*
   
   *End if*

5. **Function interval ends()**
   
   \[ C \leftarrow 0; \]

   *If inconsistent transmission received then*
   
   \( I \leftarrow I_{\text{min}} \)

   *Else \( I \leftarrow I \times 2 \)*

   *If \( I_{\text{max}} \leftarrow I \) then*

   \( I \leftarrow I_{\text{max}} \)

   *End if*

   *End if*
2.9 Trickle-S

The desynchronization in Trickle can be eliminated by adding a synchronization mechanism that resynchronizes the time intervals in Trickle. This mechanism makes all nodes execute their Trickle listen and transmission intervals at the same time.

A node is classified as synchronize with other nodes iff:

Every time consistent DIO is received each node performs the following:

\[ \partial = T - J/2 \]

\[ \begin{aligned} & \text{If } \partial < 0 \text{ unsynchronized} \\ & \text{Else synchronized} \end{aligned} \]

Where T is the listen only period, and J is the Trickle interval

Algorithm 2: Trickle with resynchronization of intervals

1. Function initialization()
   \[ T \leftarrow I_{\text{min}} \]

2. Function Begin()
   \[ T_s \leftarrow T - \partial_s \]
   \[ C \leftarrow 0 \]
   \[ \partial_s \leftarrow 0 \]
   \[ t \leftarrow [Ts/2, Ts] \]

3. Function consistent transmission Received()
   \[ C \leftarrow C + 1 \]
   \[ \text{If } T \in [T/4, T/2] \]
   \[ \partial_s \leftarrow Ws (T-T/2) \]
   \[ \text{End if} \]

4. Function TimerExpire()
   \[ \text{If } k \geq C \text{ then} \]
   \[ \text{Transmit DIO} \]
   \[ \text{End if} \]

5. Function interval ends()
   \[ C \leftarrow 0; \]
   \[ \text{If in consistent transmission received then} \]
   \[ I \leftarrow I_{\text{min}} \]
Else $I \leftarrow Ix2$

If $Imax \leftarrow I$ then

$I \leftarrow Imax$

End if

End if

K = 1

| DIO Transmission |
| DIO Suppressed |
| Reception |
| Listening Period |

Figure 2.3 Trickle-S

Fig. 2.3 illustrates the Trickle with synchronizing time intervals operation with DIO transmission suppression in the case of four nodes and k = 1. Node N1 generates the shortest time t so it transmits its DIO and increments the counter C in the first interval, while N1 transmits its DIO, nodes N2 and N4 were in the second half of their listening period and also they suppress their transmission in the first interval and starts their second interval earlier i.e., by reducing the value $T/2-\partial$s from the second interval. In the second interval Node N2 generates the shortest time t so it transmits its DIO and increment the counter C and the other nodes N1, N3, and N4 suppress their transmission because the redundancy threshold k is not greater than counter c.

2.10 Trickle-F

Trickle was originally designed as a gossiping algorithm: its original goal was to spread the same piece of information across a network rapidly with a minimum number of messages. Routing information updates, as those carried by DIO messages, are instead strictly dependent on the source of the message: suppressing one transmission or another is not always equivalent, since
the two suppressed messages carry different information. Should some node be not allowed to send any message for a long time, some routes may remain undiscovered and therefore unused for such a time even though they are better than those currently active in the DODAG. The original Trickle algorithm provides each node with equal average broadcast transmission probability in the long run. However, for routing purposes it is important that every node is given the opportunity to share its routing information in the shortest possible time scale, so as to allow the quick discovering of all available routes, and then choose the best ones according to the established routing metrics.

*Trickle*-F [19], which aims at guaranteeing a fair short-term broadcast suppression among nodes in a neighborhood in order to facilitate the rapid discovery of all available paths. The rationale behind Trickle-F is to prioritize each node strictly depending on the number of consecutive suppressions: the longer the time spent by a node without transmitting, the higher its transmission priority in the next round. In order to achieve this, Trickle-F introduces a modification to the original algorithm in the computation of the start time and length of the next transmission period.

In order to provide broadcast fairness, each node keeps track of s, the number of continuous communication intervals in which a message transmission has been suppressed. At time t, if a DIO is transmitted, s is reset; otherwise, the counter is incremented. Each node gets a transmission priority proportional to the number of last consecutive suppressed transmissions. Priority dependence on s is enforced by modifying the length of the listening and transmitting intervals. Each transmission period T is not halved as in the original algorithm: the listening and transmitting periods are set to a variable length which is proportional to the number of suppressed transmissions, $\frac{T}{2^s}$. At the beginning of each period, the transmission instant t is selected in a sub-period depending on s as follow: $[\frac{T}{2^{s+1}}, \frac{T}{2^s}]$. This ensures strict prioritization of the sub-periods according to s: the larger s is, the closer the sub-period is. This modification guarantees that nodes that have waited longer get higher transmission probability, while nodes that have been suppressing the same number of transmissions will have the same transmission probability. It is important to highlight that each node still has a listening.

![Figure 2. 4 Trickle-F with Four nodes and K=2](image-url)
sub-period equal to \( \frac{T}{2^{s+1}} \): this allows to overcome the short-listen problem as in the original design [13]. Fig. 14 illustrates an example of Trickle-F operation in the same scenario as that in Fig. 2 in the case of four nodes and \( k = 2 \). This time, nodes 1 and 2 schedule DIO transmission in a shorter next transmission period (as a matter of fact, during the listening interval of nodes 3 and 4) and therefore get priority over nodes 3 and 4, which transmitted in the previous period.

Algorithm 3 Trickle-F

function INITIALIZE()
\[ T \leftarrow I_{\text{min}} \]
\[ s \leftarrow 0 \]

function INTERVALBEGINS()
\[ c \leftarrow 0 \]
\[ t \leftarrow \text{random} \left( \frac{T}{2^{s+1}}, \frac{T}{2^s} \right) \]

function CONSISTENTTRANSMISSIONRECEIVED()
\[ c \leftarrow c + 1 \]

function TIMEREXPIRES()
if \( k \geq c \) then
    Transmit DIO
    \[ S \leftarrow 0 \]
else
    \[ s \leftarrow s + 1 \]
end if

function INTERVALENDS()
\[ c \leftarrow 0 \]
if InconsistentTransmissionReceived then
\[ I \leftarrow I_{\text{min}} \]
\[ s \leftarrow 0 \]
else
\[ I \leftarrow I \times 2 \]
if Imax \( \leq I \) then
\[ I \leftarrow I_{\text{max}} \]
end if
end if
Chapter THREE

Methods

A wireless sensor network also called Low power and lossy network (LLN) is a class of networks consisting of devices with a communications infrastructure intended to monitor physical or environmental conditions at diverse locations. Commonly monitored parameters are temperature, humidity, pressure, power-line voltage, and vital body functions etc. The devices in a sensor network called sensor nodes are equipped with a transducer, microcomputer, transceiver and power source. The transducer generates electrical signals based on sensed physical effects and phenomena. The microcomputer processes and stores the sensor output. The transceiver transmits and receives radio signals, and power source provides electricity to these devices.

The size of these devices is usually very small and powered by either battery, energy scavenging like solar cells or mains powered. The size factor of the devices also makes them resource constraint and therefore they need a very sensible use of the resources. LLNs are aimed for low traffic applications. Low traffic is very common for smart home applications, ubiquitous computing where a very long life is required.

Since the devices have limited range of transmission, therefore Routing is required in these devices to reach each other. Routing is responsible for managing the routes among sensor nodes and forwarding the packets on the most efficient route discovered. To send or receive these packets the nodes use the transceiver part which is a short range radio.

The radio medium used by LLN devices is of short range and also very susceptible to bit errors.

The lossy nature of LLN has a strong impact on the routing protocol design. Since the link failures are frequent and usually transient, therefore the routing protocol should not overreact in an attempt to converge the network as a result of temporary failures.

Due to these reasons, one of the challenging issues in wireless sensor networks is finding the best routes for the delivery of data, which implies a very efficient routing mechanism for finding and keeping the routes in the network. The routing mechanism is subjected to both the resource constraint nature of sensor nodes and lossy nature of the radio medium in LLN.

In this study, we used Cooja [2] simulator under the Contiki OS [8] to investigate and evaluate the algorithms i.e Trickle, Trickle-F, and Trickle-S and their impact on the RPL performance.
3.1 Contiki Operating System

Contiki is a wireless sensor network operating system and consists of the kernel, libraries, the program loader, and a set of processes [8]. It is used in networked embedded systems and smart objects.

Contiki provides mechanisms that assist in programming the smart object applications. It provides libraries for memory allocation, linked list manipulation and communication abstractions. It is the first operating system that provided IP communication. It is developed in C, all its applications are also developed in C programming language, and therefore it is highly portable to different architectures like Texas Instruments MSP430, and currently Contiki 2.7 to ST microelectronics STM32 families.

Contiki is an event-driven system in which processes are implemented as event handlers that run to completion. A Contiki system is partitioned into two parts: the core and the loaded programs. The core consists of the Contiki kernel, the program loader, the language run-time, and a communication stack with device drivers for the communication hardware [8].

The Program loader loads the programs into the memory and it can either obtain it from a host using communication stack or can obtain from the attached storage device such as EEPROM.

RPL is composed of several source files in Contiki. To implement the algorithms for Trickle i.e Trickle-S, Trickle-F, and Trickle-FS we modify the files rpl-timers.c, rpl.h, rpl-dag.c, rpl-conf.h. The file rpl-timers.c is the one for controlling the Trickle timers. In order to apply the algorithms we modify the file as can be shown in appendix A.

3.2 Cooja Simulator

Cooja is a Java-based simulator designed for simulating sensor networks running the Contiki sensor network operating system [2]. The simulator is implemented in Java but allows sensor node software to be written in C.

One of the differentiating features is that Cooja allows for simultaneous simulations at three different levels: Network Level, Operating System Level and Machine code instruction level [2]. Cooja can also run Contiki programs either compiled natively on the host CPU or compiled for device emulator.

In Cooja all the interactions with the simulated nodes are performed via plugins like Simulation Visualizer, Timeline, and Radio logger. It stores the simulation in an xml file with extension 'csc' (Cooja simulation Configuration). This file contains information about the simulation environment, plugins, the nodes and its positions, random seed and radio medium etc.
Cooja Simulator runs the Contiki applications whose files are placed in another directory and may also contain a “project-conf.h” file which provides the ability to change RPL parameters in one place.

Cooja can be used with or without the GUI, to run a lot of simulation the GUI is slow so it is recommended using simulation without GUI. In our simulation we used the simulation without GUI. The simulation script tool allows writing the specific output to be printed in a log file and make analysis using matlab and excel. For instance to calculate the number of RPL control messages by the nodes we write the following script in Cooja simulator. Each time the specified message is printed by the node it is save in the log file.

```cpp
TIMEOUT(36000000);
//log.log("first simulation message at time : " + time + ";n");
while (true) {
    if(msg.equals("Received an RPL control message"))
        log.log(time + ":" + id + ":" + msg + ";n");
    YIELD(); /* wait for another mote output */
}
```

The simulation conducted in this thesis is not only involved deep analysis of the protocol behavior but repetition of simulations for different values of the RPL parameters and configurations. Running a series of these simulations using Cooja simulator is a time consuming and a mundane activity. The analysis of the effect of different parameters on RPL performance need a large set of data and take significant amount of time to get such data from the simulations to perform reliable and statistically correct results. Due to this reason, the process of executing simulations is automated.

Figure 3.1 depicts the simulation and analysis process of the thesis. The process starts with the selection of the parameter, if it is for Phase 1 or Phase 2 and a Cooja Jar file is created iteratively but if the parameter selected is DIO Interval Minimum or others the script changes the values of the parameter in the project-conf.h file of the Contiki application. The output of both cases yields in Cooja jar files in a separate directory. All the jar files in the directory are executed by the script and COOJA.testlog files are maintained in the respective directories for computing the performance metrics iteratively. The performance metrics are written to an analysis files in text format. The script uses the information in the analysis files and draws the diagrams using Microsoft excel and matlab.
Fig 3. 1 Simulation Configuration
Chapter FOUR

Evaluation

The simulation for this study is performed in two phases to evaluate the performance of RPL for the Trickle algorithms i.e., Trickle, Trickle-S, and Trickle-F. In the first phase a random topology of 100 client nodes and 1 server node which acts as DODAG root that is placed at the top of the client nodes is performed and evaluated based on four performance metrics of interest: Network Formation Time, Control Traffic Overhead, Energy Consumption, and Packet Delivery Ratio, by configuring the RPL parameters (DIO minimum interval, DIO Doublings, and Redundancy Threshold). In the second phase of simulation a random topology of 500 client nodes and 1 server node which act as a DODAG root is performed, and evaluated to see the effects for RPL performance by configuring the RPL parameters (DIO minimum interval, DIO Doublings, and Redundancy Threshold).

A number of protocol parameters influence the efficiency of RPL; the three significant among them are DIO minimum interval, DIO Doublings, and Redundancy Threshold. To observe the effect of these parameters on RPL performance we use four performance metrics of interest: Network Formation Time, Control Traffic Overhead, Energy Consumption, and Packet Delivery Ratio (PDR).

In the rest of the chapter we describe the performance metrics of interest in section 4.1, that we will use to observe the performance of RPL after changing the parameters. The parameters that impact RPL performance are described in section 4.2. In section 4.3 we describe the Link failure model which is used to simulate the lossy radio medium in Cooja. We perform the simulations in phase 1 and phase 2 in sections 4.5 and 4.6 respectively.

4.1 Performance metrics

We use four performance metrics namely Network Formation Time, Control Traffic Overhead, Energy Consumption, and Packet Delivery Ratio to evaluate the performance of RPL.

The first performance metric of interest is Network formation time. The nodes in the sensor network need to form a topology in order to communicate. Therefore Network formation time is crucial metric that need to be evaluated for any routing protocol. The Network formation time of the RPL DAG is defined as the amount of time needed by all the reachable (in terms of radio) nodes in the network to join the DAG.

The second performance metric is Control Traffic Overhead for the network. This includes DIO, DIS and DAO messages generated by each node and it is imperative to confine the Control...
Traffic keeping in mind the scarce resources in LLN. RPL control the redundant control messages by making use of trickle timers. The aim of this metric is to analyze the effect of the stated parameters on the Control Traffic overhead.

The third performance metric is Energy Consumption. To make good energy estimation we use percent radio on time of the radio which dominates the power usage in sensor nodes [12]. Furthermore we take the average percent radio on time for all the nodes in the whole network setup.

The fourth performance metric is Packet Delivery Ratio (PDR) and is defined as the number of received packets at the sink to the number of sent packets to sink. We take the average PDR of all the packets received successfully at sink.

In the following sections we observe one-by-one the effect of RPL parameters namely: DIO minimum interval, DIO Doublings, and Redundancy Threshold on the performance metrics of interest in a RPL network with lossy environment.

### 4.2 RPL parameters

Trickle algorithm is used to limit the number of control packets sent, the algorithm uses three important parameters namely, DIO Interval Minimum, DIO interval doubling, and Redundancy Threshold.

#### 4.2.1 DIO Interval Minimum

This parameter controls the rate of DIO transmission and therefore crucial for Network formation Time, Energy Consumption, Packet Delivery Ratio, etc. The more quickly the DIOs are transmitted the more quickly the network gets converged but at the expense of Energy Consumption, etc. This parameter is influential on the performance of the whole protocol performance. A careful tweaking of this parameter is necessary for improved performance keeping in view the difference application areas of WSN and environmental conditions.

#### 4.2.2 DIO Interval Doublings

This parameter defines the number of times the DIO minimum interval can be doubled and is useful to keep the traffic low for steady network conditions. It’s essential to configure this parameter precisely.

DIO Doubling is defined as:

\[
I_{\text{max}} = I_{\text{min}} \times 2^n 
\]

Where \( n \) represents DIO Doublings and \( I_{\text{min}} \) is determined from DIO interval minimum while \( I_{\text{max}} \) is computed as follows:
Suppose DIO interval minimum is 12 and DIO interval doubling is 4, then Imin and Imax can be calculated as:

\[
I_{\text{min}} = 2^{\text{DIO interval minimum}} \\
I_{\text{max}} = I_{\text{min}} \times 2^{\text{DIO Doublings}}
\]

\[
I_{\text{min}} = 2^{12} \\
I_{\text{max}} = 4096 \times 2^4 \\
I_{\text{min}} = 4.096s \\
I_{\text{max}} = 65.536s
\]

These two values mean that transmission will start at the rate of 4.096s (Imin) and then it can be doubled 4 times (DIO interval doubling or n) before reaching 65.536s (Imax) after which the rest of the transmission will take place at the rate of 65.536s.

### 4.2.3 Redundancy Threshold

It is a natural number greater than 0 and is used to suppress the DIO transmission.

### 4.3 Simulation and Network Setup

In order to conduct a WSN simulation the simulation of losses in wireless medium is very important because it simulates the actual environment where the sensor nodes will work. The more accurate the simulation of the radio medium the more closer is the results to actual radio medium. In this section we first describe how we simulate the radio medium in Cooja followed by the network setup used for the simulations in this study. After this we explain how we compute the performance metrics from the data that we obtain from these simulations.

#### 4.3.1 Link Failure Model

The link failure model is emulated by using Unit Disk Graph Model (UDGM) in Cooja [2]. It uses two different range parameters one for transmission and one for interference with other radios as shown in Fig 4.1
In Fig 4.1 the radio medium is simulated by UDGM in the simulations using Cooja. The bigger green circle denotes the transmission range (R) of node 1 while the gray circle denotes its collision with other radios. The figure as percentage shows the reception ratio of the transmission between node 1 and 2. Node 3 is inside the collision range of node 1; however it is in the transmission range of node 2, so they communicate in a multi-hop fashion.

### 4.3.2 Network Setup

Our network is setup in two phases. In both phases the network is composed of client-server application. The server is running a sample application udp-server.c while the all other nodes are using udp-client.c. We use a Cooja plugin called Contiki Test Editor to measure the simulation time and stop the simulation after an hour. This plugin also creates a log file (COOJA.testlog) for all the outputs from the simulation which we will analyze at the end of the simulation using a shell script and matlab.

In order to introduce lossyness in wireless medium we use the Cooja Unit Disk Graph Medium which introduces lossyness with respect to relative distance of nodes in the Radio Medium as discussed in section 4.3.1.

### 4.3.2.1 Network Setup Phase 1

In the first set of simulation we design a sample network in the Cooja simulator containing a random topology of 100 client nodes and 1 server node acting as root of the DODAG. This
scenario is simulated for 100 replications in order to have statistically good results, and the average value of each metric with its 95% confidence interval is reported. The network setup is shown in Figure 4.2

Fig 4. 2 A network of 100 nodes

Fig 4.2: shows a network composed of 100 client nodes and 1 server node act as DODAG root. i.e., the blue-green node is the server and all other with yellow are the client nodes.

4.3.2.2 Network Setup Phase 2

In the second set of simulation we design a sample network in the Cooja simulator containing a random topology of 500 client nodes and 1 server node acting as root of the DODAG. This scenario is simulated for 100 replications in order to have statistically good results, and the average value of each metric with its 95% confidence interval is reported. The network setup is shown in Figure 4.3
Fig 4.3 A network of 500 nodes

Fig 4.3 shows a network composed of 500 client nodes and 1 server node acting as a DODAG root, i.e., the blue-green (as indicated by the arrow) node is the server and all other with yellow are the client nodes.

4.4 Measuring the performance metrics

In order to get the Network Formation Time in a RPL network we determine the time receiving at least one DIO in all nodes. The Network formation time is obtained by subtracting the first DIO sent time to the last joined DAG.

Network Formation Time = Last DIO joined DAG – First DIO sent

To compute the average Packet Delivery Ratio we measure the number of sent packets from all the nodes to the sink and divide it by the number of successfully received packets at the sink.

Average PDR = (Total packets Received / Total packets sent) * 100

To compute the power consumption we use the mechanism of Powertrace system available in Contiki [7] [3]. Powertrace is a system for network-level power profiling for low-power wireless networks which estimates the energy consumption for CPU processing, packet transmission and
listening. This mechanism maintains a table for the time duration of components like CPU, radio transmitter was on. Based on this computation we calculate the percentage of radio on time. We compute the power consumption for radio transmission and listening as these are the most energy consuming components [12].

RPL uses ICMPv6 based control messages called DIS (DODAG Information Solicitation) and DIO (DODAG Information Object) for building and maintaining DODAG. The ICMPv6 is a network layer protocol and therefore we are capturing the control messages from the network level as we are not interested in the application level messages in this experiment. And the Control Traffic Overhead is defined as:

\[
\text{Control Traffic Overhead} = \sum_{k=1}^{n} DIO \ (k) + \sum_{k=1}^{n} DIS \ (k) + \sum_{k=1}^{n} DAO \ (k) \quad \text{----------eq. 4}
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver Sensitivity</td>
<td>-94dBm</td>
</tr>
<tr>
<td>Channel Frequency</td>
<td>2.4GHz</td>
</tr>
<tr>
<td><strong>Control parameter</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>DIO Interval Minimum</td>
<td>3-14</td>
</tr>
<tr>
<td>DIO Doubling</td>
<td>3-14</td>
</tr>
<tr>
<td>Redundancy Threshold</td>
<td>1-12</td>
</tr>
<tr>
<td>Objective Function</td>
<td>ETX</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>60 minutes</td>
</tr>
<tr>
<td>Simulation Replication</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4.1: simulation parameters

**4.5 Phase1: Performance metrics evaluation**

This phase contains a random topology of 100 client nodes and 1 server node. An evaluation of the performance of RPL for the Trickle algorithms (Trickle, Trickle-S, and Trickle-F) on the RPL parameters (DIO Minimum Interval, DIO Doubling, and Redundancy Threshold) is performed. The simulation parameters used are described in table 4.1; and the results obtained are explained as follows;

**4.5.1 DIO Interval Minimum**

The objective of this experiment is to analyze the effect of DIO Minimum Interval on RPL performance metrics of interest.

The simulation parameters used in this simulation are shown in Table 4.1. We change the DIO Interval Minimum values from 3 to 14 in the subsequent iterations of the simulation, by keeping K to 10 and DIO Doublings to 8.

**A. Network formation Time**
Figure 4.4 illustrates the Network Formation Time i.e., the time needed for all the nodes to receive at least one DIO message. Trickle, Trickle-S, and Trickle-F have the same Network Formation Time. As can be seen in figure 4.4, the lower DIO interval minimum is, the sooner the DODAG is formed and the sooner all the nodes can send upward traffic.

As we increase the DIO Interval, the Network Formation Time also increases and for DIO Interval Minimum between 3 and 7 the Network Formation Time increases slightly, however from DIO Interval Minimum of 9 it increases linearly.

**B. RPL Control Overhead**
The RPL control messages are less in the case of Trickle, because of continues suppression of nodes i.e., nodes are unable to send DIO messages as discussed in section 1.3.5, which is a tradeoff for having the best path.

The Trickle-S has less Control messages than Trickle-F, because Trickle-F gives a priority to suppressed nodes, though the nodes may have already heard messages.

As can be shown in Figure 4.5 the Control Traffic Overhead is decreasing as we increase the DIO Interval Minimum. The control traffic overhead is very high for lower DIO Interval Minimum (Lower than 8) as shown in Fig 4.5, these lower values of DIO Interval Minimum enables the Trickle timer to fire the DIO very quickly.

C. Energy Consumption

To compute the Energy consumption by the network we use the Powertrace tool [3] [17]. This tool uses power state tracking to estimate the power consumption of the system for activities such as packet transmission and receptions.
Fig 4.6 Energy Consumption

In this experiment we only compute the power consumption by the transceiver part which is the most power consuming component [12]; the CPU power consumption is very low and can be ignored for simplicity reasons in this simulation.

Figure 4.6 illustrates the other side of the trade-off with respect to Network Formation Time. As can be seen, the shorter the DIO interval Minimum is, the higher the number of sent messages is, hence resulting in higher energy consumption per node.

Trickle-S has better energy consumption than Trickle and Trickle-F. In Trickle-S since all the nodes have synchronized intervals they couldn’t have propagation delay so the nodes have better load balancing. However in Trickle and Trickle-F there is propagation delay to send the DIO so they have higher energy consumption than Trickle-S.

The RPL network consumes a lot of energy about 6.7% Radio ON time when DIO interval is very small (i.e. 3) in Trickle, and Trickle-F where as in Trickle-S the energy consumption is 5.5 % as shown in the Figure 4.7. For DIO Interval Minimum of 4 to 7 the Energy Consumption keeps on decreasing in a linear fashion. However the best Energy Consumption can be observed for DIO Interval Minimum of 8 and higher where the Energy Consumption is about 2.5% and remains constant for all values of DIO Interval Minimum of 8 and higher. This means that optimum value for DIO Interval Minimum regarding Energy Consumption is between (8-16).

D. Packet Delivery Ratio

Packet Delivery Ratio is used by wireless sensor network to compute the best route, and optimum transmission rate. [6], [4].
As shown in Figure 4.7 the Packet Delivery Ratio is better in the case of Trickle-S than the Trickle, and Trickle-F, the reasons for having Trickle-S better PDR is as follows:

Because of high Control Traffic in the case of Trickle-F as shown in Figure 4.5 the Packets are suffered from collision and retransmission, though the best path could be found. But in the case of Trickle; since the loads are unbalanced there are paths that cannot be discovered so they cannot discover the best paths to deliver the packets. But in the case of Trickle-S all the nodes perform the listening and transmission timers are approximately aligned across all the nodes, so there are better paths to be discovered.

Another observation is that as the DIO interval increases the PDR increases, this is because of the Control Traffic Overhead; as shown in Figure 4.5 the control traffic overhead is decreasing as the DIO interval minimum increases so the Network is not suffering from traffic of RPL control messages.

Fig 4. 7 Packet Delivery Ratio

4.5.2 DIO Interval Doublings

The objective of this simulation is to evaluate the performance of RPL for different values of DIO Doublings. The network scenario explained in section 4.3.2 that contains 100 client nodes and 1 server node is used to perform this simulation. And the simulation parameters are shown in the table 4.1. However we set the DIO Doublings in the range from 3 to14, for each iteration of the simulation. We choose DIO Interval Minimum of 12 and Redundancy Threshold of 10.
A. Network Formation Time

After changing the DIO Doubling interval in the range 3 to 14, we observe that the network formation time is not affected and the network setup time is about 20 seconds in the algorithms Trickle, Trickle-S, and Trickle-F as shown in Figure 4.8.

Fig 4. 8 Network Formation Time

Network Formation Time is not depending on DIO Doubling i.e., changing DIO doubling doesn’t affect the Network Formation Time for all the algorithms.

B. Control Traffic Overhead
We observe that the RPL control messages are less in the case of Trickle, for the same reason discussed in section 4.5.1 B.

The Trickle-S has less Control messages than Trickle-F, because Trickle-F gives a priority to suppressed nodes, though the nodes may have already heard messages.

As can be shown in Figure 4.9 the Control Traffic Overhead is decreasing as we increase the DIO Doublings. The control traffic overhead is very high for lower DIO Interval Minimum (Lower than 6) as shown in Fig 4.9, these lower values of DIO Doublings enables the Trickle timer to fire the DIO very quickly.
C. Energy Consumption

![Energy Consumption Diagram](image)

Fig 4.20 Energy Consumption

The DIO Doublings limit the number of times the DIO Minimum interval can be doubled. For a small value of DIO Doublings the DIO need to be transmitted more often even in steady network situations, but a very large values of DIO Doublings can also cause the network to remain inconsistent for long durations, therefore a minimum value of DIO Doublings cause less Energy Consumption is required.

As can be seen from Figure 4.10 the energy consumption is better in case of Trickle-S than Trickle and Trickle-F, because of better load balancing in Trickle-S. Trickle and Trickle-F have the same energy consumption. In general as we increase the DIO Doubling the energy consumption decreases, because for low values of DIO Doublings the DIOs are transmitted more often and the nodes will become ON.

D. Packet Delivery Ratio

The PDR is not affected by DIO Doubling. However, as can be seen the PDR is better in Trickle-S than in Trickle, and Trickle-F for the same reason as explained in section 4.5.1 D.
The objective of this simulation is to evaluate the performance of RPL for different values of Redundancy threshold $k$. The network scenario explained in section 4.3.2 that contains 100 client nodes and 1 server node is used to perform this simulation. And the simulation parameters are shown in the table 4.1. However we set the redundancy threshold $k$ in the range from 1 to 12 for each iteration of the simulation. We choose DIO Interval Minimum of 12 and DIO Doubling of 8.

### A. Network Formation Time

As can be shown in Figure 4.12 the Network Formation Time for the algorithms Trickle, Trickle-S, and Trickle-F is the same. In addition the Network Formation Time is decreasing as we increase the Redundancy Threshold, because there are more DIOs to be transmitted so the sooner the network is to be formed.
Fig 4. 12 Network Formation Time

**B. RPL Control Overhead**

Fig 4. 13 Control Traffic Overhead

We observe that the RPL control messages are less in the case of Trickle, for the same reason discussed in section 4.5.1 B

The Trickle-S has less Control messages than Trickle-F, because Trickle-F gives a priority to suppressed nodes, though the nodes may have already heard messages.
As can be shown in Figure 4.13 the Control Traffic Overhead is increasing as we increase the redundancy threshold. The control traffic overhead is very high for higher values of redundancy threshold (greater than 6) as shown in Fig 4.13, these higher values of redundancy threshold enables to have more DIOs.

C. Energy Consumption

![Energy Consumption Graph]

Fig 4.14 Energy Consumption

The Redundancy Threshold is used to suppress the DIO transmission. For a high value of redundancy threshold the DIO need to be transmitted more often.

As can be seen from Figure 4.14 the energy consumption is better in case of Trickle-S than Trickle and Trickle-F, for the same reason discussed in section 4.5.1 C. In general as we increase the redundancy threshold the energy consumption increases because of high transmission of DIO.

D. Packet Delivery Ratio

As shown in Figure 4.15 the Packet Delivery Ratio is better in the case of Trickle-S than the Trickle, and Trickle-F, for the same reasons as discussed in section 4.5.1 D.

Another observation is that as we increase the Redundancy threshold the PDR increases, this is because of the fact that as we increase the number of DIO to be transmitted the better the paths to be formed, so the packet will use the best path to deliver to the DODAG root.
In the second phase of simulation we perform a simulation of random topology which contains 500 client nodes and 1 server node to evaluate the performance of RPL for the Trickle algorithms (Trickle, Trickle-S, Trickle-F) on the RPL parameters (DIO Minimum Interval, DIO Doublings, and Redundancy Threshold).

We use the same experimental setup as explained in section 4.3.2 and shown in Fig 4.3. The simulation parameters used in this simulation are shown in Table 4.1.

4.6.1 DIO Interval Minimum
We evaluate the Performance metrics by changing the Trickle parameter DIO Interval Minimum value from 3 to 14 in the subsequent iterations of the simulation, by keeping K to 10 and DIO Doublings to 8.
A. Network formation Time

Figure 4.16 illustrates the Network Formation Time i.e., the time needed for all the nodes to receive at least one DIO message. Trickle, Trickle-S, and Trickle-F have the same Network Formation Time. As can be seen in figure 4.16, the lower DIO interval minimum is, the sooner the DODAG is formed and the sooner all the nodes can send upward traffic.

As we increase the DIO Interval, the Network Formation Time also increases and for DIO Interval Minimum between 3 and 7 the Network Formation Time increases slightly, however from DIO Interval Minimum of 10 it increases linearly.

B. RPL Control Overhead
The RPL control messages are less in the case of Trickle, for the same reason explained in section 4.5.1 B.

The Trickle-S has less Control messages than Trickle-F, because Trickle-F gives a priority to suppressed nodes, though the nodes may have already heard messages.

As can be shown in Figure 4.17 the Control Traffic Overhead is decreasing as we increase the DIO Interval Minimum. The control traffic overhead is very high for lower DIO Interval Minimum (Lower than 8) as shown in Fig 4.17, these lower values of DIO Interval Minimum enables the Trickle timer to fire the DIO very quickly.

**C. Energy Consumption**

To compute the Energy consumption by the network we used the Powertrace tools as discussed in section 4.5.1 C.
Trickle-S has better energy consumption than Trickle and Trickle-F. In Trickle-S since all the nodes have synchronized intervals they don’t have propagation delay so the nodes have better load balancing so the lower energy consumption than Trickle and Trickle-F. However in Trickle and Trickle-F there is propagation delay to send the DIO so they have higher energy consumption than Trickle-S.

The RPL network consumes about 5.9% Radio ON time when DIO interval is very small (i.e. 3) in Trickle, and Trickle-F where as in Trickle-S the energy consumption is 5.3 % as shown in the Figure 4.18. For DIO Interval Minimum of 4 to 7 the Energy Consumption keeps on decreasing in a linear fashion. However the best Energy Consumption can be observed for DIO Interval Minimum of 8 and higher where the Energy Consumption is about 2.6% and remains constant for all values of DIO Interval Minimum of 8 and higher. This means that optimum value for DIO Interval Minimum regarding Energy Consumption is between (8-16).

**D. Packet Delivery Ratio**

As shown in Figure 4.19 the Packet Delivery Ratio is better in the case of Trickle-S than the Trickle, and Trickle-F, for the same reason explained in section 4.3.1 D.

Because of high Control Traffic in the case of Trickle-F as shown in Figure 4.19 the Packets are suffered from collision and retransmission, though the best path could be found but it is suffered by collision and retransmission. But in the case of Trickle; since the loads are unbalanced there are paths that cannot be discovered so they cannot form the best paths to deliver the packets. But in the case of Trickle-S all the nodes perform the listening and transmission timers are approximately aligned across the nodes, so there are better paths to be discovered.
Another observation is that as the DIO interval increases the PDR increases, this is because of the Control Traffic Overhead; as shown in Figure 4.17 the control traffic overhead is decreasing as the DIO interval minimum increases so the Network is not suffering from traffic of RPL control messages.

![Packet Delivery Ratio](image)

**Fig 4.19 Packet Delivery Ratio**

### 4.6.2 DIO Interval Doublings

The objective of this simulation is to evaluate the performance of RPL for different values of DIO Doublings. For this simulation we set the DIO Doublings in the range between 3 to 14 for each iteration of the simulation. We choose DIO Interval Minimum of 12 and Redundancy Threshold of 10.

#### A. Network Formation Time

After changing the DIO Doubling interval in the range 3 to 14, we observe that the network formation time is not affected and the network setup time is about 22.5 seconds for Trickle, Trickle-S, and Trickle-F as shown in Figure 4.20.
Network Formation Time is not depending on DIO Doubling i.e., changing DIO doubling doesn’t affect the Network Formation Time for Trickle, Trickle-S, and Trickle-F.

B. Control Traffic Overhead

We observe that the RPL control messages are less in the case of Trickle, for the same reason discussed in section 4.5.1 B
The Trickle-S has less Control messages than Trickle-F, because Trickle-F gives a priority to suppressed nodes, though the nodes may have already heard messages.

As can be shown in Figure 4.21 the Control Traffic Overhead is decreasing as we increase the DIO Doublings. The control traffic overhead is very high for lower DIO Doublings (Lower than 6) as shown in Fig 4.21, these lower values of DIO Doublings enables the Trickle timer to fire the DIO very quickly.

C. Energy Consumption

![Energy Consumption Graph](image)

Fig 4. 22 Energy Consumption

As can be seen from Figure 4.22 the energy consumption is better in case of Trickle-S than Trickle and Trickle-F, because of better load balancing in Trickle-S. Trickle and Trickle-F have the same energy consumption. In general as we increase the DIO Doubling the energy consumption decreases, because for low values of DIO Doublings the DIOs are transmitted more often and the nodes will become ON.

D. Packet Delivery Ratio

The PDR is not affected by DIO Doubling. However, as can be seen the PDR is better in Trickle-S than in Trickle, and Trickle-F for the same reason as explained in section 4.5.1 D.
4.6.3 Redundancy Threshold

The objective of this simulation is to evaluate the performance of RPL for different values of Redundancy threshold \( k \). For this simulation we set the redundancy threshold \( k \) in the range from 1 to 12 for each iteration of the simulation, and we choose DIO Interval Minimum of 12 and DIO Doubling of 8.

A. Network Formation Time

As can be shown in Figure 4.24 the Network Formation Time for the algorithms Trickle, Trickle-S, and Trickle-F is the same. As can be seen from figure 4.24, the Network Formation Time is decreasing as we increase the Redundancy Threshold, because there are more DIOs to be transmitted so the sooner the network is to be formed.
### B. RPL Control Overhead

We observe that the RPL control messages are less in the case of Trickle, for the same reason discussed in section 4.5.1 B.

The Trickle-S has less Control messages than Trickle-F, because Trickle-F gives a priority to suppressed nodes, though the nodes may have already heard messages.
As can be shown in Figure 4.25 the Control Traffic Overhead is increasing as we increase the redundancy threshold. The control traffic overhead is very high for higher values of threshold (greater than 6) as shown in Fig 4.25, these higher values of redundancy threshold enables to have more DIO messages.

C. Energy Consumption

![Energy Consumption Graph]

Fig 4. 26 Energy Consumption

The Redundancy Threshold is used to suppress the DIO transmission. For a high value of redundancy threshold the DIO need to be transmitted more often.

As can be seen from Figure 4.26 the energy consumption is better in case of Trickle-S than Trickle and Trickle-F, for the same reason discussed in section 4.5.1 C. In general as we increase the redundancy threshold the energy consumption increases because of high transmission of DIO.

D. Packet Delivery Ratio

As shown in Figure 4.27 the Packet Delivery Ratio is better in the case of Trickle-S than the Trickle, and Trickle-F, for the same reasons as discussed in section 4.5.1 D.

Another observation is that as the redundancy threshold increases the PDR increases, this is because of the increase in the number of DIO to be transmitted the better the paths to be formed, so the packet will use the best path to deliver to the DODAG root.
4.7 Discussion and Analysis

An analysis of the obtained results at large can provide a set of guidelines to tune Trickle parameters, as well as to answer the research questions on tuning the Trickle parameters and choosing Trickle algorithms, and to identify the tradeoffs among the performance of the routing protocol.

From the results we can observe that Trickle-S can form the network better in a less energy consumption than Trickle and Trickle-F. Trickle-F can identify all the available paths but it is suffering from the collisions because of its high Control Traffic Overhead which implies high energy consumption as well.

Low DIO interval minimum values result in the creation of the first DODAG within a short space of time; however its routes are suboptimal and need extra time to be refined. This behavior highlights a trade off time Vs route quality for the formation of the first DODAG, from the above results to achieve a good PDR with in short time to form the DODAG, the recommended value of DIO interval minimum is between 5 to 10. The DIO Doubling has not impact on the performance of the Packet Delivery Ratio which implies on the path of the network, the recommended values regarding to the energy consumption is between 8 to 14. The redundancy threshold k, instead, can be used to regulate the trade-off between energy consumption and route quality: low k values reduce the energy consumption of each node, resulting however in suboptimal routes.

Fig 4. 27 Packet Delivery Ratio
Chapter FIVE

5.1 Trickle-Fair broadcast suppression and Synchronizing Trickle intervals

Routing information updates, as those carried by DIO messages, are strictly dependent on the source of the message: suppressing one transmission or another is not always equivalent, since the two suppressed messages carry different information. Should some node be not allowed to send any message for a long time, some routes may remain undiscovered and therefore unused for such a time even though they are better than those currently active in the DODAG. The original algorithm provides each node with equal average broadcast transmission probability in the long run. However, for routing purposes it is important that every node is given the opportunity to share its routing information in the shortest possible time scale, so as to allow the quick discovering of all available routes, and then choose the best ones according to the established routing metrics. As discussed in section 2.10, Trickle-F can achieve to find the best path by giving explicit priority to the nodes that has been suppressed their transmission, however it experience to have high control traffic overhead that implies energy consumption and high traffic in the network. As discussed in section 2.9 and from the simulation results of sections 4.5 and 4.6, Trickle-S can achieve the best path with low energy consumption than Trickle-F. So we observe that by combining both the algorithms we can achieve a better energy consumption and efficient routing protocol, from Trickle-F we can take the advantage of explicitly giving the priority to nodes that suppressed their transmission, to send their DIO in the next interval so the DODAG will have better information soon, and from Trickle-S, the advantage of load balancing i.e., to perform the Trickle timers in approximately aligned intervals, so all the nodes will perform the listening and transmission period in approximately aligned intervals.

Based on the previous observations, we propose a modified version of the Trickle algorithms, i.e Trickle-FS, which aims at guaranteeing a fair short-term broadcast suppression among nodes in a neighborhood in order to facilitate the rapid discovery of all available paths, and to handle the propagation of meta-data efficiently and load balancing in the network, i.e., to perform the Trickle timers in approximately aligned across the nodes. The rationale behind Trickle-FS is to prioritize each node strictly depending on the number of consecutive suppression, and to start earlier if a node receives a DIO in its listening period. In order to achieve this, Trickle-FS introduces a modification to the original algorithm in the computation of the start time and length of the next transmission period.

The pseudo-code of Trickle-FS is presented in Algorithm 4. In order to provide broadcast fairness, each node keeps track of $s$, the number of continuous communication intervals in which a message transmission has been suppressed. And in order to provide an approximately aligned trickle timers each node keeps $\partial s$, the time the node receives DIO. At time $t$, if a DIO is transmitted, $s$ is reset: otherwise, the counter is incremented. And if a node receives a consistent transmission when it is in the listening period $\partial s$ will have the time difference between listen
period and DIO received time. Each node gets a transmission priority proportional to the number of last consecutive suppressed transmissions. Priority dependence on \( s \) is enforced by modifying the length of the listening and transmitting intervals, and each node starts its next interval earlier if it receives DIO in its listening interval.

Each transmission period \( T \) performs the following: first it subtracted by \( \partial s \) i.e., \( T_s = T - \partial s \), and the obtained result is set to a variable length which is proportional to the number of suppressed transmissions, \( T_s \frac{Ts}{2^s} \). At the beginning of each period, the transmission instant \( t \) is selected in a sub-period depending on \( s \) as follow: \( \frac{T_s}{2^{s+1}}, \frac{T_s}{2^s} \). This ensures strict prioritization of the sub-periods according to \( s \), and makes all the nodes to start their Trickle timer in almost aligned time. The larger \( s \) is, the closer the sub-period is. This modification guarantees that nodes that have waited longer get higher transmission probability, while nodes that have been suppressing the same number of transmissions will have the same transmission probability.

Fig. 5.1 illustrates an example of Trickle-FS operation in the same scenario as that in Fig. 2.2 in the case of four nodes and \( k=2 \). This time, nodes 1 and 2 schedule DIO transmission in a shorter next transmission period (as matter of fact, during the listening interval of nodes 3 and 4) and therefore get priority over nodes 3 and 4, which transmitted in the previous period.

Algorithm 4: Trickle-FS

1. Function initialization()

\[
T \leftarrow I_{\text{min}} \\
S \leftarrow 0
\]

2. Function Begin()

\[
T_s \leftarrow T - \partial s \\
C \leftarrow 0 \\
\partial s \leftarrow 0 \\
t \leftarrow \left( \frac{T_s}{2^{s+1}}, \frac{T_s}{2^s} \right)
\]

3. Function consistent transmission Received()

\[
C \leftarrow C + 1 \\
\text{If } T \in \left[ T/4, T/2 \right] \\
\partial s \leftarrow W_s (T-T/2) \\
\text{End if}
\]

4. Function TimerExpire()

\[
\text{If } k \geq C \text{ then} \\
\text{Transmit DIO} \\
S \leftarrow 0 \\
\text{else} \\
S \leftarrow S + 1 \\
\text{End if}
\]

5. Function interval ends()
\[ C \leftarrow 0; \]
\[ \text{If in consistent transmission received then} \]
\[ I \leftarrow I_{\text{min}} \]
\[ \text{Else } I \leftarrow I \times 2 \]
\[ \text{If } I_{\text{max}} \leftarrow I \text{ then} \]
\[ I \leftarrow I_{\text{max}} \]
\[ \text{End if} \]
\[ \text{End if} \]

Fig 5. Trickle-FS with K=2

### 5.2 Performance evaluation of Trickle-FS

In order to evaluate our proposal we run the same set of simulation as discussed in section 4.3 both for phase 1 and phase 2. Fig 5.2, and fig 5.3 illustrates the Energy Consumption with respect to DIO interval minimum and redundancy threshold respectively for the network setup of phase 1, the Energy Consumption is the same in the case of Trickle-S and Trickle-FS, as DIO interval minimum is increasing the Energy Consumption is decreasing because of the low DIO interval enable to fire the DIOs very fast which implies energy consumption, and as K is increasing the energy consumption is increasing, because of the high number of DIOs to be fired.

Fig 5.4 and Fig 5.5 illustrates the Packet Delivery Ratio with respect to DIO interval minimum and redundancy threshold for the network setup of phase 1. As can be seen the PDR is better in the modified algorithm which is Trickle-FS i.e., best route [6] are obtained using the modified algorithm, in particular when the DIO interval minimum is low the PDR in the case of Trickle-FS is better than Trickle, Trickle-F, and Trickle-S, the DODAG form the best route within short period of time because the DODAG gets all the available messages very soon, and the better paths to be formed. Better routes are achieved through the spatial fairness and approximately aligned Trickle timers among the nodes.
Both Trickle-S and Trickle-FS have better energy consumption than Trickle and Trickle-F. In Trickle-S and Trickle-FS since all the nodes have synchronized intervals they don’t have propagation delay so the nodes have better load balancing, and less control traffic overhead. However in Trickle and Trickle-F there is propagation delay to send the DIO so they have higher energy consumption than Trickle-S and Trickle-FS.
As shown in Figures 5.4 and 5.5 the Packet Delivery Ratio is better in the case of Trickle-FS than the Trickle, Trickle-S, and Trickle-F, the reasons for having Trickle-FS better PDR is as follows:

Because Trickle-FS provide explicit priority to a node that suppressed its transmission to transmit in the next Trickle interval so the DODAG will have better information of the nodes and chooses the best path to send their packet via the best route, and all the nodes provide their
Trickle timer in almost aligned intervals, the propagation delay is reduced and have better energy consumption than Trickle and Trickle-F. In the case of Trickle-FS the network will have all information very soon and will find the best path soon.

Better routes are achieved by the modified algorithm for the same value of DIO interval minimum and k.
Chapter SIX

Discussion

The results of the four performance metrics for the Trickle algorithms i.e., Trickle, Trickle-S, and Trickle-F are obtained, analyzed, and organized in a logical manner to provide results in support of answering the research questions. The performance of the protocol is influenced by several factors; among them one is the protocol parameters. In this study we studied Trickle algorithms and Trickle parameters and their impact on the performance of RPL.

At the first phase the effect of lossyness is investigated, in section 4.4 we evaluate the performance of RPL for Trickle, Trickle-S, and Trickle-F and compared for the Network scenario explained in section 4.3.2.1 and the results showed that the Network formation time for the algorithms Trickle, Trickle-S, and Trickle-F is the same, the Control Traffic Overhead is higher in case of Trickle-F for the reason explained in section 4.3.2, and the energy consumption is better in the case of Trickle-S because of the load balancing provided by the algorithm. And finally the Packet Delivery Ratio is better in the case of Trickle-S, though Trickle-F can provide better Packet Delivery Ratio; unfortunately it is suffered from congestion of the high Control Traffic Overhead, which implies to have lower Packet Delivery Ratio than Trickle-S.

In the second phase of the simulation the network scenario explained in section 4.3.2.2 i.e., the network composed of 500 nodes is performed and the result highlighted that it has the same behavior with the network scenario of section 4.3.2.1. i.e., the network composed of 100 nodes.

By observing the above results we claim that by modifying the algorithms Trickle-S and Trickle-F we can have a better route than Trickle-S, and we propose a modified algorithm of Trickle-S and Trickle-F which is the Trickle-FS, and we perform the same set of simulation as discussed in section 4.3.2.1. And the obtained results indicates that the modified algorithm can provide a better route than Trickle-S, with the same energy consumption and network formation time.

Routing is an important component in sensor networks because it performs the packet forwarding and routing decisions and therefore accounts for the utilization of the network resources. The worst routes cause more retransmissions and wastage of resources. The performance of RPL is controlled by several parameters like DIO intervals, DIO Doublings, and Redundancy threshold. We observe from the above simulation the Network Formation Time is directly related with the DIO interval minimum, and inversely related with redundancy threshold.

The Control Traffic Overhead is heavily affected by DIO interval minimum, redundancy threshold. The more recurring values of these parameters cause packet collisions and packet loss due to more radio on times by the individual nodes. So any extra radio on time by the improper configuration of these parameters is harmful for the control Traffic overhead.
Similarly energy consumption is also affected by DIO interval minimum, DIO Doublings, and redundancy threshold. Energy consumption is inversely proportional with DIO interval minimum and DIO Doublings, and directly proportional with redundancy threshold.

The Packet Delivery Ratio is affected by DIO interval minimum and redundancy threshold. PDR is directly proportional to DIO interval minimum and redundancy threshold.
Chapter Seven

Conclusion

Wireless networks have several constraints like energy, bandwidth, computing and memory, which make routing in these devices more challenging. RPL being a new proposed protocol for low power and lossy network is under experimentation and important aspects are needed to be evaluated. It is a major component of consuming the energy of sensor nodes.

We observe that a set of RPL parameters are crucial for its better performance in vast areas of sensors applications. Trickle timer is an important component of RPL in order to utilize the scarce resources of WSN more efficiently. In this work we presented a performance evaluation of RPL with particular focus on Trickle. Different Trickle algorithms i.e., Trickle, Trickle-S, and Trickle-F are extensively studied and evaluated using simulation. Simulation results showed the performance of RPL for the Trickle algorithms i.e., Trickle can form the Network at the same time, while Trickle-S have better energy consumption and better path than Trickle, and Trickle-F. However Trickle-F can provide the opportunity for all the nodes to send the DIO so the network can have better knowledge of the available routes by having the tradeoff to provide high Control Traffic Overhead, which implies energy consumption and congestion of traffic i.e., low packet delivery ratio. We claim that by having the explicit prioritize the node that suppressed for long time to send the DIO in the next Trickle interval and to have approximately aligned Trickle timers, we propose a new routing algorithm i.e., Trickle-FS. And our simulation results validate our proposal. We can achieve a better route with the same energy consumption of Trickle-S and same Network Formation Time with Trickle, Trickle-F, and Trickle-S.

We observe also how the routing behavior is affecting by Trickle settings. A set of guidelines was derived from simulation results to optimize network formation time, route quality and energy consumption.
Acknowledgment

I would like to express my deepest gratitude to my advisor Prof. Enzo Mingozzi and Dott. Carlo Vallati for their invaluable advice and continuous guidance throughout the progress of this thesis. Also I would like to express my heartfelt appreciation to the Secretaries of the Embedded Computing Systems.

Also I would like to use this opportunity to thank Prof. Giuseppe Anastasi, and the management of University of Pisa, for giving all the facilities and support for doing this project.

Last but not least, I would like to thank my family: my parents and to my brothers and sisters for supporting me spiritually, and my friends for the sleepless nights we were working together. I am grateful to all the people who in one way or the other has contributed in making this thesis succeed.
References


[19].C. Vallati, E. Mingozzi “Trickle-F: fair broadcast suppression to improve energy-efficient route formation with the RPL routing protocol” in Sustainable Internet and ICT for Sustainability (SustainIT), Palermo, Italy, 2013.


Appendix A.

```c
#include "contiki-conf.h"
#include "net/rpl/rpl-private.h"
#include "lib/random.h"
#include "sys/clock.h"
#include <string.h>
#include "net/conf.h"
#include "net/zpl/zpl.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define DEBUG DEBUG_NONE
#include "net/uip-debug.h"
#include "net/neighbor-inrc.h"
#include "net/neighbor-attr.h"

/**************************************************************************/

#if RPL_STATS
static uint16_t nextStat;
#endif

static struct ctimer periodic_timer;

static void handle_periodic_timer(void *ptr);
static void new_dio_interval(rpl_instance_t *instance);
static void handle_dio_timer(void *ptr);

static uint16_t next_dis;

/* dio_send_ok is true if the node is ready to send DIoTs */
static uint8_t dio_send_ok;

/**************************************************************************/

static void handle_periodic_timer(void *ptr)
{
    rpl_purge_routes();
    rpl_recalculate_ranks();
    /* handle DTS */
    #ifdef RPL_DIS_SEND
    next_dis++;
    if(zpl_get_any_dag() == NULL && next_dis >= RPL_DIS_INTERVAL) {
        next_dis = 0;
        dis_output(NULL);
    }
    #endif
    ctimer_reset(&periodic_timer);
}

/**************************************************************************/

static void new_dio_interval(rpl_instance_t *instance)
{
    uint32_t time, times;
    /* TODO: too small timer intervals for many cases */
    time = 1UL << instance->dio_intcurrent; //2^6
    /* Total 1 in CLOCK_TICKS */
    instance->dio_next_delay = (time * CLOCK_SECOND) / 1000;
    if(instance->trickle){
```

Continued
Continued

// Trickle-F modifications are active
// Let's modify interval length according to the number of suppressed tx
time = time >> instance->suppressed_counter;
if (time == 0)
  // I've been suppressing the transmission for long time
  // Let's set the interval length at the minimum
  time = 1UL << 1;
}

if (instance->trickles)

  // Trickle-F modifications are active
  // Let's modify interval length according to the delay of listening time
  instance->trickle_sync = time - instance->dio_received_time;
  time = instance->trickle_sync;

if (instance->tricklefs)

  // Trickle-FS modifications are active
  instance->trickle_sync = time - instance->dio_received_time;
  time = instance->trickle_sync;
  time = time >> instance->suppressed_counter;

else {
  PRINTF("RPL: trickle &u \n", time);
  PRINT_RPL("RPL: trickle &u \n", time);
}

/* Convert from milliseconds to CLOCK_TICKS. */
time = (time * CLOCK_SECOND) / 1000; // 2^3 tick

/* random number between I/2 and I */
time = time;
time = time >> 1; // divided by 2

/* the intervals must be equally long among the nodes for Trickle to
* operate efficiently. Therefore we need to calculate the delay between
* the randomized time and the start time of the next interval.
*/
instance->dio_next_delay = time;
instance->dio_send = 1;

/* reset the redundancy counter */
instance->dio_counter = 0;
instance->dio_received_time = 0;

/* new interval begins*/
PRINTF("Start of Interval\n");

/* schedule the timer */
PRINTF("RPL: Scheduling DIO timer &u ticks in future (Interval)\n", time);
timer_set(&instance->dio_timer, time, &handle_dio_timer, instance);

/*******************************************************************************/
static void handle_dio_timer(void *ptr)
{
  rpl_instance_t *instance;
  instance = (rpl_instance_t *)ptr;

Continued
PRINTF("RPL: DIO Timer triggered\n");

if(!dio_send_ok) {
    if(!ip_d6_get_link_local(ADDR_PREFERRED) != NULL) {
        dio_send_ok = 1;
    } else {
        PRINTF("RPL: Postponing DIO transmission since link local address is not ok\n");
        ctimer_set(&instance->dio_timer, CLOCK_SECOND, &handle_dio_timer, instance);
        return;
    }
}

if(instance->dio_send) {
    /* send DIO if counter is less than desired redundancy */
    if(instance->dio_counter < instance->dio_redundancy) {
ifdef /* RPL_CONF_STATS */
    instance->dio_totsend++;
#endif /* RPL_CONF_STATS */

    /* if metadata is received in the second half of the listening period*/
    clock_time_t dio_recvceive_t; /* this has to be done when the tx counter is incremented*/
    dio_recvceive_t = clock_time();
    if((dio_recvceive_t > times >> 1) && (dio_recvceive_t < times))
        instance->dio_recieveftime = abs(dio_recvceive_t - times);
    else {
        instance->dio_suppressed_counter++;
    }
    instance->dio_send = 0;
    ctimer_set(&instance->dio_timer, instance->dio_next_delay, handle_dio_timer, instance);
}
else {
    /* check if we need to double interval */
    if(instance->dio_intcurrent < instance->dio_intmin + instance->dio_intdoubl) {
        instance->dio_intcurrent++;
    }
    new_dio_interval(instance);
}
}/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%**/
void rpl_reset_periodic_timer(void)
{
    next_dio = RPL_DIS_INTERVAL - RPL_DIS_START_DELAY;
    ctimer_set(&periodic_timer, CLOCK_SECOND, handle_periodic_timer, NULL);
}/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%**/
/* Resets the DIO timer in the instance to its minimal interval. */
void rpl_reset_dio_timer(rpl_instance_t *instance)

#ifdef !RPL_LEAF_ONLY
/* Do not reset if we are already on the minimum interval, 
   unless forced to do so. */
if(instance->dio_intcurrent > instance->dio_intmin) {
    instance->dio_counter = 0;
    instance->dio_intcurrent = instance->dio_intmin;
    new_dio_interval(instance);
}
#endif /* RPL_CONF_STATS */
```c
static void handle_dao_timer(void *ptr)
{
    rpl_instance_t *instance = (rpl_instance_t *)ptr;
    if(!dio_send_ok && uip_d6_get_link_local(ADDR_PREFERRED) == NULL) {
        printf("RPL: Postpone DAO transmission\n");
        ctimer_set(&instance->dao_timer, CLOCK_SECOND, handle_dao_timer, instance);
        return;
    }
    /* Send the DAO to the DAO parent set -- the preferred parent in our case. */
    if(instance->current_dag->preferred_parent != NULL) {
        printf("RPL: handle_dao_timer - sending DAO\n");
        /* Set the route lifetime to the default value. */
        printf("RPL DAO: %s DAO to pp ", __func__);
        PRINT6ADDR_DAO(&instance->current_dag->preferred_parent->addr);
        printf("\n");
        dao_output(instance->current_dag->preferred_parent, instance->default_lifetime);
    } else {
        printf("RPL: No suitable DAO parent\n");
    }
    ctimer_stop(&instance->dao_timer);
}

void rpl_schedule_dao(rpl_instance_t *instance)
{
    clock_time_t expiration_time;
    expiration_time = etimer_expiration_time(&instance->dao_timer.etimer);
    if(!etimer_expired(&instance->dao_timer.etimer)) {
        printf("RPL: DAO timer already scheduled\n");
    } else {
        expiration_time = RPL_DAO_LATENCY / 2 +
            (random_rand() % (RPL_DAO_LATENCY));
        printf("RPL: Scheduling DAO timer %u ticks in the future\n",
            (unsigned)expiration_time);
        printf("RPL DAO: %s Scheduling DAO timer %u ticks in the future\n",
            __func__,
            (unsigned)expiration_time);
        ctimer_set(&instance->dao_timer, expiration_time,
            handle_dao_timer, instance);
    }
}
```