UNICAM

DOCTORAL THESIS

Routing and Reliability Improvements in WM-Bus protocol for smart cities

Author:
Fabio Pagnotta

Supervisor:
Prof. Leonardo Mostarda

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in

Computer Science

UNICAM

Abstract

Computer Science

Doctor of Philosophy

Routing and Reliability Improvements in WM-Bus protocol for smart cities

by Fabio Pagnotta

In a smart city, accounting information such as gas and electricity usage is gathered by meter nodes and sent to the collector node. A trade-off between the location of meters and the best quality communication signal is often needed and difficult to achieve for urban constraints and other communication signals. Meters can have limited resources such as memories and CPU. The thesis presents two routing extensions of the WM-Bus which is an open standard protocol for smart metering systems for improved performance and routing in a dense mesh utility network.

The former extension is called NARUN: Noise Adaptive Routing in Utility Networks. In NARUN, the collector calculates the path with the least noise to reach the destination meter. A weighted network graph that shows the connections among meters is used, where an edge weight defines the link failure index. No control messages are used to keep the weights updated. Meters report link failure index back to the collector by means of ordinary reading messages.

The latter extension is NARUN-PC: NARUN with path cache. NARUN-PC extends NARUN by introducing a caching strategy in the collector node.

This thesis presents models and simulations of these two extensions. Performances have been evaluated in a real-life topology where a subset of the edges is affected by different levels of noise.

Results show that NARUN has the lowest failure rate and traffic compared to the DSR protocol when the noise power is higher than -73 dBm. NARUN-PC decreases the traffic load and failure rate with respect to NARUN when the noise power is higher than -73 dBm.

Contents

Acknowledgements

I would like to express my gratitude to my primary supervisor, professor Leonardo Mostarda, who guided and supported me throughout this project.

I would also like to thank professor Orhan Gemikonakli for the support, valuable knowledge, and experience in the field that helped me in this project.

I would like to offer my special thanks to professor Rosario Culmone for the insightful idea of NARUN, the availability, and the support that he provided me during my Ph.D. period.

I wish to extend my special thanks to the University of Camerino (UNICAM) that had been providing me with an international and valuable environment for my growth.

I would also like to thank professor Diletta Cacciagrano and the other colleague whom I have been working on during my Ph.D. period.

My appreciation also goes out to my family and friends for their encouragement and support all through my studies.

Chapter 1

Introduction

This chapter provides the introduction, the motivation and the methodology of this thesis. The section 1.1, covers the introduction while the sections 1.2 and 1.3 provides the motivation. More precisely, section 1.3 summarises the motivation with a thesis statement. In the same section, the research questions and the approach used have been discussed. The section 1.4 provides the methodology of the thesis. Finally, the section 1.5 guides the navigation of the reader with the thesis structure.

1.1 Introduction

The sheer size of today's cities makes the manual reading of gas, water, and electricity meters a significant challenge. Fortunately, the technological advances in communication networks gave rise to the development of automated meter reading systems. Network meter systems enable automated, periodic, and real-time readings, reducing the management costs and enabling real-time data analysis. This technology can also help in reducing resource consumption by raising the awareness of prosumers. Directive 2009/72/EC of the European Parliament requires the use of smart metering systems to empower the consumers in the electricity and gas supply markets [1]. A metering system consists of a set of distributed nodes connected via a communication network. A collector node gathers data from meter nodes that measure usage. Data is then sent to the collector node. Repeaters can be used to extend the node transmission range. The main electronic parts of a meter consist of a simple micro-controller with sensors. These devices are limited in terms of computation power, memory and can be battery powered [2]. Hence, the highest cost is not associated with the device itself but with the subsystem of interconnection and communication. These subsystems can use optical fiber, WiFi, cellular networks, or other transmission technologies. The network deployment and technologies used in communication represent the critical parts of the reading service [3].

Meter Bus (M-Bus) is a European standard for metering systems that includes the physical layer, the data-link layer [4], and the application layer [5]. The network layer is presented as an optional one and there is no specification for any standard routing algorithm. No topology restrictions are stated, except for the token ring which cannot be used. Wireless Meter Bus (WM-Bus) is the wireless version of the M-Bus protocol. The EN 13757-4 standard describes its physical and data-link layer [6]. The EN

13757-5 provides node relaying [7] whereas the EN 13757-7 introduces transport and security services [8]. The line of sight communication range can be a few hundred meters (with 868 MHz or 433 MHz frequencies) or more than 5 km (using 169 MHz frequency). The former is widely used in Europe [9].

1.2 Motivation

In smart cities [10–12], meter installation introduces a trade-off between reliable connectivity and setup cost. Placing meters with the best quality communication signals is often expensive or it is not feasible because of urban constraints [13–15]. WM-Bus networks can be installed in dense urban areas where many other communication signals coexist. This can cause interference and generate poor quality links [16], which increases communication overhead and reduce the lifetime of the battery-operated meters[17]. Poor quality links also affect the data reading time since a successful reading message can require several attempts.

1.3 Thesis statement and research questions

The motivations previously mentioned bring us to the following thesis statement:

In a smart city, devices can route messages by extending the coverage area and improving reliability without introducing extra messages.

The routing task also needs to be computationally simple enough for devices with limited capabilities in terms of memory and CPU [2]. The following research questions concern the characterization of this routing protocol.

1. How can routing be performed in a dense smart metering network in order to reduce failure rate ?
2. How can routing be performed using devices with limited capabilities in a dense smart metering network?
3. How can routing be performed in a dense smart metering network when the failure rate is high ?
4. How can routing be performed in a dense smart metering network when the routing path frequently changes?

1.4 Methodology

This section covers the scientific methodology chosen to approach the research questions.

First of all, the state of art of routing and simulations has been analyzed to understand the strength and weaknesses of the already proposed solutions in the research field. The analysis covers the protocol behavior and how reliability has been

achieved and measured. These information were reviewed to check how these protocols are tailored for dense networks in smart cities (Chapter 2). The authors focus on the WM-Bus protocol, an european standard protocol for smart metering solutions.

This analysis shows the need of a routing protocol which takes into account the meter position and the communication interference without introducing overhead messages compared to the well-known protocols.

A detailed model of Noise Adaptive Routing for Utility Networks (NARUN) has been proposed in this thesis. The model covers the representation of the network in NARUN, the communication and considered metrics, and the collector and meter behavior. NARUN has been compared with a well-known protocol called Dynamic Source Routing (DSR)[18]. The protocol has been compared through simulations (chapter 3.4).

Various routing simulations for WM-Bus have been analyzed in the thesis but none of them were updated or consider the same kind of topology. Therefore, a routing simulation has been developed and tested against a set of small networks.

Collector reading failure rate, Reading rate and Message received have been considered in the evaluation of the protocols. The first and second metrics are related to the failure rate from the collector prospective while the last metric represents the overhead of the protocol.

Simulations has been performed under different network conditions (by changing or disconnecting edges) using the Additive White Gaussian Noise (AWGN) model in a dense urban area. Result has been discussed in section 3.5. Although results show the advantage of NARUN with respect to the DSR protocol [18], it 'suffers' for unstable links, i.e. links frequently broken. As NARUN always selects the best available path, unstable links might be frequently selected, hence generating failing paths. This increases the number of messages. NARUN-PC is a protocol improvement that resolves this issue, described in chapter 4. A detailed model of NARUN-PC has been proposed in section 4.1 to fix the disadvantage of NARUN protocol. NARUN-PC had been compared against DSR[18] and NARUN protocols under different network conditions (by changing or disconnecting edges) using the Additive White Gaussian Noise (AWGN) model in a dense urban area. The results show an improvement in terms of number of messages respect to the NARUN protocol.

In the conclusion, future works of this protocol have been proposed to further analyze the performance of NARUN and NARUN-PC under different scenario and noise model.

1.5 Thesis organization

The following chapters are organized as follows: Chapter 2 reports the state of the art, Chapter 3 provides the system model and the benchmark of NARUN protocol, 4 provides the system model and the benchmark of NARUN protocol and the Chapter 5

concludes the thesis. The Appendix 6 includes an additional work similar to NARUN, that has been done during the PhD period.

1.6 Contribution

The dissertation proposes two novel routing protocols namely Noise Adaptive Routing for Utility Networks (NARUN) and Noise Adaptive Routing for Utility Networks with Path Cache (NARUN-PC). They aim to reduce the failure rate and traffic load without introducing overhead messages. An early formalization of NARUN has been introduced in paper [19]. This includes the routing algorithm and the eavesdrop behavior of nodes. In NARUN, communication occurs in a mesh topology without sending extra service messages. The routing path is appended to the read request message (as advised by the WM-Bus standard). A collector can request and receive data from networked meters. Meters can relay the request to the destination node and forward back the response. The collector decides the routing path depending on the internal representation of the network. This knowledge is represented as a graph where nodes are vertices and edges are interconnections. The edge weight gives a measure of the link failure index. More precisely, an infinite weight can denote a broken link, a weight equal to one denotes a perfect functioning link. Finally, a number greater than one denotes a noisy link that requires frequent re-transmission or error recovery. The routing path is calculated by minimising the failure index and path length. Meters eavesdrop on the surrounding environment and efficiently report information on link failure indexes back to the collector with ordinary reading messages. These are ordinary response WM-Bus packets. NARUN can run on simple meter devices with limited CPU and a small amount of memory. Hamming Error Correction Code (ECC) is used to increase communication reliability and measure the link failure index. A revised version of this protocol extends the previous one by covering edge cases and improving the algorithms [20]. The paper introduces the persistence routing behavior of the collector. Persistence behavior improve reliability by trying alternative paths to reach a destination node when the algorithm cannot find one. In [20], NARUN has been evaluated by simulations and compared against the Dynamic source routing (DSR) protocol under different network conditions in a real-life topology. Results show an improvement in terms of failure rate and traffic load. However, NARUN 'suffers' for unstable links, i.e. links frequently broken. To overcome this problem, NARUN-PC has been proposed and evaluated [21]. NARUN-PC introduces a path cache strategy in the collector. The evaluation compares NARUN, NARUN-PC, and DSR protocols in a real-life topology. Results show a slight improvement of failure rate and relevant improvement in terms of traffic load when NARUN-PC is compared against NARUN and the noise power is higher than -73 dBm.

Chapter 2

Literature Review

This chapter analyzes routing protocols and reliable mechanisms in literature to understand their strength and weaknesses. The context of the study is smart metering networks which are also referred as Advanced Metering Infrastructure (AMI) networks [22]. The analysis takes into consideration reliable frameworks and then, WM-Bus and more generally smart metering routing protocols.

Different frameworks are proposed for measuring and analyzing the failure rate of the WM-Bus for different cases. A framework for the analytical study of failure rates is presented as a case study where the suitability of WM-Bus for smart water grids is considered in [23]. The approach considered Packet Error Rate (PER), bandwidth, frequency, range, and energy requirements. The simulation framework for failure rate analyses of WM-Bus is also considered in [24]. The authors discussed the approach for developing a simulation of the WM-Bus protocol by extending NS-3[1] simulator. This work focused mostly on the physical and data-link layers of WM-Bus. The implementation details for this approach are given in the study [25]. They found out that their simple simulation model was comparable to the general trend of the hit rate in a real WM-Bus deployment.

The paper [26] analyses the WM-Bus for home automation systems by using 5G and M2M technologies. They had been developing a WM-Bus module for NS-3[2] simulator. They used the interference and packet delivery ratio between meters to validate their module. Technical details associated with the 868 MHz communication, such as range and indoor signal transmission efficiency under various interference levels, are studied in [27]. The frequency is also widely used in Europe and for this reason, this thesis implements it. Simulations were used to analyze the failure rate by varying the interference level in a smart city network.

Improvements to the reliability of the WM-Bus protocol are considered in some studies. In papers [13, 14], authors presented discussions on the inherent reliability problems related to the deployment of the WM-Bus network and they proposed a method for reliable data reception. In particular, the authors addressed the trade-off between reliability and the deployment cost for WM-Bus networks, which is one of the fundamental and inherent problems of the WM-Bus. The meter nodes are not mobile and since they are added to existing infrastructures, their optimum deployment

[1] (https://www.nsnam.org/)
[2] (https://www.nsnam.org/)

becomes difficult. The authors proposed a data recovery scheme exploiting the use of a deterministic packet transmission interval from the meter nodes to deal with this issue. The scheme groups erroneous packets coming from the same sender and saves them in memory. This method can recover packets by using this information. The drawback is the memory footprint requested by the scheme which depends on the number of senders. In [28], the authors proposed a method to increase the reliability and the network lifetime of WM-Bus. The proposed method extends the standard WM-Bus protocol making it suitable for "home energy management systems" which require bi-directional real-time communication. The authors added functions such as asynchronous meter trigger, adaptive slot scheduling, and bitmap-wise re-transmission request from the collector to WM-Bus to achieve reliable and energy-efficient real-time communication. They evaluated the success ratio, the node lifetime, and the time spent reading by changing the size of the network.

These studies analyze the reliability issue in a single-hop communication between the collector and meters. This thesis tackles this issue in a mesh topology by monitoring link failures without introducing extra messages to the WM-Bus protocol.

The mesh topology can extend the coverage area of the entire network however a routing protocol is needed to forward messages from the source to the receipt node.

In the context of Advanced Metering Infrastructure (AMI), routing protocols can be divided into geographic, reactive, and proactive [29]. Geographic routing protocols such as [30] can perform routing by avoiding critical traffic conditions and low-density networks. Nevertheless, this type of routing requires smart meters with a Global Positioning System (GPS) which is not always feasible due to the location of meters and energy consumption of meters. Moreover, this approach does not scale well with larger networks [31]. The proactive approach implements periodical broadcasts to update the data structures saved by each node in the network while the latter queries nodes on-demand without using this mechanism. Proactive protocols can be divided into link-state or distance vector.

In link-state protocols such as Open Shortest Path First (OSPF) [32] or Optimized Link State Routing (OSLR) [33], the network topology is stored by nodes and is maintained through message flooding. Flooding notifies the whole network when the link costs change or a link failure occurs. In the context of AMI, various optimizations have been proposed over Optimized Link State Routing (OSLR) protocol [34]. These extensions improve packet delivery ratio and latency against small networks. However, their approach are unsuitable for larger networks due to their service overhead when the number of nodes increases.

In distance-vector protocols, nodes store and maintain their local view of the network which includes the set of neighbours along with their respective link costs. This information is used to route messages in the network. Nodes periodically send messages to update the local view of neighbors. PSA-HD [35] is a proactive protocol used in Mobile Ad-hoc NETwork (MANET) networks where nodes are mobile and links are frequently created and broken. This protocol tries to find the shortest stable path

in the network. In detail, the shortest path is computed by minimizing the REfined Hamming DIStance (REHDIS) metric required to reach the destination node. The selection of the shortest path is performed amongst multiple paths in which metrics are shared between nodes with periodical message exchange. PSA-HD decreases the average end-to-end delay and increases the Packet Delivery Ratio (PDR) with respect to Proactive Source Routing (PSR) [36] and Predicted Probabilistic Coefficient Link Stability (PPCLSS) [37] protocols. PSR is a proactive protocol where the network topology is shared between nodes while PPCLSS is a routing protocol where stability is evaluated using the weighted sum of energy utilized by a node, link loss, and the average path size. Like PSA-HD, the NARUN protocol presented in this thesis uses Hamming distance for shortest path computation but NARUN path updates are done without exchanging periodic messages.

The most used and stable proactive routing protocol for low-power and lossy networks is called Routing Protocol for Low-Power and Lossy Networks (RPL) [22]. The network used is a destination-oriented directed acyclic graph (DODAG). The ancestor is the root node and intermediate nodes are chosen using a specific objective function (OF) by children. The standard objective functions are objective function zero (OF0), which minimizes the distance between the parent and the root node, and the minimum rank with hysteresis objective function (MRHOF) which minimizes the link cost associated with the routes. Although this protocol scales with thousands of nodes, periodical broadcasting is needed to update metrics, which requires high bandwidth when node disconnection and topology changes frequently occur.

Generally speaking, proactive protocols are not suitable in smart cities composed of many battery-operated nodes that have limited processing capabilities. In this case, the usage of periodic service messages (such as HELLO messages) is unsuitable since they require high bandwidth and high memory allocation.

Reactive protocols are designed to restrict the bandwidth consumed by service packets by removing the periodic service messages [38]. Routes are generated on-demand when a node starts a new communication. Dynamic Source Routing (DSR) [18] is one of these reactive protocols. In the beginning, a node performs route discovery in order to reach the destination node. The route discovery floods the network with route request packets. The destination node that receives this type of packet responds by sending a route reply packet back to the source. The source node stores the paths used to reach the destination. Link fault is recognized when the node tries a path that involves it. The maintenance of these paths does not include service messages to update the routing tables. The major limitation of this protocol is the absence of a local routing maintenance mechanism for a broken link. Therefore, flooding is also required when the sender has no suitable path to reach a destination node. Such a process may restrict the bandwidth available in a dense network. The path found using the discovery process, may also not necessarily be the shortest one in between the given pair of source and destination nodes. Ad hoc On-Demand Vector

(AODV) [39] is quite similar to the DSR. Both protocols find routes through flooding. However, DSR stores the routes in the source node while AODV distributes and stores the routing information in the entire network. While AODV scales with a large population of nodes, the size of routing tables (stored in intermediate nodes) increases as well. This augmentation makes the protocol unsuitable for networks composed of devices with a limited amount of memory. In [40], the authors proposed an AODV extension. The protocol performs route discovery by partially flooding the network. Topology information is sent back in the response packet after a specific amount of time. The source node calculates the shortest path to reach a specific destination by minimizing the packet reception rate and the number of hops. Their simulation results improve the AODV packet reception rate when harsh channel conditions are considered. The packet reception rate depends on the distance between nodes and the nominal transmission range. While the latter is a known value, the former is fixed at the beginning or computed by exchanging the geographical coordinates. The optimal reactive routing protocol (ORRP) [41] is a reactive protocol that finds the shortest path between the source and destination nodes in a distributed fashion. The shortest path is calculated using the Dijkstra algorithm. The protocol assumes symmetric links between neighboring nodes and that each node maintains neighboring and costs links. In [42], authors propose a proactive extension of this protocol by introducing a periodic HELLO message exchange for sensing neighborhood as well as for determination of cost list. However, this protocol generates a huge overhead when the network is dense.

Various routing approaches specifically designed for WM-Bus networks have been proposed. In [43], the study presented a model routing extension for the Wireless M-Bus Q-mode in TinyOS[3]. In [44, 45] the authors proposed an energy-aware routing that is implemented as an extension to the EN13757 Wireless M-Bus Q-mode [7]. A cost-efficient integrated energy harvesting system powered by the available water flow was developed to enable operation independent from the main grid. This eliminates the need for battery replacement with near-zero maintenance costs. A noteworthy idea for routing in low power-lossy link wireless networks comes from the study presented in [46]. The proposed routing considers metrics involving the residual energy and packet reception rate of neighbor nodes in ISA100.11a industrial wireless networks. In [47], the authors proposed a routing protocol for Wireless Mesh Networks by modifying the AODV routing protocol. The proposed protocol considers the load of nodes and link quality.

These studies show the need for a lightweight reactive routing protocol tailored for dense smart metering network which improve reliability and does not require any extra service messages. This thesis presents NARUN and NARUN-PC protocols. Noise Adaptive Routing for Utility Networks protocol (NARUN) [19, 20] is a routing protocol for dense mesh utility networks. In NARUN, the collector calculates the path to reach a destination meter by minimizing the link failure rate and the path length. A

[3](http://www.tinyos.net/)

weighted network graph represents the network topology and is stored in the collector node. No control messages are used to keep the weights updated. Meters reports back to the collector, link failure index with ordinary reading messages. NARUN-PC [21] is an extension of NARUN that introduces a path cache in the collector and therefore, avoids the selection of unstable links. These protocols have been evaluated using a simulator that has been tested against small test networks. The evaluation takes into consideration, the failure rate measured at the collector node, the number of sensor reads and the traffic load in terms of the number of messages successfully received.

Chapter 3

NARUN

This chapter provides a detailed model of Noise Adaptive Routing for Utility Networks protocol (NARUN), a routing protocol for WM-Bus applications in the context of smart cities. Section 3.1 covers the structure of a NARUN network, the communication protocol, and the collector and meter behavior. Section 3.3 describes the WM-Bus protocol and, subsequently, how the NARUN protocol extends this protocol in detail. Finally, this chapter provides the simulation setup in section 3.4, and then the chapter discusses and summarizes the simulation results in section 3.5. The last section 3.6 summarizes the chapter content.

3.1 The NARUN protocol

This section introduces some notation that is needed to describe the network model and the protocol. The set of all network nodes is denoted as $n_0, \ldots, n_z$ with N, and the WM-Bus read request frame packet with M. M is composed of three fields: the header, the payload, and the footer which are denoted with $M.h$, $M.d$, and $M.o$, respectively. The frame structure is described in section 3.3.1.

$G(N, E, w)$ denotes a network graph where (i) N is the set of nodes; (ii) $E \subseteq N \times N$ is a finite set of links, $(n_i, n_j) \in E$ when n_i can directly communicate with n_j; (iii) $w : E \longrightarrow R$ is an edge weight function.

Three weight functions has been defined: (i) constant; (ii) connection-based; and (iii) Hamming-based. The constant weight function, $w^c(n_i, n_j)$ assigns to each link (n_i, n_j) a constant number 1. This can be used to find the path with the least number of hops from the collector to any meter on the network. The connection-based weight function, $w^r(n_i, n_j)$ is ∞ when the link (n_i, n_j) is not working, and 1 otherwise. As shown in the next section, w^r can be used by the collector node to avoid broken links or faulty nodes.

The Hamming-based weight function, $w^h(n_i, n_j)$ is calculated by using the Hamming code. The header $M.h$ and the payload $M.d$ of a frame M are divided into l equal parts $M_1, \ldots, M_l$. The sender of the packet M calculates the Hamming code $hmc(M_i)$ of each part M_i and adds it into the footer M_o. The format of the message (when the Hamming-based weight function is used) can be summarized as follows [1]:

[1] For the sake of simplicity, the footer only shows the Hamming code part.

$M_t = M_1||\ldots||M_l||hmc(M_1)||\ldots||hmc(M_l)$. The equation 3.1 define the correction function $R(M_i)$.

$$R(M_i) = \begin{cases} 0 & M_i \text{ IS RECEIVED WITH NO ERROR} \\ \infty & M_i \text{ HAS A NON RECOVERABLE ERROR} \\ 1 & M_i \text{ HAS A RECOVERABLE ERROR} \end{cases} \tag{3.1}$$

The Hamming-based weight function $w^h(n_i, n_j)$ is defined by the equation 3.2.

$$w^h(n_i, n_j) = \begin{cases} \infty & \text{IF } \sum_{i=0}^{l} R(M_i) = \infty \\ \frac{\sum_{i=0}^{l} R(M_i)}{l} + 1 & otherwise \end{cases} \tag{3.2}$$

In other words, $w^h(n_i, n_j)$ is set to ∞ when M has a part M_i with no recoverable error. Otherwise, w^h is equal to one or two. It is one when no error is recovered, while it is two when each part M_i has a recoverable error. Effectively, $w^h(n_i, n_j)$ measures the link failure index lfi between n_i and n_j. In the next section, the weight function w^h can be used by the collector node not only to avoid faulty links or faulty nodes but also to select the path with the lowest link failure index.

In the rest of the chapter, we denote with $P_{n_k}(G) = \{n_0, \ldots, n_k, n_{k+1}, \ldots, n_0\}$ (with $k > 0$) a path that starts from the collector node n_0, reaches the node n_k and returns to the node n_0. This can be used by the collector to read the node n_k. The cost of the path can be calculated as follows:

$$W(P_{n_k}(G)) = \sum_{i=0}^{i=k-1} w(n_i, n_{i+1})$$

Symmetric links are assumed; which means that the path P_{n_k} will be palindromic (i.e. the paths from n_0 to n_k and n_k to n_0 are the same). In fact, WM-Bus links are rarely asymmetric.

In the rest of the chapter, $G_{n_0}(N_0, E_0, w_0)$ denotes the network graph of the collector node. Each meter node n_s also defines a meter network graph (from here onward, referred to as projection graph) which is denoted by $G_{n_s}(N_s, E_s, w_s)$. N_s is the set of neighbors of n_s that is $n_i \in N_s$ when n_i is in the communication range of n_s. The set E_s contains communication edges of the type (n_s, n_j) and (n_i, n_s) where the node n_s takes the role of sender and receiver, respectively. A projection graph is used by the meter n_s to store its local network view that is all its neighbors and the failure index of the related links. As we are going to see in the following, NARUN keeps the collector graph updated by performing an efficient merge of all projection graphs.

This thesis assumes that the collector node keeps a global timestamp t that is a sequence number. The collector node increases t by one when a meter reading is attempted. It is assumed that the collector node n_0 reads all meter nodes $n_1, \ldots, n_z$ in turn. A timestamp t_q denotes the q^{th} attempt that is made by the collector to read a meter n_k. Two consecutive timestamps t_q and t_{q+1} may be related to the same

meter node n_k when the routing path selected at time t_q fails to reach n_k. In this case, a different path can be tried at time t_{q+1}.

In the rest of the chapter, $G_{n_k,t_q}(N_{k,t_q}, E_{k,t_q}, w_{k,t_q})$ denotes a meter graph last updated at time t_q. Each $n_j \in N_{k,t_q}$ is a neighbor of n_k added at time t_u with $t_u \leq t_q$. The link $(n_k, n_j)_{t_u}$ would also be added to E_{k,t_q} at time t_u. For the sake of simplicity, the thesis does not include the algorithms for adding new nodes into the network. This is done by broadcasting standard hello messages. The weight $w_{k,t_u}(n_k, n_j)$ of the link (n_k, n_j) denotes a weight that was updated at some time t_u with $t_u \leq t_q$.

3.1.1 Communication primitives and message format

The collector node uses a WM-Bus frame packet to read a meter node at time t_q by M_{t_q}. Then, the three fields; the header, the payload, and the footer are denoted by $M_{t_q}.h$, $M_{t_q}.d$, and $M_{t_q}.o$, respectively. The frame structure is described in section 3.3.1.

The payload $M_{t_q}.d$ contains a NARUN application layer message d that complies with a standardized PAYLOAD structure. More precisely, the payload D is defined as PAYLOAD D = {TYPE Y, NODE N, TIMESTAMP T, GRAPHLIST L, NUMERIC R, PATH P} where: (i) D.Y specifies the types of the message, that is either *REQ* or *REPLY* (the former defines a payload that contains a reading request sent by the collector to a meter while the latter is the reading returned value); (ii) D.N contains the meter n_k to be read (this is often referred to as the destination node); (iii) D.P contains the path P_{n_k} that leads to the destination node and back to the collector node; (iv) D.T contains the global timestamp t_q; (v) D.L contains a list of projection graphs G_{n_1,t_q}, ..., G_{n_h,t_q}, each graph G_{n_s,t_q} is added by the node n_s as the payload travels along the routing path; (iv)D.R contains the reading of the node D.N (if any).

The list of projection graphs $G_{n_1,t_q}, \ldots, G_{n_h,t_q}$ is used to update the collector node graph G_{n_0,t_q}. This field is the NARUN addition to ordinary WM-Bus packet to update the collector weights without extra messages. The algorithm 1 sketches the merge procedure. This considers each link (n_j, n_i) that is contained inside each projection graph G_{n_s,t_q} in the list. The procedure updates the weight of the collector link (n_j, n_i) when the one received from the meter has a fresher timestamp.

Algorithm 1 NARUN **MERGE** OF PROJECTION GRAPHS

```
procedure MERGE(GRAPHLIST L)
    for each G_{n_s,t_q}(N_{s,t_q}, E_{s,t_q}, w_{s,t_q}) ∈ L do
        for each (n_j, n_i) ∈ E_{s,t_q} do
            LET t_u BE THE UPDATE TIME OF w_{s,t_u}(n_j, n_i) ∈ G_{n_s,t_q}
            LET t_v BE THE UPDATE TIME OF w_{0,t_v}(n_j, n_i) ∈ G_{n_0,t_q}
            if t_v < t_u then                    ▷ true if the collector receives a fresher weight
                w_{0,t_v}(n_j, n_i) ← w_{s,t_u}(n_j, n_i)
            end if
        end for
    end for
end procedure
```

NARUN uses send and receive primitives. The SEND(N,M) primitive can be used to send a message M from a node to its neighbor N. Acknowledgements are used

in order to provide reliable one hop communication. An error is returned when the destination node cannot be reached after a pre-set number of communication attempts. The RECEIVE(N,M) primitive can be used by a node to receive a message M from its neighbor N. This blocks the node execution until the message is received or a timeout is generated. The SEND(N,M) communication primitive is used to define a more abstract NARUN send primitive that is SENDN (see Algorithm 2). This can be used by the node n_s to send the message m to its neighbor n_d. The link weight between n_s and n_d is updated according to the *weight* function that has been selected. More precisely, when n_s fails to send M to n_d the weight $w_{s,t_q}(n_s, n_d)$ is updated to infinity. This means that at current time t_q the link n_s, n_d is broken. Otherwise the function UPDATEFAILUREINDEX is executed which updates the weight $w_{s,t_q}(n_s, n_d)$ by considering the selected weight function that is constant; connection-based; or Hamming-based.

Algorithm 2 NARUN **SEND** PRIMITIVE primitive

```
1: procedure SendN(node n_s, node n_d, Graph G_{n_s,t_q}, Payload D)
2:     M ← generateFrame(D)
3:     Result ← send(n_d, M)
4:     if Result=Error then
5:         w_{s,t_q}(n_s, n_d) ← ∞                ▷ weight of link updated at time t_q
6:         return Failure
7:     else
8:         UpdateFailureIndex(w_{s,t_q}(n_s, n_d))   ▷ weight of link updated at time t_q
9:         return Success
10:    end if
11: end procedure
```

3.1.2 NARUN collector behavior

The collector node uses its network graph in order to read each sensor node. While the least cost path is obtained by using a simple Dijkstra algorithm, NARUN uses a novel routing protocol strategy to update the collector network graph weights. More precisely, unlike most of the protocols, NARUN does not use any keep-alive or any control messages to update variation in the link failure index. In NARUN, each node locally collects link failure index information in the form of projection graphs. This is performed by using meter nodes that continuously eavesdropped on surrounding communications and update their local projection graphs. When the collector node selects a routing path to read a sensor node, all nodes in the routing path will add their local projections to the reply message. The projections will be merged and used by the collector to update its network graph. This strategy avoids the use of keep-alive or any other control messages.

Algorithms of Figure 3 describe the collector behavior. This discards all messages that have an old timestamp. An external procedure that connects new nodes and disconnects leaving ones is assumed. WM-Bus uses hello and disconnect messages. The variable $G_{n_0,t}$ is assumed to be a global one that contains the collector network graph. Without loss of generality, It is assumed that the collector reads the data of all sensor nodes in turn. The procedure COLLECTORLOOP(INT M) performs a for

Algorithm 3 Collector n_0 behavior

Require: discard any received message with less than timestamp t
Require: update global variable $G_{n_0,t}(N_{0,t}, E_{0,t}, w_{0,t})$ with new connected/disconnected nodes

```
1: procedure CollectorLoop(int m)
2:     t ← 0
3:     for each node n_k inside cluster N_{0,t} do
4:         Result ← readOperation(n_k,max)        ▷ try at most max times to read the same sensor n_k
5:     end for
6: end procedure
7: procedure readOperation(node n_k, int m)
8:     i ← 0                                      ▷ counter for making sure that no more than max attempts are done
9:     toBeMerged ← false                         ▷ true when a unary graph must be merged with G_{n_0,t}
10:    G1_{n_0,t} ← G_{n_0,t}                     ▷ A reference to collector graph G_{n_0,t} is put into G1_{n_0,t}
11:    res ← null
12:    do
13:        P_{n_k} ← Dijkstra(G1_{n_0,t},n_k)     ▷ best path P_{n_k} = {n_0, ..., n_k, ..., n_0}
14:        if P_{n_k} is not empty then           ▷ A path leads to n_k
15:            t ← t + 1                          ▷ timestamp incremented when a read is attempted
16:            i ← i + 1
17:            res ← ReadMeter(n_k,G1_{n_0,t},P_{n_k})
18:            if res ≠ Failure then              ▷ a value was read
19:                Break
20:            end if
21:        else
22:            G1_{n_0,t} ← Clone(G_{n_0,t},1)    ▷ a unary graph
23:            toBeMerged ← true
24:        end if
25:    while i ≤ m                                ▷ Try all possible paths to n_k
26:    if toBeMerged then                         ▷ read a value
27:        Merge(G1_{n_0,t})
28:    end if
29:    return res
30: end procedure
```

loop (lines 3-5) that reads all sensor nodes in turn. This is performed by using the procedure readOperation(node n_k, int m) that tries to read each sensor n_k at most max times. More precisely, when a reading attempt fails, readOperation will try again until a read is performed or the max number of attempts max has been reached. The result of line 4 can be either the value read or the error Failure. The lines 7-30 of Figure 3 describes the code of readOperation. The do-while cycle of lines 12-25 will try at most m routing paths to read the meter n_k. More precisely, the shortest path P_{n_k} is selected by considering the current network graph $G1_{n_0,t}$ (line 13). When a path is found, the procedure ReadMeter is called (line 17). This procedure tries to reach the node n_k by using the selected path. In case n_k is successfully read, the do-while cycle terminates (line 19), otherwise another path is tried. The variable P_{n_k} is empty when each path that leads to n_k has a link with infinity weight. In this case, lines 22-23 are executed. These set the pointer of the temporary variable $G1_{n_0,t}$ to a unary clone of the global network graph $G_{n_0,t}$. $G1_{n_0,t}$ is a copy of $G_{n_0,t}$ that has the same edges, nodes, and timestamps but all the weights set to one. The unary graph $G1_{n_0,t}$ allows the collector to retry paths that have an infinity link weight in the global network graph $G_{n_0,t}$. When the unary graph is used, the merge algorithm 1 is used to update the weights of the network graph $G_{n_0,t}$ with the new discovery weights from the unary graph (line 27). The procedures terminate by providing the result that is either the sensor reading (line 19 was executed), or the error Failure (max paths were tried but none of them reached the sensor), or null (no path exists).

This last option should never happen since it is assumed that each node connects and disconnects correctly from the collector.

Algorithm 4 Reading procedure of the collector node n_0

```
1: procedure ReadMeter(node n_k, Graph G_{n_0,t}(N_{0,t}, E_{0,t}, w_{0,t}), Path P_{n_k})
2:     Payload d                ▷ {Type y, node d, TimeStamp t, GraphList l, Numeric r, Path p}
3:     d.y ← req                                              ▷ WM-Bus request of reading
4:     d.n ← n_k
5:     d.p ← P_{n_k}
6:     d.t ← t                                  ▷ sequence number of attempted reading
7:     d.l ← ∅                                  ▷ local-graph set l = {G_{n_1}, ..., G_{n_h}}
8:     Result ← SendN(n_0, n_1, G_{n_0,t}, D)
9:     if Result = Failure then
10:        return Failure                       ▷ Collector cannot contact next hop n_1
11:    end if
12:    if TimeOut receive(n_1, M) then          ▷ Reply message not received
13:        w_{0,t}(n_1, n_0) ← ∞                ▷ Message was not successfully sent
14:        return Failure                       ▷ No path leads to the node
15:    end if
16:    Merge(G_{n_0,t},M.L)                     ▷ Update graph G with received local-graphs
17:    if Contains(n_k, M.L) then               ▷ Data value received from n_k
18:        return M
19:    else
20:        return Failure
21:    end if
22: end procedure
```

Algorithm 4 outlines the collector node's reading procedure. The collector node reads the sensor value of a node n_k by taking into account the collector network graph G_{n_0}. The procedure starts by defining the payload. This contains the type of message (in this case, a request *REQ* of reading), the destination node, the routing path, the timestamp, and the list of projection graphs to empty. A local projection graph G_{n_i} is added to the reply list by any node in the path in order to keep the weight of the collector node graph updated. The collector uses the sendN communication primitive in order to forward the request to the node n_1. This is the next node in the routing path. When the sending fails, the collector node returns an error (in this case, sendN updates the weight of the link between n_0 and n_1 to infinity); otherwise, the collector waits for the reply message (line 12 of the algorithm). When no *reply* is received the collector node assumes its communication link to the first node of the path (i.e., n_1) is not working; thus the collector sets the weight of the link $w_{0,t}(n_1, n_0)$ to infinity and returns an Failure error. When a reply value is returned, the collector calls the Merge function in order to update its local network graph G_{n_0} with the list of projection graphs $M.L = \{G_{n_1,t}, \ldots, G_{n_h,t}\}$. The list of projection graphs is further analyzed. More precisely, when the projection graph G_{n_k} of the node to be read (i.e., n_k) is in the list, it means the data was successfully read and the reply message M is returned. When the projection graph G_{n_k} is not in the list, it means the data was not successfully read, and an Failure error is returned.

3.1.3 Meter behavior

A meter n_s uses an eavesdrop procedure to sniff each message that is exchanged between neighbors (see Algorithm 5 for details). More precisely, this is a message that

is sent from n_j to n_i with $n_i \neq n_j \neq n_s$. This indirect observation allows n_s to update the link failure index of its projection graph.

Algorithm 5 NARUN eavesdrop routine

```
1: procedure EAVESDROP(NODE n_s, GRAPH G_{n_s}(t)(N_s, E_s, w_s))
2:     MESSAGE m
3:     while true do
4:         SNIFF(M)                ▷ sent from n_j to n_i with n_i ≠ n_j ≠ n_s and n_j ∈ E_s
5:         if MESSAGE FROM n_j TO n_i IS READABLE then
6:             UPDATEFAILUREINDEX(w_{s,m.T}(n_j, n_s))
7:         end if
8:         if ACK FROM n_i TO n_j IS READABLE then
9:             UPDATEFAILUREINDEX(w_{s,m.T}(n_i, n_s))
10:        end if
11:    end while
12: end procedure
```

The Algorithm 6 shows the reading behavior of a meter n_s. This loops forever waiting for incoming payloads (line 3 of the algorithm). When a payload D is received from a node n_{s-1} the timestamp of the local graph $G_{n_s,t}$ is updated with the received one (i.e., D.T.) and the failure index of the link n_{s-1}, n_s is also updated. When the meter is the destination one (i.e., the one to be read), a read is performed and the type of the message is changed to reply (lines 9-10 of the algorithm). The local projection graph is added to the received payload (this is needed to update the collector node). Finally, the payload is forwarded to the next hop (line 14). When the next hop cannot be reached, the payload is updated with the new projection graph since the procedure SENDN would set the weight of the link (n_s, n_{s+1}) to infinity. The REPLY message is sent back to the previous node in the path. In this case, no reception of the message is checked.

Algorithm 6 Meter n_s reception behavior after bootstrap

```
Require: discard any message with less than t
Require: update G_{n_s,t}(N_s, E_s, w_s) with new connected/disconnected nodes
1: procedure METERRECEIVE
2:     PAYLOAD D                ▷ {TYPE Y, NODE D, TIMESTAMP T, GRAPHLIST l, PATH p}
3:     while true do
4:         WAITRECEIVE(n_{s-1}, D)          ▷ sent by the previous node in the routing path
5:         G_{n_s,t}(t ← D.T)
6:         UPDATEFAILUREINDEX(w_{s,d.T}(n_{s-1}, n_s))
7:         if D.Y=REQ then                  ▷ A request from collector
8:             if D.N = n_s then            ▷ destination reached
9:                 D.R ← READ()             ▷ fill the reply with my reading
10:                D.Y ← REPLY              ▷ set message type to reply
11:            end if
12:        end if
13:        D.L ← D.L ⋃ G_{n_s,t}            ▷ add local graph
14:        n_{s+1} ← GETNEXTHOP(D.P)        ▷ get the next hop
15:        if SENDN(n_s, n_{s+1}, D)≠ SUCCESS then   ▷ when the next hop is unreachable
16:            D.Y ← REP                    ▷ a reply is sent back
17:            D.L ← D.L ⋃ G_{n_s,t}        ▷ add local graph
18:            SENDN(n_s, n_{s-1}, D)
19:        end if
20:    end while
21: end procedure
```

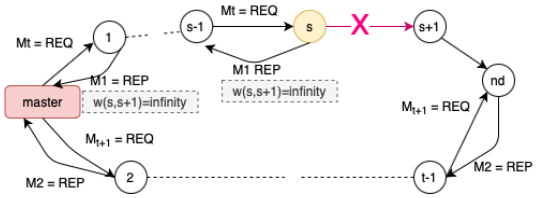

FIGURE 3.1: Updating a broken link

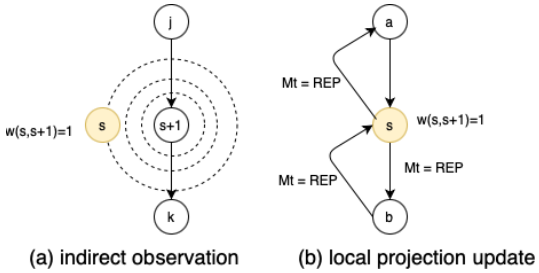

FIGURE 3.2: Indirect update of s by eavesdropping $s+1$ communication

3.1.4 NARUN connectivity

As we are going to see in the simulation Section, NARUN is suitable for dense networks that are composed of many links between meter nodes. This setting can be found in many applications, such as smart metering in smart cities. Dense networks ensure connectivity and fast reading since the collector network graph always has updated links. Figure 3.1 shows the detection of a broken link by the collector node. The collector node tries to read a node n_d at time t by using a message M_t. The first broken link in the path (the link $(s, s+1)$ in Figure 3.1) will cause a timeout and the sending of a reply back to the collector. This is used to update the collector graph with the broken link (i.e., $w_{s,t}(s, s+1) = \infty$ in the example of Figure 3.1). The collector will retry to read n_d by using an alternative path at time $t+1$ (see message M_2 of Figure 3.1). NARUN can update the weight of the link $(s, s+1)$ without the need of any extra service messages. This is performed in two steps that are (i) indirect observations, and (ii) local projection updates.

Indirect observations will eventually allow the detection of a link that starts functioning again. For instance, in the example of Figure 3.1, the collector node will eventually try to route a message via the nodes s and $s+1$. This must necessarily happen when the collector tries to read these nodes. For instance, the meter s can update the link failure index of $(s+1, s)$ when the routing path includes $s+1$ (see Figure 3.2). In this case, s can eavesdrop on the messages/ACK sent by $s+1$ to another node and update its local projection graph. This update can be sent back to the collector node when s is part of a routing path. This allows the collector to correctly update the link $(s, s+1)$. Symmetrically, $s+1$ can update the link failure index of $(s, s+1)$ when the routing path includes s. Similar observations can be done for a node n_s that is not working. In these cases, all links that have n_s as destination node will be detected as not working. NARUN will be able to detect that n_s is functioning

again when a path that goes via n_s will be tried.

We can easily prove that if a path leading to a node exists, the collector node will eventually find it. When the collector network graph is updated (through indirect observations), the collector node will eventually try the path (see the do-while cycle, lines 12-25 of Algorithm 3). When the collector network graph is not updated and all of the paths leading to the node appears to have been broken, the use of the unary graph of lines 22-23 of Algorithm 3 will force the collector to retry all possible paths until one is found working. While this behavior also updates link qualities, it can cause additional communication. As we are going to see in the simulation section, when the network is dense and the error is low, indirect observations always keep the collector network graph updated and the unary graph is rarely used. This can be proved by showing a low percentage of failing paths. These are paths the collector node believes lead to the destination node but they do not.

3.2 DSR model

NARUN has been simulated against the Dynamic Source Routing (DSR)[18] and WMBUS protocols. In this section, we present the DSR implementation compared to the NARUN protocol. The DSR nodes manage the same data structures previously described in section 3.1. The difference lies in the behavior of the collector and meter nodes. The algorithm 7 shows the collector behavior. The collector only updates the weights of the network graph when a routing path is discovered or fails. The weight of the edges in path p, are set to infinity when a reading that uses the path p fails (see lines 24-26 in the algorithm 7). The discovery starts when no paths have been found to reach the destination node (see lines 14-15 in the algorithm 7). The discovery process is implemented by broadcasting HELLO messages in the network (see the algorithm 8). The $HELLO$ message D includes the final destination node $D.N$ and a path $D.P$ that contains the node identifiers crossed by the packet (see lines 3-6 in the algorithm 8). The packet also includes a sequence number $D.T$ used to avoid loop and multiple retransmissions. The message is sent to all neighborn through the method $SendB$. The packet is then broadcast by them until it reaches the destination node. If a node receives an $HELLO$ message that was previously received (can be checked using the timestamp $D.T$ and t), the packet is discarded. The collector waits for the reception of multiple $HELLORESP$ messages by meters (see line 8 in the algorithm 8). Each $RESPHELLO$ message is processed by setting the weight edges related to the path (embedded in the $RESPHELLO$ message) with the value one (see lines 12-14 in the algorithm 8). Once at least one path is found, the collector retries to look at the shortest path (see line 16 in the algorithm 7). The algorithm tries to read the meter if a path is found (see lines 19-28 in the algorithm 7). A new $RESPHELLO$ is forwarded back to the collector when the destination meter $D.N$ receives the $HELLO$ packet. Each $RESPHELLO$ includes the path $M.p$ which is the path that leads to the destination node. The $RESPHELLO$ and REQ messages do not include the

local graph or any type of update (see algorithm 9) which means that meters simply forward response messages and does not eavesdrop on messages around (see lines 15 and 16 of algorithm 9).

Algorithm 7 Collector n_0 behavior with DSR routing protocol

Require: discard any received message with less than timestamp t
Require: update global variable $G_{n_0,t}(N_{0,t}, E_{0,t}, w_{0,t})$ with new connected/disconnected nodes
1: **procedure** CollectorLoop(int m)
2: $t \leftarrow 0$
3: **for** each node n_k inside cluster $N_{0,t}$ **do**
4: Result ← readOperation(n_k,max) ▷ try at most max times to read the same sensor n_k
5: **end for**
6: **end procedure**
7: **procedure** readOperation(node n_k, int m)
8: $i \leftarrow 0$ ▷ counter for making sure that no more than max attempts are done
9: res ← null
10: **do**
11: P_{n_k} ← Dijkstra($G_{n_0,t}$,n_k) ▷ best path $P_{n_k} = \{n_0, \ldots, n_k, \ldots, n_0\}$
12: $t \leftarrow t + 1$ ▷ timestamp incremented when a read is attempted
13: $i \leftarrow i + 1$
14: **if** P_{n_k} is empty **then** ▷ No paths that leads to n_k are found
15: **if** discoveryPaths(n_k,$G_{n_0,t}$) is not empty **then** ▷ New paths found
16: P_{n_k} ← Dijkstra($G_{n_0,t}$,n_k)
17: **end if**
18: **end if**
19: **if** P_{n_k} is not empty **then** ▷ A path leads to n_k
20: res ← ReadMeter(n_k,$G_{n_0,t}$,P_{n_k})
21: **if** res $\neq$ Failure **then** ▷ a value was read
22: **Break**
23: **else** ▷ the path is invalid
24: **for** each $(n_j, n_i) \in P_{n_k}$ **do**
25: $G_{0,t_v}(n_j, n_i) \leftarrow$ inf
26: **end for**
27: **end if**
28: **end if**
29: **while** $i \leq$ m ▷ Try all possible paths to n_k
30: **return** res
31: **end procedure**

Algorithm 8 Discovery procedure of the collector node n_0

1: **procedure** DiscoveryPaths(Numeric n_k,Graph $G_{n_0,t}(N_{0,t}, E_{0,t}, w_{0,t})$)
2: **Payload** d ▷ {Type y, node d, TimeStamp t, GraphList l, Numeric r, Path p}
3: d.y ← HELLO ▷ DSR discovery
4: d.n ← n_k
5: d.p ← [0]
6: d.t ← t ▷ sequence number of attempted discovery
7: Result ← SendB(n_0, D)
8: **if TimeOut** multiReceive(H) $\wedge$ $H = \emptyset$ **then** ▷ Wait reply messages
9: **return** Failure ▷ No path found that leads to the destination
10: **end if**
11: **for** each $M \in H$ **do** ▷ Adds discovered paths
12: **for** each $(n_j, n_i) \in M.p$ **do**
13: $G_{0,t_v}(n_j, n_i) \leftarrow 1$
14: **end for**
15: **end for**
16: **return** H
17: **end procedure**

Algorithm 9 Meter n_s reception behavior after bootstrap in the DSR protocol

Require: discard any message with less than t
Require: update $G_{n_s,t}(N_s, E_s, w_s)$ with new connected/disconnected nodes
1: **procedure** METERRECEIVE
2: **PAYLOAD** D ▷ {TYPE Y, NODE D, TIMESTAMP T, GRAPHLIST l, PATH p}
3: **while** true **do**
4: WAITRECEIVE(n_{s-1}, D) ▷ sent by the previous node in the routing path
5: $G_{n_s,t}(t \leftarrow$ D.T)
6: **if** D.Y=REQ **then** ▷ A request from collector
7: **if** D.N $= n_s$ **then** ▷ I am the destination
8: D.R $\leftarrow$ READ() ▷ fill the reply with my reading
9: D.Y $\leftarrow$ REPLY ▷ set message type to reply
10: **end if**
11: **end if**
12: **if** D.Y=HELLO **then** ▷ Send broadcast message
13: RESULT $\leftarrow$ SENDB(n_s, D)
14: **else** ▷ HELLORESP, REQ, RESP messages are sent to the next hop
15: $n_{s+1} \leftarrow$ GETNEXTHOP(D.P) ▷ get the next hop
16: **if** SENDN(n_s, n_{s+1}, D)$\neq$ SUCCESS $\wedge$ D.Y=REQ **then** ▷ fault detection
17: D.Y $\leftarrow$ *REP* ▷ a reply is sent back
18: SENDN(n_s, n_{s-1}, D)
19: **end if**
20: **end if**
21: **end while**
22: **end procedure**

3.3 Implementation

This section describes the standard link layer of the WM-Bus. More precisely, the section illustrates the node discovery process and the communication behavior of WM-Bus nodes. Afterward, this section presents NARUN extensions, which include hamming encoding, an updated node discovery algorithm, and other protocol details.

3.3.1 WM-Bus protocol

The WM-Bus protocol is a European standard for smart metering sensor networks. The protocol uses some of the layers of the OSI 7 Layer Model. Figure 3.3 shows the comparison between the OSI layers and the WM-Bus protocol. The application layer describes the data transmitted by meters [5]. The CI-field describes the format and type of data field included in the payload, as we can see in Figure 3.6. A data header can also be included to encrypt and describe the data field. For the sake of simplicity, the thesis does not include this header in simulations. The EN 13757-7 standard covers the wireless M-BUS transport and service layer [8]. However, this standard is rarely used. WM-Bus supports star, tree, or mesh topologies [7]. In star topology, meters communicate with the collector, whereas in a tree topology, meters are grouped by a gateway node. Each gateway forms its network. The mesh topology provides routing capability to meters. Communication can be achieved by including the routing path in the payload or using the internal knowledge of the node (like the RIP protocol [48]). The standard does not specify any routing protocol. In this context, the proposed NARUN provides routing functionalities by using the mesh type of communication.

The WM-Bus data link and physical layers are specified by EN 13757-4 standard [6]. The physical layer specifies the physical parameters such as frequencies, preamble,

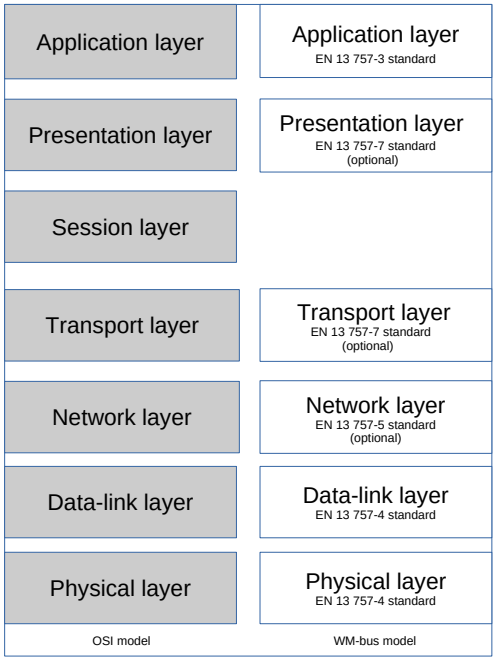

FIGURE 3.3: The OSI Layered Model and the WM-Bus model

and postamble length, hardware connections, and cable types, the maximum transmit power, the node sensitivity, and other details. The addressing, the communication protocol, and the reliability are managed by the data-link layer.

A primary or secondary address identifies WM-Bus Nodes. The former is a byte long, and the addresses from 1 to 250 may be assigned to meters. The broadcast packet uses the addresses 255 and 254. 251 and 252 are reserved and should not be used. When the former address has a value equal to 0 or 253, the secondary address is used. The secondary address is a unique address assigned by the manufacturer, composed of the Manufacture id (M-field, 2 bytes) and the address field (A-field).

Messages are encoded using two packet formats, namely, A and B. Both have a preamble and a postamble section. A single frame is divided into blocks. The former block is the header. Figures 3.4 and 3.5 show the header for format A and B both contain (i) the frame length byte (L-field), i.e., the frame size; (ii) the control information (C-field), i.e., a byte defining the message type (listed in tables 3.2 and 3.3); and (iii) the secondary address of the sender (M-field and A-field). In addition, format A includes the CRC field. The payload can be encoded with one or more sequential blocks. Figure 3.6 defines the block structure. The payload is composed of the following fields: (i) CI byte field, which specifies the data encryption (i.e., whether or not the data is encrypted) and the type of payload; (ii) the CRC-field that is related to the entire block; (iii) the data field, i.e., the payload sent by the application layer. The maximum size of a block is 15 bytes for the A frame type and 115 bytes for the B frame type. The type of payload described by CI-field are defined in tables 3.2, 3.3. These tables contain a list of packets that can be sent by the collector and meter nodes, respectively.

WM-Bus sum up this information using modes. In detail, each specifies uplink and downlink data rates, whether unidirectional or bidirectional, the supported frequency (which can be 169 MHz, 433 MHz, and 868.3 MHz), the encoding method (Manchester, 3 out of 6 encodings and NRZ), and the meter behavior. Table 3.1 shows the mode supported by WM-Bus.

These modes are designed for different scenarios. The mode S or Stationary mode (S1, S1-m, and S2) limits the number of readings per day. Frequent communication can be done in the T mode; meters periodically send data to the collector node. The interval can be set at as small as a few seconds to various minutes. Similar to the T mode, the Compact mode or C mode allows periodical transmission with NRZ encoding scheme and higher up-link bit rate. In T, S, and C modes are always the meter that starts the communication. The collector node is passive and reacts to the meter node messages. In the R2, F, P, and Q modes, the communication is bidirectional, and the collector node can request data from the meter nodes. In detail, messages can be relayed using modes P or Q. The Q mode allows retransmissions in a tree topology. The P mode implements routing also in a mesh topology.

TABLE 3.1: WM-Bus trasmission modes

Mode name	Frequencies	Direction	Encoding	Uplink	Downlink
S1	868.3,433 MHz	Unidirectional	Manchester	32.70 kbps	32.70 kbps
S1-m	868.3,433 MHz	Unidirectional	Manchester	32.70 kbps	32.70 kbps
S2	868.3,433 MHz	Bidirectional	Manchester	32.70 kbps	32.70 kbps
T1	868.3,433 MHz	Unidirectional	3 out of 6	100 kbps	32.70 kbps
T2	868.3,43 3MHz	Bidirectional	3 out of 6	100 kbps	32.70 kbps
C1	868.3,433 MHz	Unidirectional	NRZ	100 kbps	50 kbps
C2	868.3,433 MHz	Bidirectional	NRZ	100 kbps	50 kbps
R1	868.3 MHz	Unidirectional	Manchester	4.80 kbps	4.80 kbps
R2	868.3 MHz	Bidirectional	Manchester	4.80 kbps	4.80 kbps
F	433 MHz	Bidirectional	NRZ	2.40 kbps	2.40 kbps
N1	169 MHz	Unidirectional	NRZ	2.40/4.80 kbps	2.40/4.80 kbps
N2	169 MHz	Bidirectional	NRZ	2.40/4.80 kbps	2.40/4.80 kbps
N1g	169 MHz	Unidirectional	NRZ	19.20 kbps	19.20 kbps
N2g	169 MHz	Bidirectional	NRZ	19.20 kbps	19.20 kbps
P	868 MHz	Bidirectional	NRZ	4.80 kbps	4.80 kbps
Q	868 MHz	Bidirectional	NRZ	4.80 kbps	4.80 kbps

In the aforementioned modes, nodes have a transmission range of hundreds of meters. The N mode was introduced in order to overcome this limitation. In this case, meters use narrow-band communication in the 169 MHz frequency band. The communication range can reach 10 km.

In a WM-Bus network, network initialization occurs by broadcasting the SND-IR packet. Subsequently, the collector node can acknowledge (using SND-NKE packet) or refuse the new node (CNF-IR packet). Data communications can start after the collector node acknowledges.

Table 3.2: WM-Bus collector packet

Alias	Explanation	Response from collector
SND-NKE	Complete communication (No ACK required)	-
SND-UD2	Send packet with subsequent previous request (REQ-UD2)	RSP-UD, NACK
SND-UD	Send packet to the meter	ACK, NACK
REQ-UD	Request alarm data	RSP-UD
REQ-UD2	Request data	RSP-UD
ACK	Acknowledge the delivery of a packet	-
CNF-IR	Confirms the successful installation of meter/actuator into this gateway	-

Table 3.3: WM-Bus meter packet

Alias	Explanation	Response from collector
SND-NR	Send spontaneous/periodical application data without request	-
SND-IR	Install a new meter in the network	CNF-IR, SND-NKE
ACC-NR	Access demand to collector in order to request new data, without reply.	-
ACC-DMD	Access demand to collector in order to request new important application data (alerts)	ACK
ACK	Acknowledge the delivery of a packet	-
NACK	Notify the collector that the meter reception leads to a buffer overflow or invalid CI field	-
RSP-UD	Responds with a message containing user data.	-

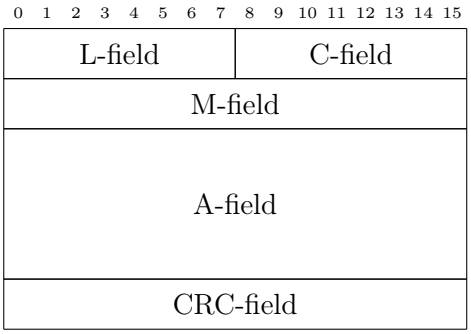

Figure 3.4: WM-Bus Header block format A

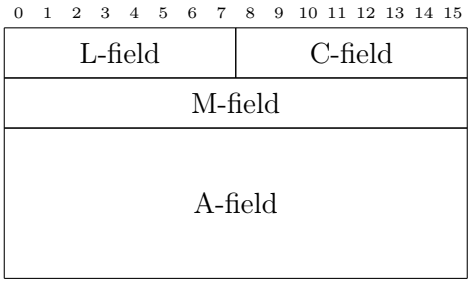

FIGURE 3.5: WM-Bus Header block format B

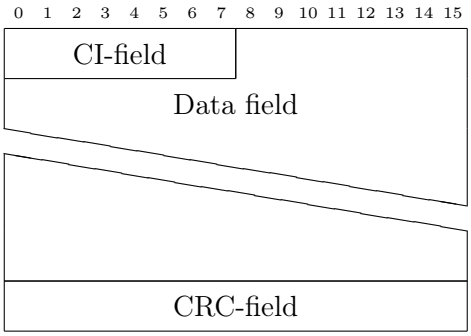

FIGURE 3.6: WM-Bus Data block format A/B

3.3.2 NARUN routing protocol

NARUN is a routing protocol that extends and modifies the standard WM-Bus protocol by providing the network layer in the OSI model.

NARUN modifies addressing and implements payload reliability using hamming instead of CRC code.

With Hamming, every seven bits have four data and three parity bits. This method can check the packet readability and recover one bit whether a bit changes in the transmission. For the sake of simplicity, simulations consider 1 byte for addressing nodes. Nodes are identified using the primary addressing index (PAI). Its size is 1 byte and can range between 0 to 255. These addresses are unique and fixed at the beginning.

The collector can spontaneously request accounting information using the meter identifier. These messages are relayed in a mesh topology to reach the destination node.

The packet format of NARUN is composed of the following fields: (i) the source primary address (SPA-field), which represents the source hop address, destination primary address (DPA-field), the transmit field (T-field), which specifies the communication type (unicast, broadcast, and multicast) and the data field as we can see in Figure 3.7.

In the beginning, meters need to be discovered by the collector node. To do this, meters spread a broadcast packet called HELLO packet to discover the neighbors. Recipients answer this packet with a HELLORSP packet. This packet has an empty payload. In this phase, meters initialize their local graph. Meters broadcast their

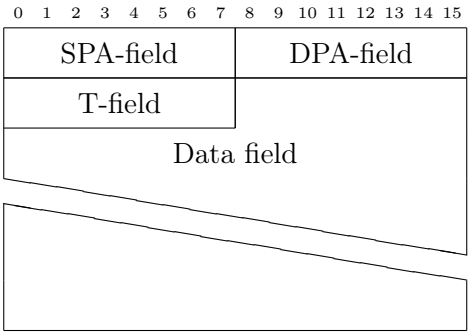

FIGURE 3.7: NARUN Packet format

identifier and neighbor list with the SND-IR packet. Subsequently, recipients append their identifier in the list of hops, (ii) increases the hop size, and (iii) broadcasts the new packet. The same procedure can also be used for meter joining.

Figure 3.8 shows the payload structure of SND-IR packet. The SND-IR packet is composed of: (i) the meter identifier, (ii) the list of hops that reply to the packet, (iii) the size of the hop list (Hop count field), (iv) the list of neighbors and (v) the size of the neighbors list. The hop list can contain up to 255 nodes and cannot contain duplicates. This avoids long and cycle paths.

The collector receives SND-IR packets and initializes the network topology G. The graph G is stored in the form of an adjacency matrix, which associates source and destination indexes to the link failure index. The weight used is the discretized version of equation 3.2 (where the upper bound is 255).

After the discovery phase, the collector can request accounting information by using algorithm 4. The request timeout is $2 \cdot c \cdot |P_k|$ where c is the latency of the channel used. Figure 3.9 shows the request (REQ-UD2) packet implementation. The request contains the path used to reach the destination n_k. Response message (RSP-UD) is sent back to the collector node using the same routing path. Figure 3.10 shows the response (RSP-UD) structure. This packet contains the set of local graph $M.L$ along with the $M.R$ $(D-field)$ accounting information (whether it is provided). This graph contains new information about the failure index of the links. This structure is encoded as multiple lists. Each list refers to a source node. Each source has a set of destinations coupled with the link failure index.

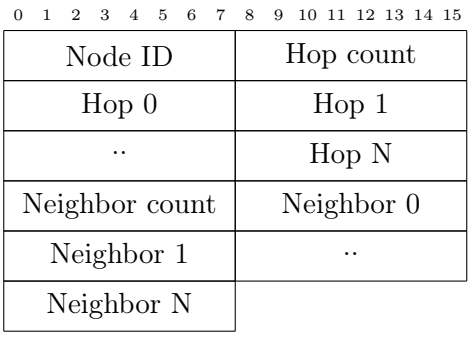

FIGURE 3.8: NARUN SND-IR payload

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Path size	n_0
n_1	..
n_k	

FIGURE 3.9: NARUN REQ-UD2 payload

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
DE-field	
D-field (Optional)	
Sources count(n)	Source 0
Dest. count(m)	Destination 0
lfi 0	Destination 1
lfi 1	Destination m
lfi m	Source n
Dest. count(s)	Destination 0
lfi 0	Destination 1
lfi 1	Destination s
lfi s	

FIGURE 3.10: NARUN RSP-UD payload

3.4 Simulation Setup

NARUN has been simulated in the San Paolo district, Camerino City (MC, Italy) where various performance measures have been evaluated. Figure 3.11 shows the considered smart meter network where the red and white markers represent the collector and meter nodes, respectively. The white lines represent connections between nodes. Simulations use a free space propagation model. The WM-Bus collector node is positioned in order to reach all meters by using a few hops. The characteristics of this topology are summarized in Table 3.4. Wider random topology have been tested in the paper [19]. The following input parameters are used for the simulations: (i) Noise power: The noise level in the network; (ii) ECC: A boolean parameter that defines the error correction code used by the simulation (i.e., ECC=true when Hamming is used and ECC=false for CRC); (iii) Routing: A boolean parameter that determines whether the meter includes the link failure index lfi in a response packet.

Simulations consider a specific configuration for the physical model. WM-Bus supports frequencies of 169MHz (Narrowband), 433 MHz, and 868 MHz. The frequency of 169 MHz is a novel frequency that allows distance up to 10 km. The frequency of

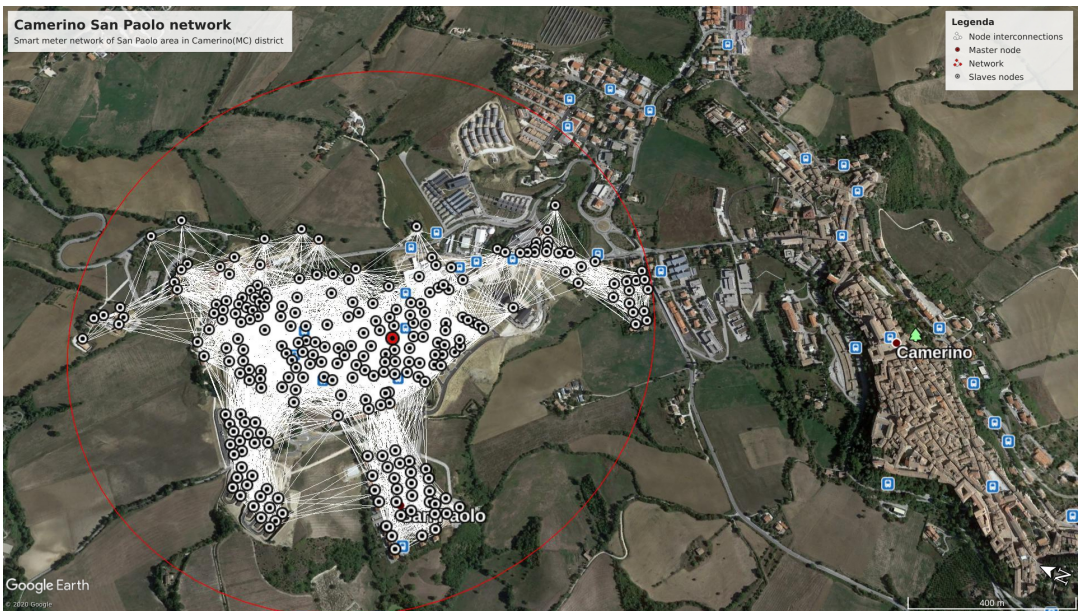

FIGURE 3.11: Camerino topology.

433 MHz instead is widely used in Europe [9] and reaches up to a few hundred meters of distance. The frequency of 433 MHz is intended for sectors where 868MHz is not allowed. Each frequency has different modes that specify the communication baud rate and transmitter capabilities. Simulations consider the widely used frequency of 868 MHz. For channel modeling, Additive white Gaussian noise and the free space model have been chosen. Furthermore, the transmitter is assumed to use Frequency Shift Keying (FSK) modulation, 10 dBm for transmission, and an antenna with 0 dB gain. Equation 3.3 shows the path loss computation formula, which depends on the frequency f (expressed in megahertz or MHz), the distance between the source and the destination d (expressed in meters), and the gain of antenna g (expressed in dBi).

$$Pathloss_{dB} = 20 \times \log_{10}(d) + 20 \times \log_{10}(f) - 27.55 - g \tag{3.3}$$

With the path loss value, we can compute the signal to noise ratio at the receiving end and the bit error rate as we can see in equation 3.4.

$$BER = \frac{1}{2} \times erfc(\sqrt{(}\frac{SNR_{rec}}{2})) \tag{3.4}$$

The bit error rate value determines the success of communication. Equations 3.5, 3.6, and 3.7 show respectively the probability of a success, a recoverable, or unrecoverable packet by assuming a binary symmetric channel [49]. In these equations, n represents the size of the packet, whereas r is the bit error rate.

$$P(successful\ packet) = (1-r)^n \tag{3.5}$$

$$P(recoverable\ packet) = n \times r \times (1-r)^{n-1} \tag{3.6}$$

$$P(fail\ packet) = 1 - (1-r)^n - n \times r \times (1-r)^{n-1} \tag{3.7}$$

3.4.1 Assumptions and simulation methodology

This section summarises the simulation methodology and all assumptions. The basic unit of our experiments is a collector reading attempt. This is done by using the ReadMeter procedure of Figure 4. A reading attempt that is related to the sensor i is denoted with a^i. The attempt a^i is a random variable that is equal to Failure (in the following denoted with 0) when the collector fails to read the sensor i, Success (in the following denoted with 1) otherwise. The DSR attempt may include the discovery of new paths (this is done via broadcasting). These additional messages are sent when no path is known or all available paths failed.

A collector reading operation is a finite sequence of reading attempts $r^i = a^i_1 a^i_2 \dots a^i_k$ ($0 < k < max$) where max is the maximum number of consecutive attempts the collector performs on the same sensor. The details of the readOperation are described in the algorithm 3. These models that when a collector reading attempt fails, the collector will try again until a read is performed or a max number of attempts has been reached. The maximum number of attempts in simulations is ten which is near the number of retries managed by uIP TCP, an extremely small implementation of the TCP/IP protocol suite [50]. Hence, a reading operation failure is a sequence that contains all zeros. A successfully reading operation is any sequence of zeros (also an empty one) that ends with a one (the length cannot exceed max). Equation 3.8 formally defines the reading operation failure rate when reading the sensor i.

$$F(r^i) = \frac{\sum_{k=0}^{max}(1 - a^i_j)}{max} \tag{3.8}$$

$F(r^i)$ measures the number of failed attempts during the reading operation of the sensor i. As we are going to see, in the following a higher failure rate corresponds to a higher number of network messages. Equation 3.9 formally defines the successful reading operation rate when reading the sensor i.

$$O(r^i) = \frac{\sum_{k=0}^{max} a^i_k}{max} \tag{3.9}$$

This can be either zero or one. It is worth mentioning that a 100% reading rate does not imply a 0% failure rate. For instance, in the sequence of attempts 0000000001 we have 90% failure rate and 100% reading rate.

We can now formally define a collector *round*. Suppose that our system is composed of z sensors. We denote with $N = \{n_1, \dots, n_z\}$ the set of sensors. In a round, a collector tries to read all sensors in turns by producing a sequence of z reading operations. The notation $R = \{r^1, \dots, r^z\}$ denotes a round where each r^i is the reading

operation on the sensor i. Equations 3.10 and 3.11 formally define the failure round rate and the reading round rate.

$$F(R) = \frac{\sum_{i=0}^{z} F(r^i)}{z} \tag{3.10}$$

$$O(R) = \frac{\sum_{i=0}^{z} O(r^i)}{z} \tag{3.11}$$

We can define a simulation run that is a sequence of h rounds. This is denoted with $U = \{R_1, \ldots, R_h\}$. Equations 3.12 and 3.13 formally define the failure run rate and the reading run rate.

$$F(U) = \frac{\sum_{i=0}^{h} F(R_i)}{h} \tag{3.12}$$

$$O(U) = \frac{\sum_{i=0}^{h} O(R_i)}{h} \tag{3.13}$$

An experiment is a sequence of q runs. This is denoted with $E = \{U_1, \ldots, U_q\}$. Equations 3.14 and 3.15 formally define the failure experiment rate $F(E)$ and the reading experiment rate $O(E)$.

$$F(E) = \frac{\sum_{i=0}^{q} F(U_i)}{q} \tag{3.14}$$

$$O(E) = \frac{\sum_{i=0}^{q} O(U_i)}{q} \tag{3.15}$$

We can finally define a simulation that is a sequence of p experiments. This is denoted with $S = \{E_1, \ldots, E_p\}$. Equations 3.16 and 3.17 formally define the failure rate $F(S)$ and the reading rate $O(S)$.

$$F(S) = \frac{\sum_{i=0}^{p} F(E_i)}{p} \tag{3.16}$$

$$O(S) = \frac{\sum_{i=0}^{q} O(E_i)}{p} \tag{3.17}$$

The experiments $S = \{E_1, \ldots, E_p\}$ are a sequence of independent and identically distributed random variables. The criteria used to stop a simulation have been presented in [51]. At each run t, all the t experiments are considered, that have been performed so far (i.e, $S_t = \{E_1, \ldots, E_t\}$) and we calculate $F(S_t)$ and $O(S_t)$. The simulation stops when for a sequence of k consecutive experiments (i.e., E_t, E_{t+1} ... E_{t+k}) the following conditions hold: (i) $|F(E_i) - F(E_{i+1})| < \epsilon$; (ii) $|O(E_i) - O(E_{i+1})| < \epsilon$ (with $t < i <$t+k). This stop criteria makes sure that the sample average converge within a threshold ϵ.

The following protocols have been compared: (i) WMBUS; (ii) ECC-WMBUS; (iii) NARUN; (iv) ECC-NARUN; (v) DSR. These are briefly described in the following.

TABLE 3.4: Network characteristics

Network characteristic	Value
Collector latitude	43.146 509 423 809
Collector longitude	13.061 599 661 224
Number of meter nodes	254 nodes
Node to node minimum distance	8.86 m
Node to node maximum distance	249.99 m
Node to node average distance	151.14 m
Node to node median distance	156.33 m
Node to node variance distance	3915.86 m^2
Collector to meter minimum path length*	1 hop
Collector to meter maximum path length*	4 hops
Collector to meter average path length*	2 hops
Collector to meter median path length*	2 hops
Collector to meter variance path length*	0.82 $hops^2$
Meter to meter average path length*	2.63 hops
Red circle radius	778.10 m
Node density in the red circle	134.28 nodes/km^2
Network density**	21.15 %

* Path lengths computed by finding the shortest path with the lowest number of hops.
** Network density is the percentage of the ratio between the actual and possible connections in between meters.

WMBUS routing method allows node forwarding by including the path in the request. The weight function is constant-based (see Section 3.1 for details). The path chosen by the collector minimizes the number of hops. Hop to hop communication uses the send described in the Algorithm of Figure 2. This uses CRC to detect erroneous data frames and tries re-transmission 4 times. ECC-WMBUS is similar to WMBUS one except for the ECC use. Hop to hop communications use Hamming forward correction code to validate and recover messages. This is emphasized for both WMBUS and ECC-WMBUS routing, the weights are never updated but are always set to 1.

NARUN uses a connection-based weight function (see Section 3.1 for details) while ECC-NARUN uses a hamming-based one. The algorithms are described in section 3.1. DSR algorithm had been implemented as described in [18].

3.5 Simulation results

This section provides the chosen simulation methodology along with the simulation results under different network conditions. Two kinds of simulations are described: (i) with disconnected links, a random subset of links is disconnected and (ii) with noisy links, a random subset of the links are affected by noise. The subsets selection is uniformly distributed.

TABLE 3.5: Failure rate F(S), reading rate O(S) and average number of read sensors with disconnected links

failing links	protocol	F(S)	O(S)
30%	WMBUS	90.0373 %	51.66%
	NARUN	29.9265 %	94.35%
15%	WMBUS	77.6022 %	72.93%
	NARUN	2.1864 %	99.79%
5%	WMBUS	48.4440 %	90.22%
	NARUN	0.1245 %	99,99%

3.5.1 Simulation with disconnected links

This section compares the protocols when some of the links are temporarily not working. More precisely, it considers the case where a subset of the links is disconnected. Each simulation is a sequence of experiments $S = \{E_1, E_2, \ldots, E_k\}$. Each experiment E_i is composed of a sequence of runs (i.e., $E_i = \{U_1, U_2, \ldots, U_q\}$ with $q = 50$) . At each run U_i, a random subset of the links is treated as disconnected ones (any communication via those links fails). All the remaining links always deliver messages. For each run U_i, h rounds are performed $\{R_1, \ldots, R_h\}$ ($h = 50$). When the run U_i is completed, a new one U_{i+1} is performed where the same is repeated (a subset of the edges are randomly picked and disconnected). When moving to a run U_{i+1} from the previous one (i.e., U_i), the protocol is not restarted (i.e., its state is kept). This allows us to test the ability of NARUN to update correctly all links and converge to the new network state. Simulation has been performed for 30%, 15%, and 5% of disconnected links. Each simulation has been executed for the WMBUS protocol and the NARUN protocol. Table 3.5 shows the results of our simulations. The increase in the percentage of failing links results in a higher failure rate $F(S)$ and lower reading one $O(S)$. WMBUS has always had a higher failure rate and lower sensor reading with respect to NARUN. In fact, NARUN marks failing links and always tries alternative routes (if any). NARUN improves the WMBUS reading rate by 42.67% and 26.86% when 30% and 15% of the links are not usable. NARUN improves the WMBUS reading rate by 9.77% when 5% of the links are not working. We can conclude that NARUN eavesdropping capabilities allow the update of the collector network graph with the right link information. This allows NARUN to find an alternative path and to have a higher reading rate.

3.5.2 Simulation with noisy links

This section compares protocols when some of the links are noisy. More precisely, it considers the case where a certain percentage of the links are affected by an increasing amount of noise (i.e., from -70 dBm to -80 dBm). Each simulation is a sequence of experiments $S = \{E_1, E_2, \ldots, E_k\}$. Each experiment E_i is composed of a sequence of runs (i.e., $E_i = \{U_1, U_2, \ldots, U_q\}$ with $q = 50$). At each run U_i, a random subset of the links is selected. These have a noise power value equal to Y (with $-69 < Y < -81$)

while all the rest of the links always deliver messages. For each run U_i we perform h rounds $\{R_1, \ldots, R_h\}$ ($h = 50$). When the run U_i is completed, a new one U_{i+1} is performed where the same is repeated (a subset of the edges are randomly picked which have the noise power equal to Y). When moving to a run U_{i+1} from the previous one (i.e., U_i), the protocol is not restarted (i.e., its state is kept). This allows us to test the ability of the protocols (e.g., NARUN and DSR) to learn the new network conditions. Simulations have been performed for 30%, 15%, and 5% of noisy links with noise power values between -70 dBm and -80 dBm. The following protocols have been considered: (i) WMBUS; (ii) ECC-WMBUS; (iii) NARUN; (iv) ECC-NARUN; (v) DSR.

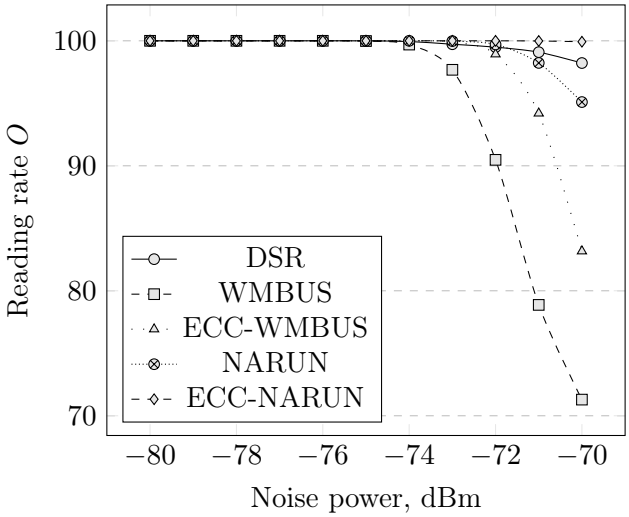

FIGURE 3.12: Reading rate O with 30% noisy links

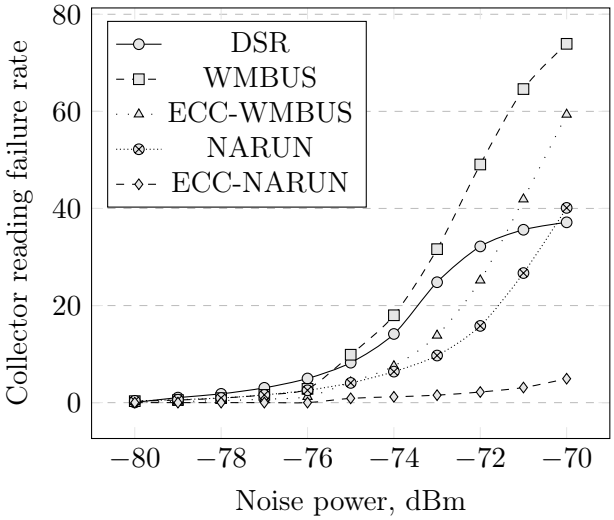

FIGURE 3.13: Collector reading failure rate with 30% of noisy links

Figure 3.12 shows the reading rate (we recall that the total number of sensors is 254) when 30% of links have noise. When the noise power is lower than -73 dBm, all protocols are able to read all the 254 sensors (i.e., 100% of sensor reading rate).

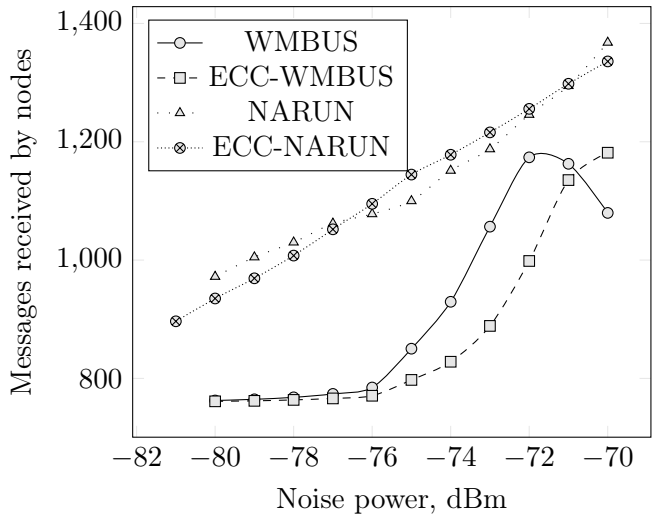

FIGURE 3.14: Average messages received by the sensors with 30% of noisy links

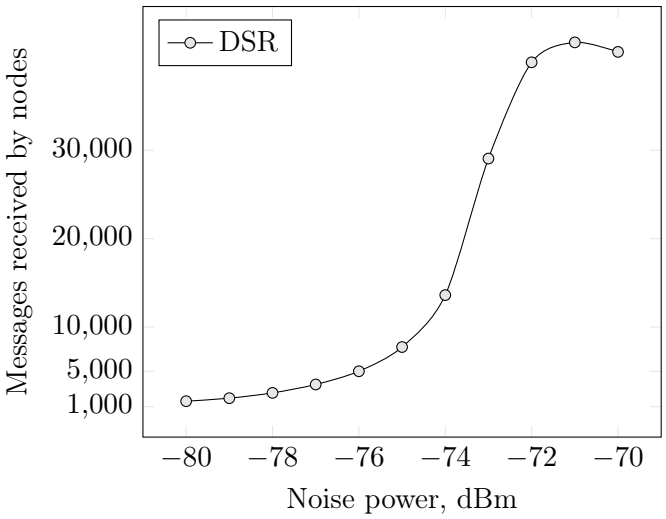

FIGURE 3.15: Average messages received by the sensors with 30% of noisy links using DSR protocol

When the noise power is -70 dBm the reading performance of the WMBUS drastically decreases to 180 sensors out of 254 (i.e., 71.29%). The use of hamming code (i.e., WMBUS-ECC) improves the reading rate of WMBUS. WMBUS-ECC reads 210 sensors out of 254 (i.e., 83.1% reading rate). NARUN reads 96% of the sensors while ECC-NARUN and DSR read 99% of the sensors. These protocols perform better since they are able to discover a path with no noisy links, unlike the WMBUS which always tries the shortest path. Figure 3.13 show the failure rate with 30% of noisy links. When the noise power is lower than -75 dBm, all protocols have a reading failure rate close to 0%. When the noise power is between -70 dBm and -74 dBm, WMBUS has a high failure rate since it always tries the shortest path without considering alternative paths with less noise. The introduction of the Hamming code (i.e., ECC-WMBUS) reduces the failure rate by 14%. The DSR and NARUN protocols reduce the failure

rate of the WMBUS by 30% and 78%, respectively. In fact, both DSR and NARUN will eventually find the least noisy path. We recall that NARUN and DSR mark a noisy link with infinity when it fails to deliver a message. The caching strategy of DSR will make paths with less noisy links become a stable choice while NARUN indirect observations can update noisy links and make them a possible choice. This explains the slightly higher failure rate of NARUN (it retries a shorter path with noisy links). When all paths that lead to a node are noisy, both NARUN and DSR can keep discovering new paths. In this case, the flooding discovery strategy of DSR will cause a higher amount of messages when compared with the indirect observations and retrial collector strategy (see Algorithm 3) of NARUN. This behavior has been validated by checking the total amount of messages received by the sensors in a round (see Figure 3.14 and Figure 3.19). DSR generates the highest traffic load in order to perform various flooding discoveries. Although WMBUS has a very low reading rate, its traffic load is not high. In fact, although it performs several attempts to read the same sensors, no flooding is involved and the shortest path is always used.

The addition of weights that are based on hamming (i.e.,ECC-NARUN) results in the lowest failure rate. ECC-NARUN reduces the failure rate by 70% and 32% when compared to WMBUS and DSR. ECC-NARUN has a low traffic load since it has the least failure rate and no overhead messages are used (i.e., link weights are updated via indirect observations).

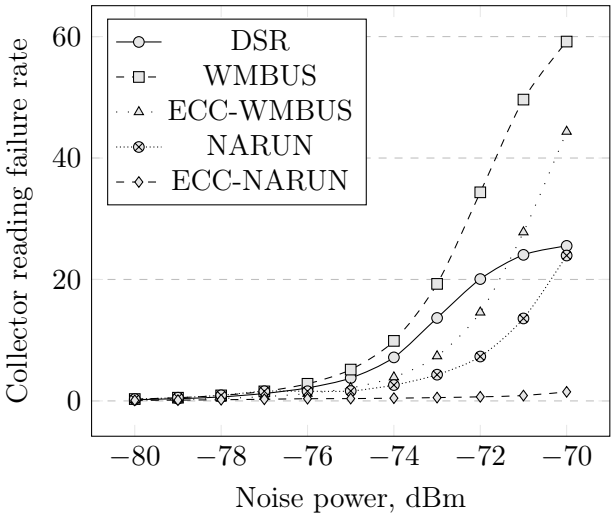

FIGURE 3.16: Collector reading failure rate w.r.t. the noise power with 15% block percentage

Figure 3.17 and 3.16 shows the reading rate and the failure when 15% of the links are noisy. Figure 3.18 and Figure 3.19 show the traffic load. The trend of the protocols is the same as the 30% noisy links case that we have already discussed.

We can conclude the following: (i) DSR and NARUN are able to read the highest number of sensors since they eventually select paths with less noise ; (ii) WMBUS reads the lowest number of sensors since always selects the shortest path without considering the link noise; (iii) ECC-NARUN has the lowest failure rate and a low

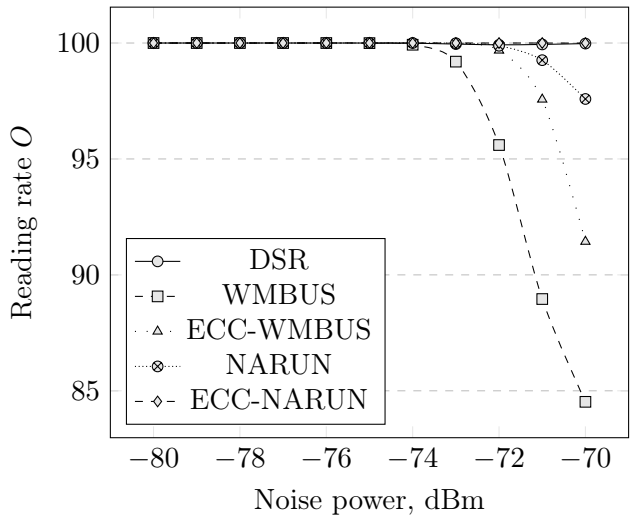

FIGURE 3.17: Reading rate O with 15% noisy links

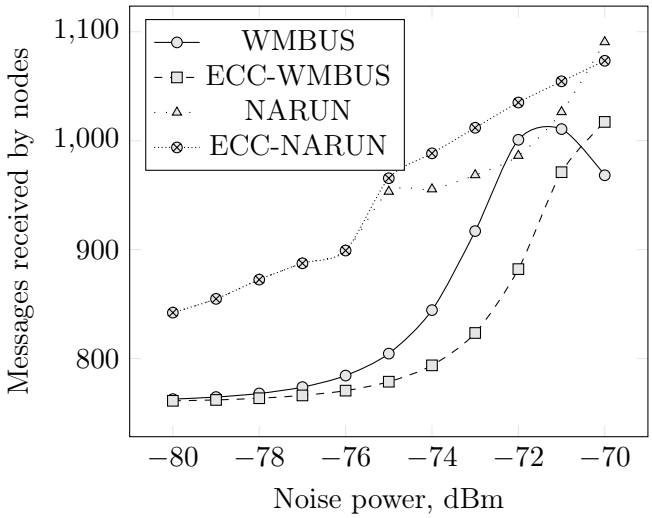

FIGURE 3.18: Messages received by nodes variation w.r.t. the noise power with 15% block percentage

traffic load. In fact, indirect link observations allow a quick update of the collector network graph without the need for overhead messages; (iv) DSR has the highest traffic load since various floodings may be needed to converge to the paths with the lowest noise. The use of the Hamming code seems effective for improving the reading rate and reducing failure.

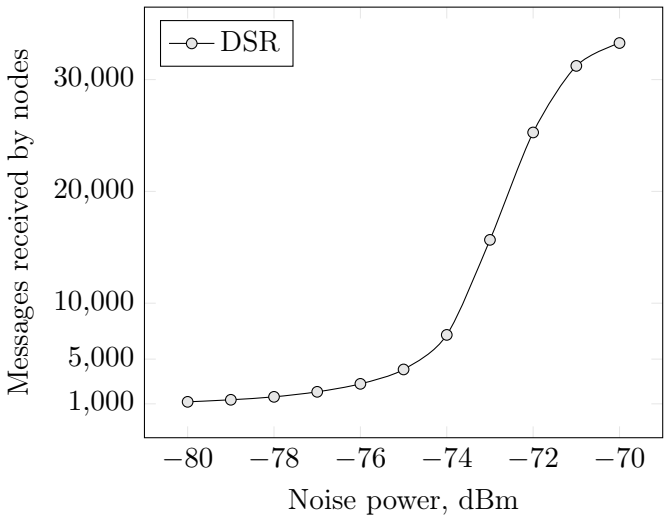

FIGURE 3.19: Average messages received by the sensors with 15% of noisy links using DSR protocol

3.6 Conclusion

The WM-Bus is a widely used meter protocol in utility networks. However, in a noisy environment, faults become a serious issue resulting in excess volumes of transmissions. The increased energy consumption due to this activity is also problematic for battery-operated nodes.

This chapter proposes NARUN which aims at reducing reading failure through the use of the Hamming ECC combined with a noise adaptive routing protocol. The employed Hamming ECC reduces re-transmissions by correcting the single-bit errors. The proposed methods are benchmarked in simulations by changing the noise power value between -80 dBm and -70 dBm. The simulations have been executed when with links that are temporary not working and noisy links. In presence of disconnected links, NARUN eavesdropping capabilities allow the update of the collector network graph with the right link information. This allows NARUN to find an alternative path and to have a higher reading rate. When link are noisy, results show that DSR and NARUN are able to read the highest number of sensors since they eventually select paths with low noise while WMBUS reads the lowest number of sensors since always select the shortest path without considering the link noise. ECC-NARUN has the lowest failure rate and a low traffic load. In fact, meters eavesdrop on the surrounding environment and efficiently report information on the link failure index back to the collector with ordinary reading messages. DSR has the highest traffic load since various floodings may be needed to converge to the paths with the lowest noise.

Chapter 4

NARUN-PC

This chapter presents a routing protocol called Noise Adaptive Routing for Utility Networks with Path Cache NARUN-PC [21]. NARUN-PC is an improvement of NARUN protocol with the usage of a path cache in the collector node. Section 4.1 provides a detailed model of NARUN-PC by comparing it to the NARUN protocol while section 4.2 presents the simulation setup and discusses related results.

4.1 NARUN and NARUN-PC protocols

In what follows, $N = \{n_0, \ldots, n_z\}$ denotes the set z nodes (meters); by M a WM-Bus message. A message is composed by three fields: the header $M.h$, the payload $M.d$, and the footer $M.o$. A network graph is denoted with $G(N, E, w)$ where $E \subseteq N \times N$ defines the communication links. $w : E \longrightarrow R$ is the edge weight function. NARUN defines a *connection-based* weight function w^r where $w^r(n_i, n_j) = \infty$ when the link (n_i, n_j) is broken while $w^r(n_i, n_j) = 1$ when the link is working. NARUN defines also a *Hamming-based* weight function w^h where $w^h(n_i, n_j)$ is calculated by using the Hamming code. In particular, the header $M.h$ and the payload $M.d$ of a frame M are divided into l equal parts $M_1, \ldots, M_l$. The sender of M calculates the Hamming code $hmc(M_i)$ of each part M_i and adds it into the footer $M.o$. The format of the message can be summarised as follow: $M = M_1||\ldots||M_l||hmc(M_1)||\ldots||hmc(M_l)$. The correction function $R(M_i)$ is defined by the equation 4.1.

$$R(M_i) = \begin{cases} 0 & M_i \text{ is received with no error} \\ \infty & M_i \text{ has a non-recoverable error} \\ 1 & M_i \text{ has a recoverable error} \end{cases} \tag{4.1}$$

The Hamming-based weight function $w^h(n_i, n_j)$ by the equation 4.2.

$$w^h(n_i, n_j) = \begin{cases} \infty & \text{IF } \sum_{i=0}^{l} R(M_i) = \infty \\ \frac{\sum_{i=0}^{l} R(M_i)}{l} + 1 & otherwise \end{cases} \tag{4.2}$$

$w^h(n_i, n_j)$ is equal to ∞ when M has a non-recoverable error, $w^h = 1$ when no error is recovered, $w^h = 2$ when each part M_i has a recoverable error. Effectively, $w^h(n_i, n_j)$ measures the link failure index lfi between n_i and n_j.

$P_{n_k}(G) = \{n_0, \ldots, n_k, n_{k+1}, \ldots, n_0\}$ (with $k > 0$) denotes a path that starts from the collector node n_0, reaches the node to be read n_k and returns to the node n_0. The path cost of P_{n_k} is defined by the equation 4.3.

$$W(P_{n_k}(G)) = \sum_{i=0}^{i=k-1} w(n_i, n_{i+1}) \tag{4.3}$$

Symmetric links are assumed; which means thus the path P_{n_k} is palindrome (i.e. the paths from n_0 to n_k and n_k to n_0 are the same). $G_{n_0}(N_0, E_0, w_0)$ denotes the collector node network graph. Each meter node n_s also defines a local meter network graph (referred to as projection graph) which is denoted by $G_{n_s}(N_s, E_s, w_s)$. N_s is the set of neighbours of n_s; E_s contains communication edges of the type (n_s, n_j) and (n_i, n_s), where the node n_s takes the role of sender and receiver, respectively. A projection graph is used by the meter n_s to store the failure indexes of its local communication links. These are sent back to the collector with ordinary reading messages and can be used to update the collector network graph. A collector keeps a timestamp t that is increased by one when a meter reading is attempted. The collector node n_0 reads all meter nodes $n_1, \ldots, n_z$ in turn (this is called **round**). A timestamp t_q denotes the q-th attempt made by the collector to read a meter. Two consecutive timestamps t_q and t_{q+1} may refer to the same meter node n_k when the routing path selected at time t_q fails to reach n_k. In this case, a different path can be tried at time t_{q+1}. In the rest of the chapter, $G_{n_k,t_q}(N_{k,t_q}, E_{k,t_q}, w_{k,t_q})$ denotes a meter graph last updated at time t_q. Each $n_j \in N_{k,t_q}$ is a neighbour of n_k added at time t_u with $t_u \leq t_q$. The link $(n_k, n_j)_{t_u}$ would also be added to E_{k,t_q} at time t_u with the related weight $w_{k,t_u}(n_k, n_j)$.

Fig. 4.1 sketches a collector round. The round starts by setting the id of the meter to be read to zero. This is stored inside the variable k. A clone of the collector network graph $G_{n_0,t}$ is put inside the variable $G1_{n_0,t}$. The collector will proceed by increasing the variable k by one so that the first meter ($k = 1$) will be read. A reading operation is described inside the pink square of Fig. 4.1 and is composed of various read meter attempts. A meter attempt is performed by using the procedure *READMETER*. The variable i stores the number of reading meter attempts the collector performed on the same meter. The i-th attempts on the meter k is denoted as a_i^k . The reading attempt a_i^k is equal to 1 when the meter n_k is successfully read, zero otherwise. Effectively, a collector reading operation can be formally defined as a finite sequence of reading attempts $r^k = a_1^k a_2^k \ldots a_i^k$ ($0 < i < max$) where max is the maximum number of consecutive attempts the collector can perform on the same meter. A reading operation r^k fails to read n_k when is composed of a sequence of max consecutive zeros. A reading operation r^k successfully reads n_k when is composed of a sequence of zeros (also an empty one) that ends with 1 (the length cannot exceed max). The system follows the MSB first approach, a sequence of attempts can not have any number after an attempt with value 1 (for instance, 1000000000 is not allowed) . Eq. 4.4 formally defines the reading operation failure rate. $F(r^i)$ measures

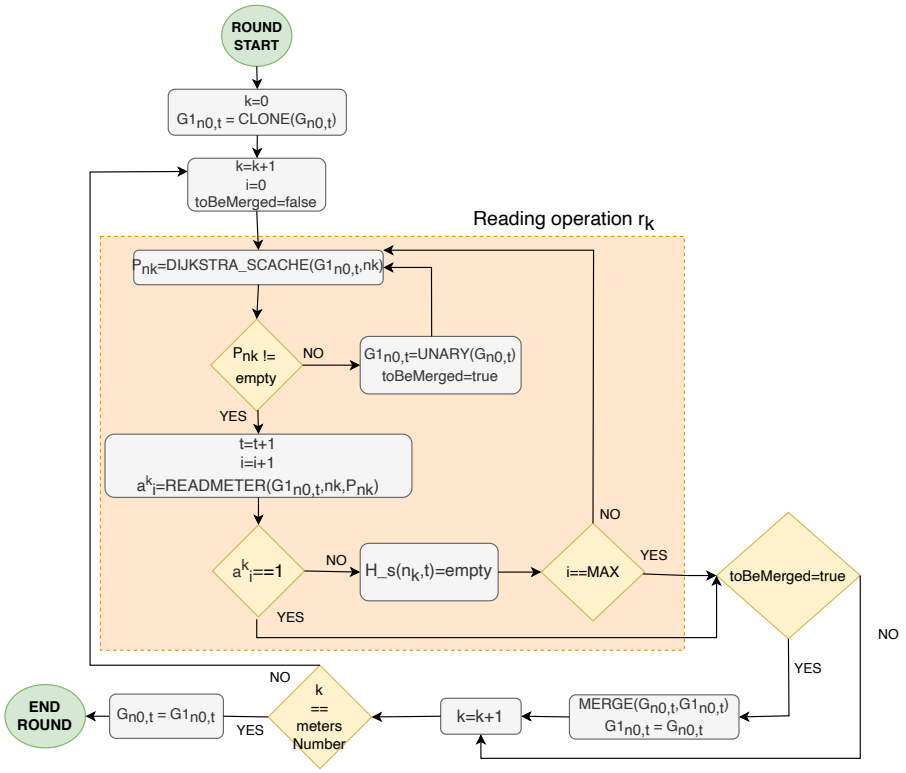

FIGURE 4.1: NARUN collector behaviour

the number of failed attempts during the reading operation. Eq. 4.5 formally defines the successful reading operation rate when reading meter i.

$$F(r^i) = \frac{\sum_{k=0}^{max}(1 - a_k^i)}{max} \tag{4.4}$$

$$O(r^i) = \frac{\sum_{k=0}^{max} a_k^i}{max} \tag{4.5}$$

A 100% reading rate does not mean a 0% failure rate. For instance, in the sequence of attempts 0000000001 there are 90% failure rate and 100% reading rate. During a reading attempt, the collector uses its network graph to find a path that leads to n_k (that is the meter to be read). When no path is found, the collector creates a unary clone of its network graph. This is a graph that has all edges equal to one. The use of the unary graph is twofold: on the one hand, it allows the collector to retry edges that have their weights set to ∞; on the other hand, ensures connectivity. When the unary graph is generated, the variable *toBeMerged* is set to true. This allows the collector to note down that the unary graph needs to be merged with the network graph $G_{n_0,t}$. In fact, the unary graph will contain fresh weights discovered by the collector that needs to be stored in the network graph $G_{n_0,t}$.

Fig. 4.2 shows a successful *READMETER* execution where a meter n_k is read by the collector n_0. This generates a *REQ* WM-Bus standard packet which contains the path P_{n_k} to reach n_k. Each node in the path receives the *REQ* message and forwards it to the next-hop by using the path P_{n_k}. The destination node receives the

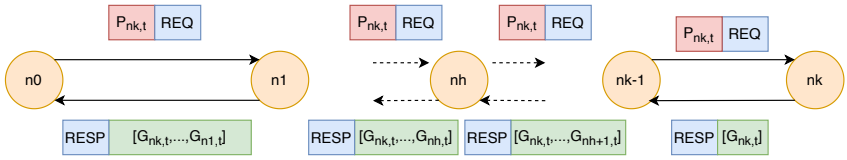

FIGURE 4.2: READMETER procedure successful execution

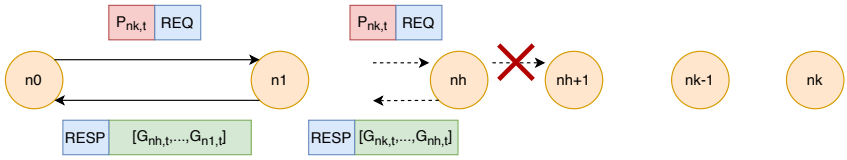

FIGURE 4.3: READMETER procedure unsuccessful execution

request and builds a response message $RESP$. This contains the reading and its local network graph $G_{n_k,t}$ with the quality of all links n_k was able to eavesdrop from the neighbour communications. The response packet is sent back to the collector node and at each hop, each meter adds its local network graph. Effectively, the collector $READMETER$ procedure receives the list of all network graphs and updates its local graph $G_{n_0,t}$ with all newer link failure indexes. Fig. 4.3 shows an unsuccessful $READMETER$ operation. In this case, a timeout will be triggered at the node that does not receive a response packet $RESP$. This node will send back to the collector node the reading failure information and its local network graph.

NARUN-PC is a simple but effective improvement of NARUN. It substitutes the $DIJKSTRA$ procedure of Fig. 4.1 with a DIJKSTRA_SCACHE($G1_{n_0,t}, n_k$) procedure (see Fig. 4.4). This includes a simple hashmap function $H_s(n_k, t)$ that stores a meter/path value. The function $H_s(n_k, t)$ can store the following values: (i) a path $P_{n_k,q}$ that was successfully used to reach the node at time q with $q < t$; (ii) a path $P_{n_k,q}$ that is return by the $DIJKSTRA$ procedure; (iii) empty when a the $DIJKSTRA$ procedure finds no path. Fig. 4.4 shows the modification to the round reading operation of Fig. 4.1 in order to add cache management. The cache path $H_s(n_k, t)$ is updated with *empty* when the procedure $READMETER$ has a result which is not ok.

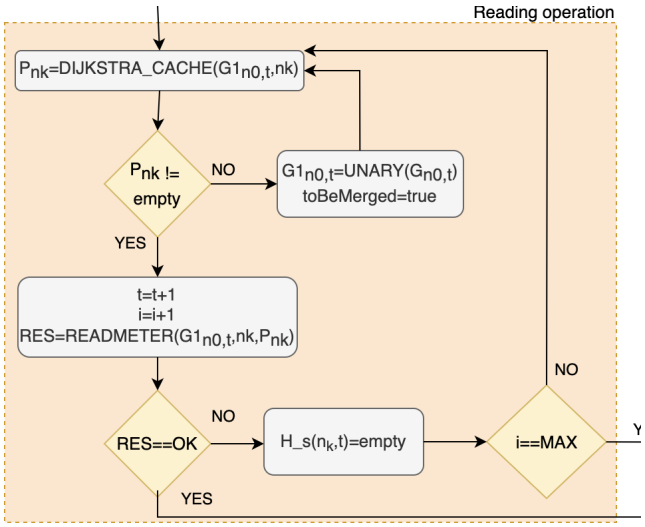

FIGURE 4.4: Cache addition into the collector round reading operation

Algorithm 10 NARUN-PC

procedure DIJKSTRA_SCACHE($G1_{n_0,t}, n_k$)
  Path $P_{n_k,q} \leftarrow H_s(n_k, t)$
  if $P_{n_k,q}! = empty$ **then**
    Return $P_{n_k,q}$
  end if
  $H_s(n_k, t) \leftarrow$ DIJKSTRA($G1_{n_0,t}, n_k$)
  Return $H_s(n_k, t)$
end procedure
procedure DIJKSTRA_ACACHE($G1_{n_0,t}, n_k$)
  Path $P_{n_k,q} \leftarrow H_s(n_k, t)$
  if $P_{n_k,q}! = empty$ **then**
    $res \leftarrow$ CHECK($G1_{n_0,t}, P_{n_k,q}$)
    if $res! = empty$ **then**
      Return $P_{n_k,q}$
    end if
  end if
  $H_s(n_k, t) \leftarrow$ DIJKSTRA($G1_{n_0,t}, n_k$)
  Return $H_s(n_k, t)$
end procedure

This simple cache strategy has been proved to be not effective since can try paths without checking their current state. A path can currently contain some broken links, thus extra overhead messages can be generated. Our solution is a slightly more sophisticated caching strategy that checks the validity of the current stored path. The algorithm 10 compares the two strategies. Procedure $DIJKSTRA\_CACHE$ shows the implementation of the NARUN strategy while procedure $DIJKSTRA\_SCACHE$ is the implementation used by the cache mechanism. Function $CHECK$ is used to verify when the stored path contains a broken link (i.e., a link with weight set to infinity). In this case, a new path is stored inside the hashmap.

4.2 Simulation

This section presents the experiments performed by comparing NARUN-PC with NARUN and DSR. The simulations have been tested in the same simulation setup described in Section 3.4.

The following protocols have been compared: (i) NARUN which uses a connection-based weight function; (ii) ECC-NARUN which uses a Hamming-based function and the Hamming code can be used to detect and recover errors; (iii) DSR [18] which is an efficient protocol suitable for wireless sensor networks [52, 53]. NARUN-PC-ECC denotes the addition of advanced caching into ECC-NARUN while NARUN-PC is the addition of caching into NARUN.

4.2.1 Experimental Results

Protocols have been compared in the case where a certain percentage of the links is affected by an increasing amount of noise (i.e., from -70 dBm to -80 dBm). At each run U_i, a random subset of the links is selected. These have a noise power equal to Y (with $-69 < Y < -81$) while all the rest of the links can always deliver messages. Simulations has been performed for 30% of noisy links.

Fig. 4.5 shows the failure rate with 30% of noisy links. When the noise power is lower than -78 dBm, all protocols have a reading failure rate close to 0%. When the noise power is between -70 dBm and -74 dBm, NARUN and DSR have always the highest failure rate. At -70dbm the failure rate of DSR becomes stable around 35% while for NARUN increases to 42%. NARUN indirect observations cause NARUN to retry paths that have noisy links but they also produce very low overhead. Results also show that at -70dBm noise, power when 30% of noisy links are considered NARUN-PC-ECC decreases the failure rate of ECC-NARUN by 2.29% and generates 31% traffic less.

Fig. 4.6 shows the reading rate (the total number of meters is 254) when 30% of links have noise. When the noise power is lower than -72 dBm, all protocols can read all the 254 meters. When the noise power is -70 dBm the reading rate of the NARUN drops by 2.5%. All protocols have a good reading performance since they can discover paths with no noisy links.

Contrary to NARUN, DSR floods the network to discover less noisy paths. After various message flooding phases, DSR eventually converges to the least noisy paths. This behaviour has been validated by checking the total number of messages received by the meters in a round (see Fig. 4.7). At -70 dBm, DSR generates 14825 messages (that is an average of 58.3 messages per reading operation) while NARUN 1387 (that is an average of 5.4 messages per reading operation). The use of Hamming and caching seem to be effective in terms of failure rate and traffic overhead. At -70dbm, ECC-NARUN has a failure rate of 5.29% with 1335 messages. NARUN-PC and NARUN-PC-ECC have a failure rate of 10.93% and 3% with 1100 and 1014 messages (see Fig.

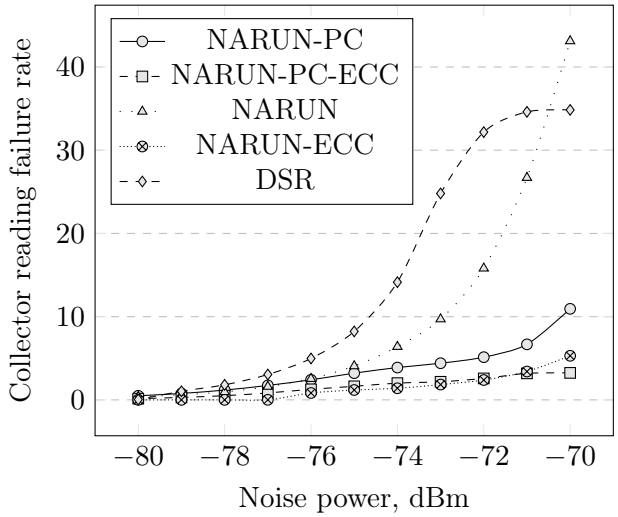

FIGURE 4.5: Collector reading failure rate with 30% of noisy

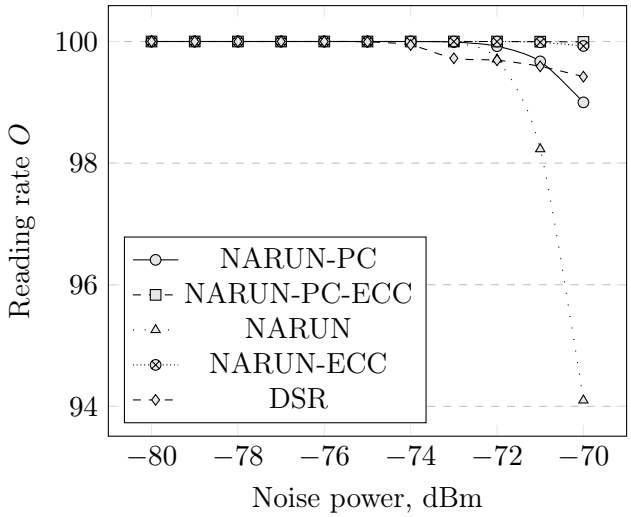

FIGURE 4.6: Reading rate O with 30% noisy links

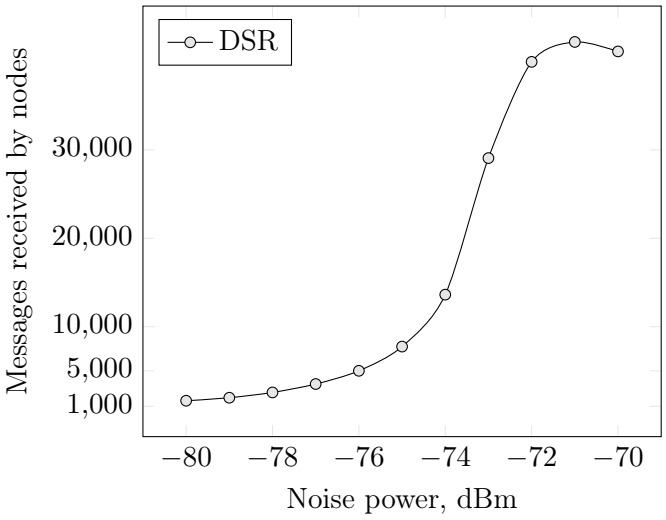

FIGURE 4.7: Average messages received by the meters with 30% of noisy links using DSR protocol

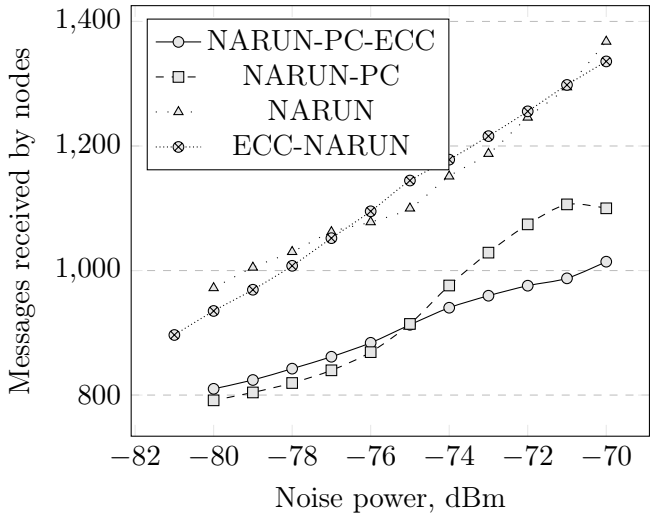

FIGURE 4.8: Average messages received by the meters with 30% of noisy links

4.8). We can conclude that NARUN-PC-ECC has the lowest failure rate and the lowest traffic. The caching strategy generates the least traffic while the use of Hamming seems to be very effective for decreasing the failure rate and traffic overhead. Similar trends can be observed for the case with the 15% of noisy links(for the sake of presentation 15% results are not presented).

Protocols have been compared when links are *unstable*. At the beginning of an experiment,a random subset of the links has been selected and therefore alternate a run where the selected links are broken and the next run where all such links are working. Fig.s 4.9 and 4.10 show the reading rate and the failure, respectively, when 30% of the links are unstable. We can see that the failure rate is greatly reduced compared to the version without noise alternation. Fig.s 4.12 and 4.11 show the traffic load. The number of messages exchanged are also lower than the figure 4.8. It is worth mentioning that The distance between NARUN-PC and ECC-NARUN at 70 dBm of noise power is 236 messages in Fig. 4.8 while the same distance is 69 messages in Fig. 4.12. Path cache simulations try stable path instead of frequently changing links. This trend confirms that NARUN-PC and NARUN-PC-ECC are suitable for networks where links are unstable. In detail, it can be seen that NARUN-PC-ECC decreases the failure rate of ECC-NARUN by 2.1% and generates 13% of traffic less. Similar trends can be observed for the case with the 15% of noisy links. NARUN and DSR have a significant reduction in terms of failure rate (9.4 % and 6.84% at 70 dBm, respectively). Path cache simulations are resilient in terms of the number of links affected by noise power. These improvements can be seen by comparing Fig. 4.13 and Fig. 4.9. NARUN improvements in terms of reading rate, can be seen by comparing Fig. 4.14 and Fig. 4.10. NARUN has 3% of improvement in terms of reading rate because it greatly depends on noisy links. Received messages are also reduced by around 200 messages at 70 dBm of noise power. This can be seen in Fig. 4.15. DSR received messages are also reduced by nearly 50% as we can see from Fig.

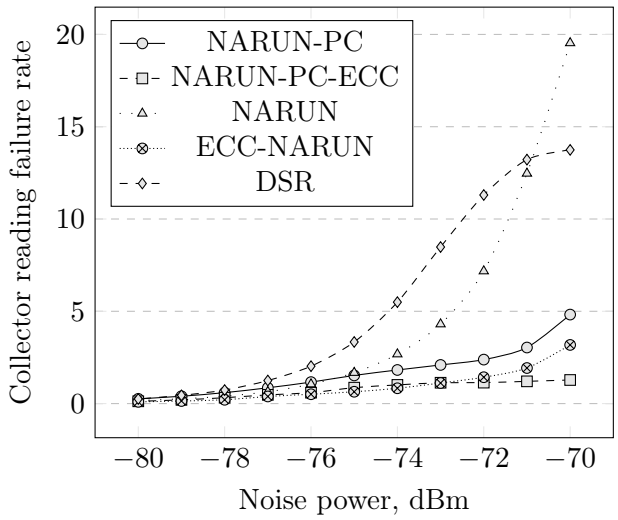

FIGURE 4.9: Collector reading failure rate with 30% of noisy links in ON/OFF simulations

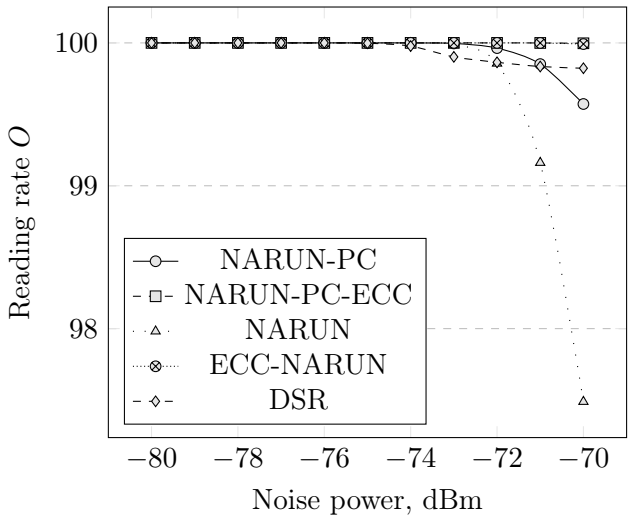

FIGURE 4.10: Reading rate O with 30% noisy links in ON/OFF simulations

4.16.

We can conclude the following: (i) NARUN-PC-ECC has the lowest failure rate, traffic load and highest reading rate in when the noise power is higher than -74 dBm; (ii) NARUN indirect observations cause NARUN to retry paths that have noisy links but they also produce very low overhead (iii) NARUN-PC and NARUN-PC-ECC are suitable for networks where links are unstable due to the lower failure rate and traffic load observed when the noise power is -70dBm.

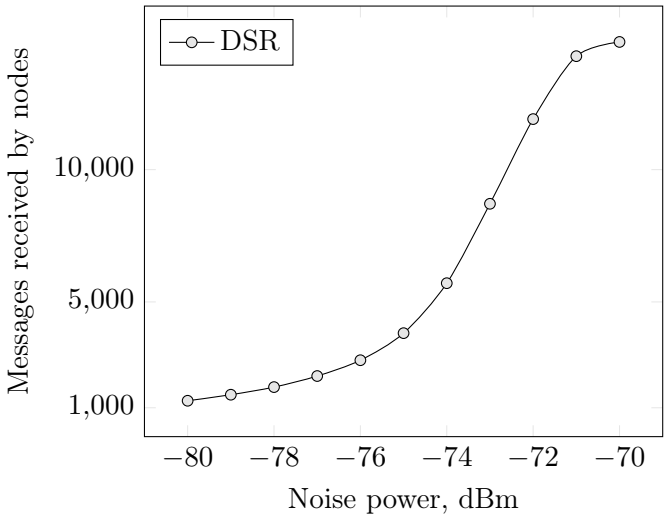

FIGURE 4.11: Average messages received by the meters with 30% of noisy links using DSR in ON/OFF simulations protocol

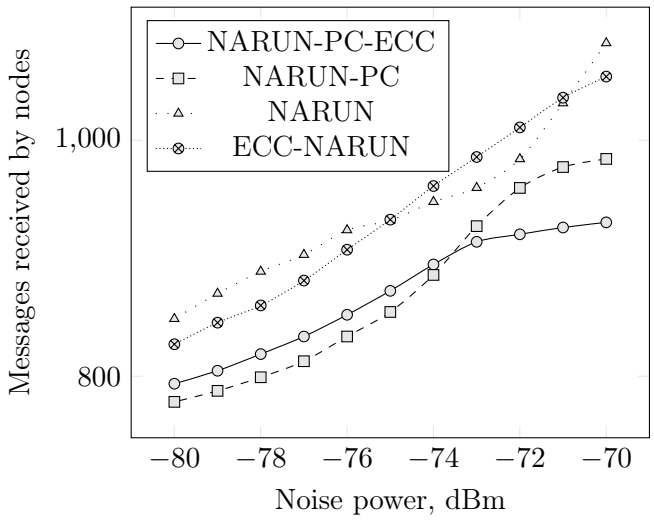

FIGURE 4.12: Average messages received by the meters with 30% of noisy links in ON/OFF simulations

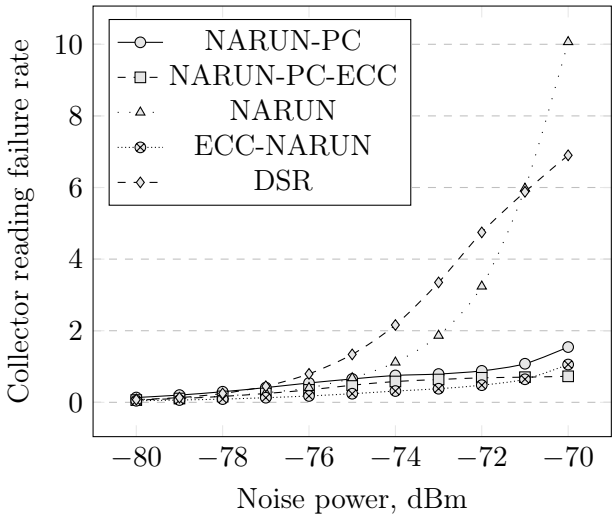

FIGURE 4.13: Collector reading failure rate with 15% of noisy links in ON/OFF simulations

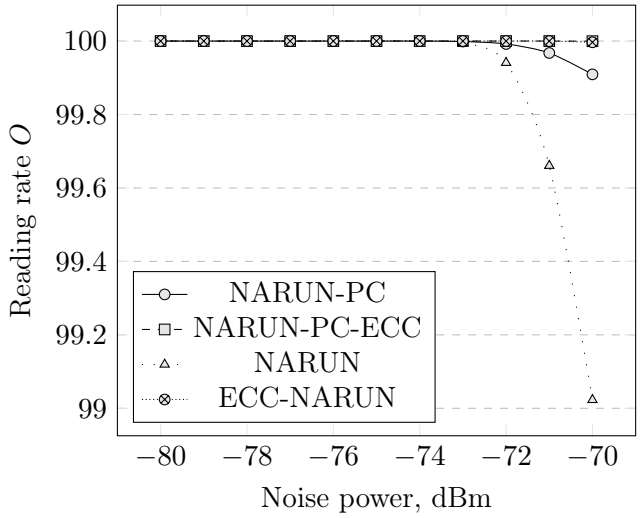

FIGURE 4.14: Reading rate O with 15% noisy links in ON/OFF simulations

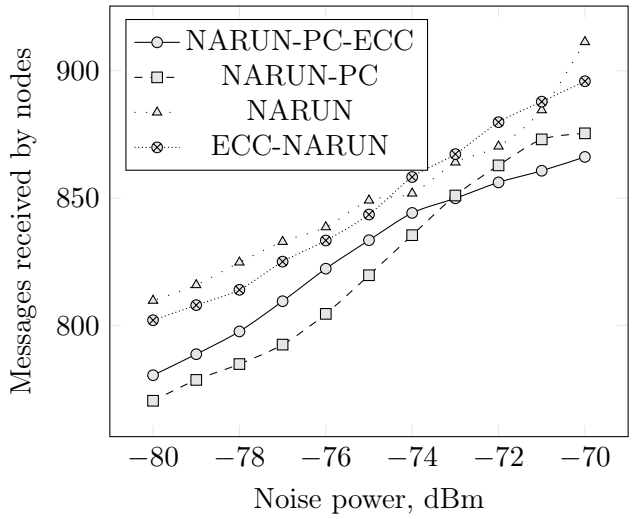

FIGURE 4.15: Average messages received by the sensors with 15% of noisy links in ON/OFF simulations

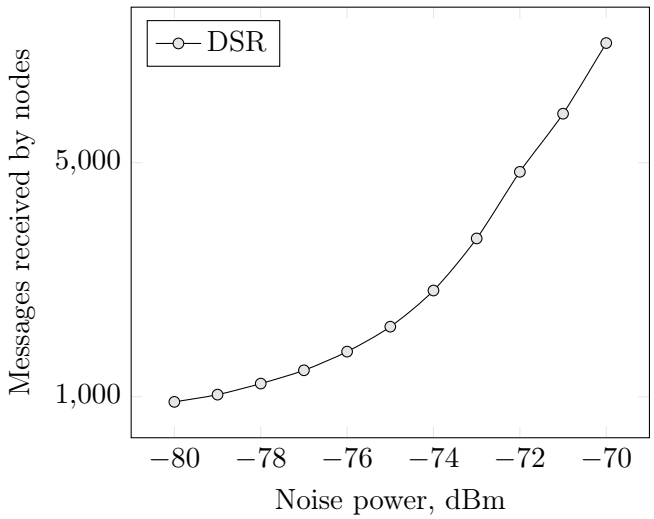

FIGURE 4.16: Average messages received by the sensors with 15% of noisy links using DSR protocol in ON/OFF simulations

4.3 Conclusion

This chapter presented NARUN-PC, an improvement of the NARUN protocol. The model has been detailed and compared against NARUN. NARUN-PC introduces a path cache strategy in the collector. The evaluation section simulates and compare NARUN, NARUN-PC and DSR protocols in a real life topology. The evaluation analyzes the failure rate, the number of readings and the message received by meters. For each simulation, experiments have been launched until the final value converges. Results show that at -70dBm noise, power when 30% of noisy links are considered NARUN-PC-ECC decreases the failure rate of ECC-NARUN by 2.29% and generates 31% traffic less. When unstable links are considered, NARUN-PC-ECC decreases the failure rate of ECC-NARUN by 2.1% and generates 13% of traffic less.

Chapter 5

Conclusion

The WM-Bus is a widely used meter protocol in utility networks. However, in a noisy environment, faults become a serious issue resulting in excess volumes of transmissions. The increased energy consumption due to this activity is also problematic for battery-operated nodes.

This thesis has been presenting NARUN and NARUN-PC routing protocols as an extension of the WM-Bus protocol.

NARUN aims at reducing reading failure through the use of the Hamming ECC combined with a noise adaptive routing protocol. The employed Hamming ECC reduces re-transmissions by correcting the single-bit errors. NARUN has been evaluated through simulations by changing the noise power value between -80 dBm and -70 dBm. The evaluation considers the following metrics: failure rate, number of sensor readings and messages received by meters. NARUN has been benchmarked against DSR, NARUN (without Hamming), WMBUS (a multi-hop version of WM-Bus that chooses the shortest path) and WMBUS-ECC (WMBUS with the usage of Hamming). Our results show that DSR and NARUN read the highest number of sensors since they eventually select paths with low noise while WMBUS reads the lowest number of sensors since always selects the shortest path without considering the link noise. ECC-NARUN has the lowest failure rate and a low traffic load. In fact, meters eavesdrop on the surrounding environment and efficiently report information on link failure index back to the collector with ordinary reading messages. DSR has the highest traffic load since various floodings may be needed to converge to paths with the lowest noise.

NARUN-PC has been proposed, an improvement of NARUN with the addition of a caching strategy. While NARUN always selects the best available path, NARUN-PC adopts a caching strategy that converges to stable paths avoiding the selection of unstable links. Likewise NARUN, NARUN-PC has been validated on a real topology. NARUN-PC-ECC decreases the failure rate of ECC-NARUN by 2.29% and generates 31% traffic less at -70 dBm noise power and when 30% of noisy links are considered. When unstable links are considered, NARUN-PC-ECC decreases the failure rate of ECC-NARUN by 2.1% and generates 13% of traffic less when the noise power is at -70 dBm. As future works, NARUN and NARUN-PC protocols can be simulated using different noise models (e.g. fading channels) and networks with different levels of network density. A test bed can be also used to verify the hardware capabilities

usage, similarly to PICO-MP. The hardware implementation can be done by using custom firmware and reasonably priced hardware, similarly to the one implemented in paper [44]. Improvements can be also introduced in protocols to increase reading reliability (e.g. add alternative paths in the *REQ* message) without adding extra service messages.

Chapter 6

Appendix

This thesis also presents PICO-MP[54], a type-based pub/sub middleware for Heterogeneous Wireless Sensor and Actuator Network (HWSAN). HWSAN are composed of devices with different and limited hardware hardware capabilities. This work has been done along with my college Andrea Piermarteri and my supervisor Prof. Leonardo Mostarda. Likewise NARUN, PICO-MP routes messages in a network where devices are constrained in terms of memory and CPU. Both protocols aim to reduce the number of messages exchanged. NARUN improves reliability by choosing a stable path while PICO-MP filters messages to monitor the network using subscriptions. This chapter provides a detailed model of PICO-MP. The section 6.1 covers the composition of publication and subscription structures, the section 6.2 provides the network structure by giving an overview of the system and the section 6.3 presents a tested implementation written in C++ to evaluate the feasibility and the advantages of this middleware.

6.1 Publication and subscription model

PICO-MP is a type-based pub/sub middleware which means that publications are coupled with information called type. Multiple nodes can provide publication using the same type. A node can provide multiple publications, as we can see in the listing 6.1. A publication structure is composed of a type and a map object (a key-value data structure), as we can see in the listing 6.1. The key is used to describe the value, whereas the value is the content provided by the publisher. For instance, A GPS measurement can be structured as follow: type "G" and a key-value object composed of latitude, longitude, and latitude. These three are the keys, whereas contents are the measured values of the object. The values can be numeric (byte, short, int, long, float), byte array, and ASCII string. We denote the type of these values as *PicoType*, as we can see in the listing

Listing 6.1: PICO-MP Publication model

```
1 Publication = {
2     type: string,
3     obj: Hashmap<key: string ,value: PicoType>;
4 }
```

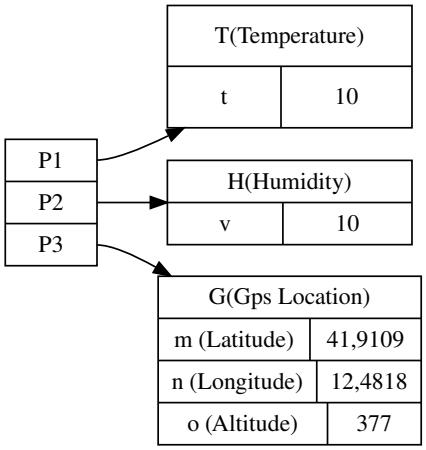

FIGURE 6.1: PICO-MP publication structure

These publications are queried using a PICO-MP subscription. Subscription are variant of the first order logic formula, as we can see in the listing 6.3. The listing 6.2 shows the data structures involved in a formula. The formula is composed of quantifiers Q^f and predicates P^f (line 1 of Listing 6.3). Each quantifier has a type of the quantifier q (existence or universal quantifier), a variable v and a type t (lines 2 and 3 of Listing 6.3). The set of predicate P^f are joined by conjunctions. Each predicate p_i in P_f where $i = 0..|P_f|$, has a name n and a set of parameters X (lines 8 and 9 of Listing 6.3). Each parameter x in X, can be a constant or a variable (previously defined in the quantifier part) (line 11 of Listing 6.3). These constant can be number, float or ASCII string whereas the variable is an instance of a publication provided by PICO-MP nodes (line 12 of Listing 6.3). For the sake of simplicity, we denote t_x as the publication type of x. If x is constant, then the t_x is $\emptyset$. We denote also $T^f_{p_i}$ as the set of t_x for each x in $p.X$ and T^f as the set of $T^f_{p_i}$ for each p in P^f. We define predicate variety $v^f_{p_i}$ as $|T^f_{p_i}|$ and the formula variety as v^f as the $\max_{\forall p_i \in P^f} v^f_{p_i}$. Checking formula with $v_f = 0$ does not require any publication since variables are not present in the formula f. The subscriber node itself can evaluate these formulae. The implementation considers a formula f with $v_f = 1$ to simplify the deployment and the distribution of these formulae.

LISTING 6.2: PICO-MP Subscription model

```
1
2  PicoVariable = { name: char;}
3
4  Formula = {
5      Q^f: Set<{q: char, v_1: char, t_1: char}>
6      P^f: Set<{n: string, X: Set<PicoType | PicoVariable> }>
7  }
```

LISTING 6.3: PICO-MP Subscription language

```
1  Prenex_formula := Prefix Predicates
```

```
2  Prefix := Prefix Q, | Q
3  Q:= Quantifier Variable : Channel
4  Channel:=[A−Z]
5  Variable:= [a−z]
6  Quantifier:= '∀' | '∃'
7  Predicates:= Predicates Predicate | Predicate
8  Predicate:= PredicateName (Parameters)
9  PredicateName:= [a−z|A−Z]+
10 Parameters:= Parameter, | Parameter
11 Parameter:= Variable | Constant
12 Constant:= Number | Float | String
```

6.2 Architecture

In PICO-MP, nodes are deployed in a hierarchical topology. Figure 6.2 shows the structure of a PICO-MP instance. Each circle represents a node, and black links are their interconnections. Each node is connected to a father one except for the R node. This has the root role, which means it is at the first level of the hierarchy. Brokers are intermediate node which mainly routes messages and performs fog verification of formulae that they receive. The root node can decide whether a formula is true or not globally. Publisher and Pubsmart nodes provide publications to their father node. The Publisher stores the last publication. The Pubsmart is an advanced publisher that can also stores formulae. The Pubsmart node can verify formulae and send publication only when is needed. The subscriber keeps a list of formulae sent to parent node.

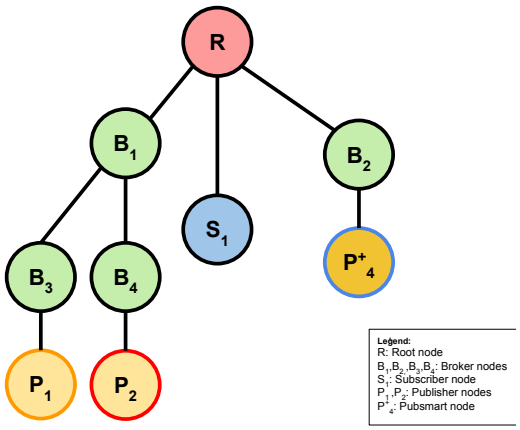

FIGURE 6.2: a PICO-MP network instance

6.3 Evaluation

The evaluation has been done using a tailored c++ implementation of PICO-MP for sensor devices. The tool is divided into application and framework code. The framework code provides an asynchronous and platform-independent application program interface (API) to implement the aforementioned behavior. The application code

mainly initializes and uses the network connection, sensors module (whether the node reads environmental data), and the framework.

The framework statically allocates data structures in memory. This allocation avoids issues such as heap fragmentation [55]. The size of these structures is fixed once the code is compiled. These values are stored in the configuration file. In the same file, constants also describe the node behavior.

The following sections evaluate the feasibility of PICO-MP in a constrained protocol and device. MQTT and PICO-MP have been simulated and compared to evaluate the energy consumption of protocols.

6.3.1 Memory evaluation

This section evaluates the feasibility of PICO-MP middleware in a constrained device such as Arduino Uno. PICO-MP can filter messages at the price of an higher memory footprint with respect to the NARUN protocol. In NARUN, a meter node only save their local graph which associates the link to the link failure rate. This size of this graph depends on the network coverage area and can be limited to avoid buffer overflow attacks. PICO-MP nodes store multiple data structures depending on the node type. This section evaluates the memory footprint of nodes by limiting the number of entries in data structures and functionalities of PICO-MP.

This device has 32 kilobytes of Flash, 2 kilobytes of static random access memory (SRAM), and 1 kilobyte of electrically erasable programmable read-only memory (EEPROM). The flash memory is used to store the program and bootstrap code. Arduino Uno uses 5 kilobytes for the bootloader code, which means that 28 kilobytes are used for instructions. The SRAM memory is used for storing global variables, virtual methods, heap, and stack space. The EEPROM is used for storing permanent information. This type of storage has not been considered since the implementation is not specific for the Arduino Uno device.

The implementation previously mentioned uses the flash memory for code instructions and the SRAM memory to store data structures. In detail, these data are preserved and stored in the global variables section. In this subsection, the run-time stack allocation in SRAM is not covered for the sake of simplicity. However, the results show that the maximum allocation of the global variables is 53% using the configuration, which means that half of the memory can be used for stack variables and virtual methods (since run-time heap allocation is not used).

Measurements were performed using two minimal configurations, namely full and basic. Table 6.1 shows the two setups which define the size of data structures used and the behavior of PICO-MP nodes. The main difference between the full and basic configuration is the supported types. The former supports all data types in a publication structure, whereas the latter allows only numeric type (without float).

The measurement was done without connectivity and with User Datagram Protocol (UDP) and Ethernet libraries. The first type of measurement evaluates the impact of framework code and data structures in Flash and SRAM memory. In contrast, the

Name	Full	Basic
Predicate count in a formula	2 predicates	2 predicates
Formula length	40 characters	40 characters
Quantifier count in a formula	2 quantifiers	2 quantifiers
Predicate name length	1 character	1 character
Predicate parameters length	5 characters	5 characters
Predicate parameters count	2 parameters	2 parameters
Number of entries in the publication object field	2 parameters	2 parameters
Number of projection in the pubsmart node	2 parameters	2 parameters
Number of publication that the a node can be publish	2 publications	2 publications
Number of publisher and pubsmart	2 nodes	2 nodes
Number of brokers children	2 nodes	2 nodes
Number of equivalent subscription	2 subscriptions	2 subscriptions
Number of subscription in the root/broker nodes	1 subscription	1 subscription
Number of projection in the root/broker nodes	1 subscription	1 subscription
Number of predicates in the pubsmart/root/broker nodes	2 predicates	2 predicates
Number of subscriber	2 subscribers	2 subscribers
Number of subscription for subscriber node	2 subscribers	2 subscribers
Number of bytes in a packet	60 bytes	60 bytes
Content based formula support	true	false
byte support	true	true
Short support (2 bytes)	true	true
Int support (4 bytes)	true	true
Long support (8 bytes)	true	true
Float support (4 bytes)	true	false
Data array support	true	false
Data array length	20	-
String support	true	false
String length	20 characters	-
Broadcast support	true	false

TABLE 6.1: Configuration full and basic. Reported values refer to the maximum availability in the node

latter is used to evaluate the feasibility in a specific scenario. The first scenario is also used to identify the best-case scenario in terms of memory. The difference between the former and the latter is the memory cost of the network connection. Such cost can be reduced by plugging a hardware shield, such as the one provided by ZigBee or Bluetooth protocol. The communication between the board and the shield occurs through a serial module that occupies less memory than any WiFi and Ethernet modules.

The tables 6.3, 6.2, 6.5, 6.4 shows the performed measurements. These results shows that publisher, pubsmart and subscriber nodes can be deployed without concerning the flash size. The root and broker nodes can be deployed depending on the configuration and loaded modules. The usage of UDP and Ethernet connectivity with a full configuration does not allow their deployment due to their code size (118% and 125% respectively), as we can see from table 6.2. The root can be deployed in other cases. The root includes most of the features provided by the broker but is not connected to anyone, which means that the root is lighter than the broker in terms of flash memory usage. The broker can not be deployed with Ethernet and UDP shield as we can see from tables 6.4 and 6.5. In the tables 6.2 and 6.3, the code size of the broker is 73% and 100% respectively, which means that the broker can be deployed, but the addition of new code (such as the one provided by the new module) requires configuration tuning.

Device	Code size	Global variable size
Pubsmart	25%	25%
Pub	19%	17%
Root	83%	30%
Broker	100%	32%
Sub	39%	18%

Table 6.2: Memory allocation at compile time using a full configuration with PICO-MP bare bone

Device	Code size	Global variable size
Pubsmart	20%	18%
Pub	15%	14%
Root	62%	22%
Broker	78%	24%
Sub	30%	14%

Table 6.3: Memory allocation at compile time using a minimal configuration with PICO-MP bare bone

Device	Code size	Global variable size
Pubsmart	68%	45%
Pub	52%	41%
Root	97%	44%
Broker	~~104%~~	45%
Sub	71%	41%

Table 6.4: Memory allocation at compile time using a minimal configuration with Ethernet and UDP protocols

Device	Code size	Global variables size
Pubsmart	78%	52%
Pub	56%	44%
Root	~~118%~~	52%
Broker	~~125%~~	53%
Sub	83%	46%

Table 6.5: Memory allocation at compile time using a full configuration with Ethernet and UDP protocols

6.3.2 MQTT vs PICO-MP benchmark

This section compares PICO-MP and MQTT-SN in terms of energy efficiency. MQTT-SN is the most popular protocol for WSAN applications.

The evaluation is modeled under a set of assumptions. The experiments were performed in a single machine. Nodes are implemented as a separate process, and communication occurs through UNIX sockets. This model avoids any hardware discrepancy and network interference between nodes. The experiments use a star topology where 15 publishers and 15 subscribers are directly connected to the root node. Each publisher and subscriber has only one sensor and actuator, respectively. We considered two cases of studies in these scenarios: distributed heating control and automatic light switch systems. These require the deployment of WSANs. Sensors and actuators are embedded in separated nodes, namely sensor and actuator nodes. Sensor nodes are also called publisher nodes for their behavior. They measure and publish data to their root node. Actuator nodes are also called subscriber nodes since they act only when a certain condition is met. These nodes are deployed in a building. More specifically, each room has only one actuator and sensor node. In the former case of the study, publishers measure the temperature in the room. In contrast, subscribers switch on radiators when the measured temperature of rooms is under a certain threshold, off otherwise. More precisely, the condition is specified by the subscription (which is requested by actuator nodes): $\forall t \in T | F(t)$ where T is the temperature topic and $F(t) = t.value > 23$. $F(t)$ is a predicate that checks temperature value $t.value$ contained in the publication t. Experiments consider publications fetched from a real-life data-set [56]. This data-set gathers a day of measurements in the Architecture Faculty in San Sebastian (Spain) during a typical spring week. For the sake of simplicity, these measurements are repeated over time during the entire simulation. The automatic presence-based light switch system is also deployed in a wireless sensor and actuator network. In this network, publisher nodes detect at least one person in their room, whereas the actuator can switch on or off the light depending on a certain condition. All lights are turned on only when at least a person is present in an arbitrary room. More precisely, the policy is equivalent to the following formula $\exists p \in P | G(p)$ where P is the presence topic and $G(p) = p.value > 0$. $G(p)$ is a predicate that checks the number of people in a room (namely $p.value$), which is contained in the publication p. Experiments consider publication that frequently changes. In particular, we assume that a person enters the room after 30 seconds, stays there for the same amount of time, and then goes out. The case of studies has been evaluated using PICO-MP and MQTT-SN protocols. In MQTT-SN, formulae are evaluated in the subscriber nodes, whereas, in PICO-MP, formulae are verified in a distributed and efficient manner. More precisely, in PICO-MP, the publishers are pubsmart nodes which means that projections are also verified locally. The same verification is also applied in the root node by considering the publication provided by children. In these experiments, the root listens and processes the incoming messages from children. In the beginning, the subscriber sends the aforementioned formula (

that depends on the case study) and waits for notification messages. Subsequently, the publisher node receives these subscriptions in the case of PICO-MP. The publisher then starts a recurrent behavior: wakes up, measures environmental data, sends the publication (whether needed), and goes to sleep. The time spent in sleep mode is denoted as sleep interval. This parameter varies this parameter between 28.8 and 192 seconds. For technical reasons, experiments were sped up to reduce their duration by 1:960. This reduction means that a day of simulation corresponds to 90 seconds of experiments. The same is applied to sensors. Their sleep interval varies between 30 and 200 milliseconds. The evaluation considers the energy consumption of publisher, pubsmart, and subscribers nodes. We assume that the root node is connected to the power supply. Each child is equipped with a battery that is fully charged at the beginning. The maximum energy stored is 2 J. This is a common settings for simulation in WSAN [57] [58] [59]. Their energy cost of a child node is made up of (i) measuring activity, (ii) processing activity, and (ii) messaging (radio) activity. Our approach aims at reducing the number of message exchanges. For this reason, we take into consideration only the messaging activity. More precisely, for each application layer message, we save the received time, the content, along with its size (measured in bits). These are aggregated per node, obtaining the total amount of bits each node sent/received during the experiment. These are necessary to calculate the energy consumption of the node. The MIT energy model is commonly used to calculate the energy consumption under certain network assumptions [17]. Communications are done by considering a free space and multi-path channel radio model. In experiments, we compute the energy consumption of each node. This consumption includes the reception and transmission of packets. The reception consumption $E_{Rx}(k)$ is the multiplication between the number of bits received k and the energy $E_{elec,Rx}$ consumed by the circuits when the node is receiving one bit, as we can see from the equation 6.3. The transmission consumption $E_{Tx}(d,k)$ is a function that takes in input the distance d between the source and destination node and the number of bits sent k as we can see from the equation 6.2. The formula uses (i) $E_{elec,Tx}$ which is the energy consumed by the circuits when the sender is transmitting one bit, and (ii) the amplifier energy $E_a(d)$ used to reach a destination at a certain distance d. The amplifier energy $E_{a,Tx}(d)$ is the multiplication between the transmit amplifier $E_{amp,Tx}$, which is 0.10 nJ/bit/m^2 [17], and the square of the distance as we can see from the equation 6.1. In experiments, the distance d (which refers to the distance between the children and the root node) is 20 m which also means that $E_{a,Tx}(d)$ =4 nJ/m^2. We also assume that the $E_{elec,Tx}$ is 50 nJ/bit and is also equal to the energy $E_{elec,Rx}$ consumed by the circuits when the node is receiving one bit [17].

$$E_{a,Tx}(d) = E_{amp,Tx} \cdot d^2 \tag{6.1}$$

$$E_{Tx}(d,k) = k \cdot (E_{elec,Tx} + E_{a,Tx}(d)) \tag{6.2}$$

$$E_{Rx}(k) = k \cdot E_{elec,Rx} \tag{6.3}$$

We used energy consumption to detect when the first node dies in the network. Experiments are executed until one node dies. The result obtained is the average duration of the simulation among 20 network runs. We refer to it as "First node Die Time". We show the trend of this value by varying the sleep interval of sensor nodes using PICO-MP and MQTT-SN protocols in the aforementioned case studies. We also highlight the consumption trend of these two protocols by using a percentage, as we can see in figures 6.7 and 6.8. In the distributed heating control system case study, the measured temperature gradually changes over time. This variation implies that the PICO-MP packets sent are less than the MQTT-SN one (since the formula truth changes rarely). More precisely, results show that PICO-MP reduces the energy consumption by some orders of magnitude compared to MQTT-SN, as we can see from figures 6.3 and 6.4. We can also see the trend difference in terms of percentage in the figure 6.7. MQTT-SN drops linearly, while PICO-MP tends to remain stable and drops faster with shorter sleep intervals. The presence frequently changes in the automatic light switch system since a person enters or leaves a room every 30 seconds. Likewise, in the first case study, PICO-MP can reduce the energy consumption of some orders of magnitude, as we can see in Figures 6.5 and 6.6. Figure 6.8 shows that both approaches tend to drop linearly. This difference is due to the pace of environmental data fluctuation. While the temperature threshold is exceeded a couple of times a day, data in the automatic light switch system varies every 30 seconds. Therefore, in the second case study, the root node notifies the subscribers more than in the distributed heating control system. PICO-MP scales better in systems where the expected period of environmental data fluctuation is dramatically longer than the sleep interval of sensors.

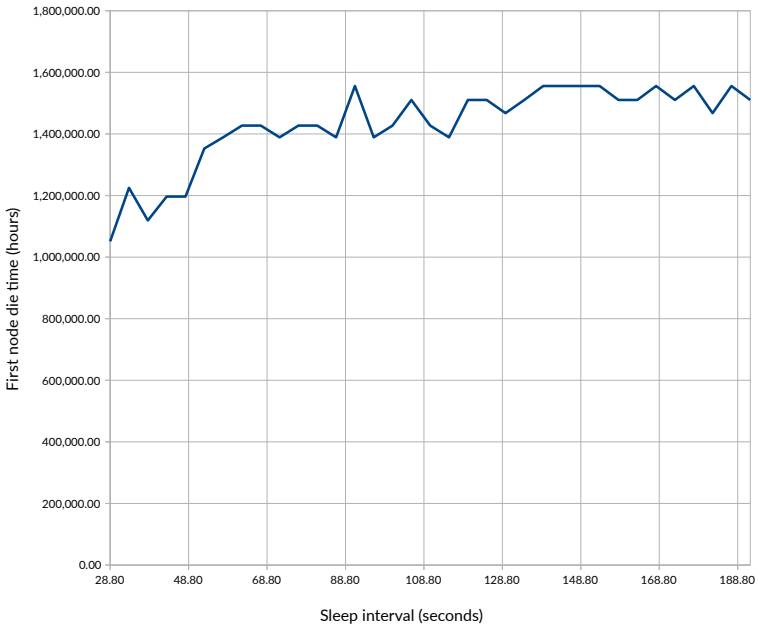

FIGURE 6.3: Distributed heating control system - PICO-MP implementation performances

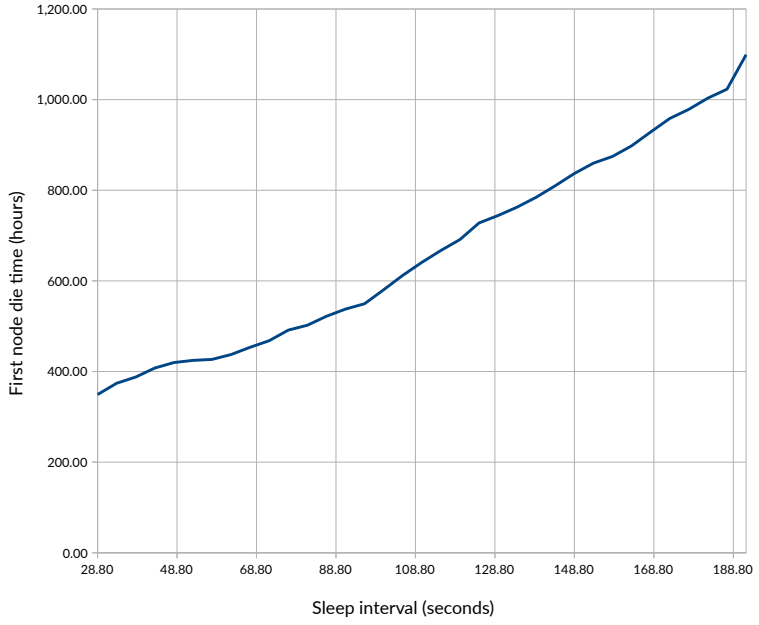

FIGURE 6.4: Distributed heating control system - MQTT-SN implementation performances

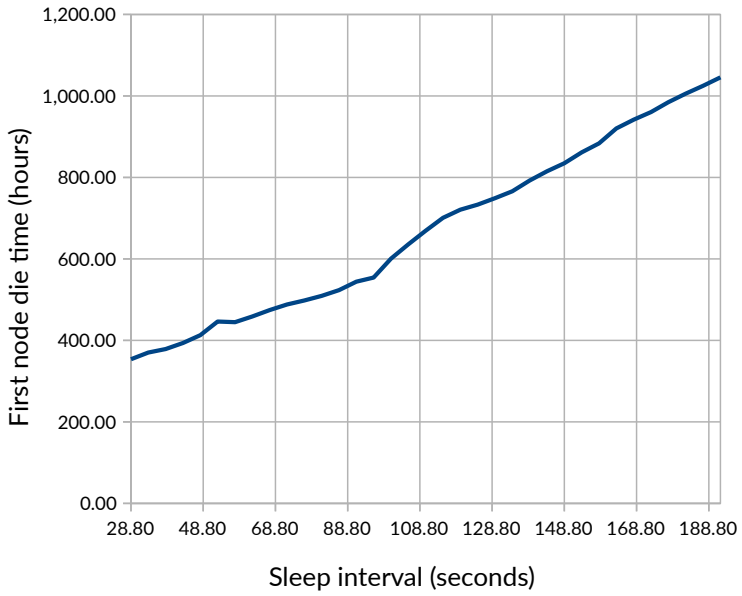

FIGURE 6.5: Automatic light switch system - MQTT-SN implementation performances

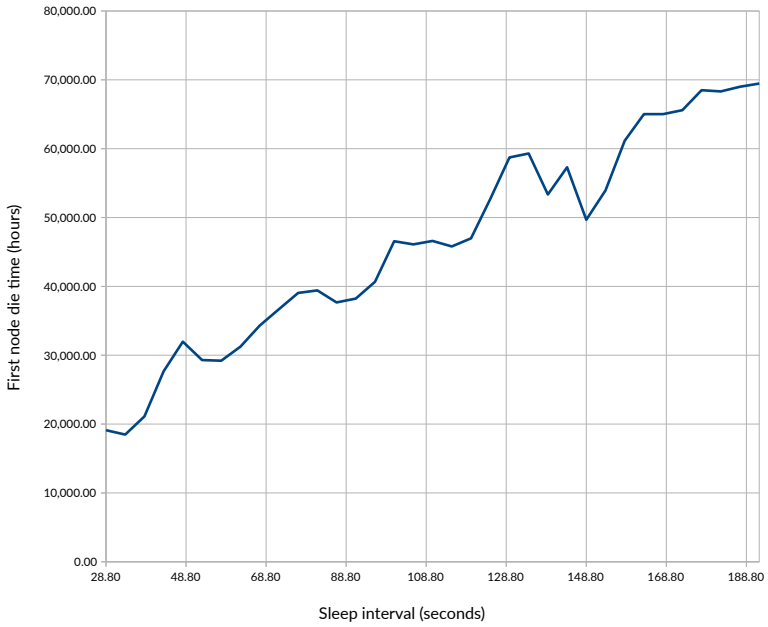

FIGURE 6.6: Automatic light switch system - PICO-MP implementation performances

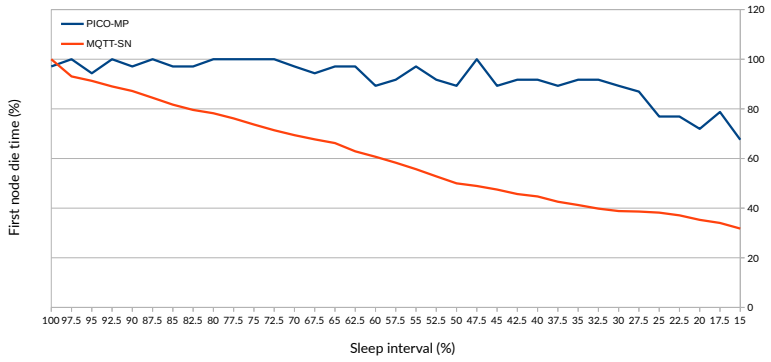

FIGURE 6.7: Distributed heating control system - Percentage comparison

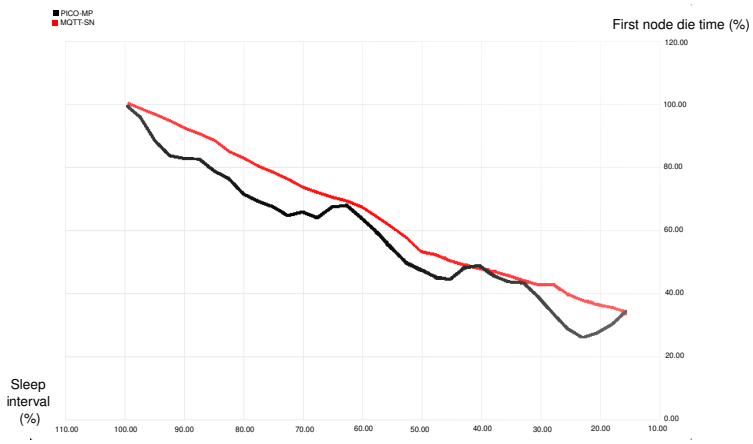

FIGURE 6.8: Automatic light switch system - Percentage comparison

Bibliography

[1] European Parliament and Council of European Union. "DIRECTIVE 2009/72/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 July 2009 concerning common rules for the internal market in electricity and repealing Directive 2003/54/EC". In: *Official Journal of the European Union* L 211 (2009), pp. 55–93.

[2] Susanna Spinsante et al. "Wireless m-bus sensor networks for smart water grids: analysis and results". In: *International Journal of Distributed Sensor Networks* 10.6 (2014), p. 579271.

[3] Nico Saputro, Kemal Akkaya, and Suleyman Uludag. "A survey of routing protocols for smart grid communications". In: *Computer Networks* 56.11 (2012), pp. 2742–2771. ISSN: 1389-1286.

[4] European Committee for Standardization. *Communication systems for and remote reading of meters, Part 2: Physical and link layer*. Standard EN 13757-2. European Committee for Standardization, 2005.

[5] European Committee for Standardization. *Communication systems for and remote reading of meters, Part 3: Dedicated application layer*. Standard EN 13757-3. European Committee for Standardization, 2013.

[6] European Committee for Standardization. *EN 13757-4 Communication systems for meters and remote reading of meters - Part 4: Wireless meter readout (Radio meter reading for operation in SRD bands)*. Standard EN 13757-4. European Committee for Standardization, 2013.

[7] European Committee for Standardization. *EN 13757-5 Communication systems for meters - Part 5: Wireless M-Bus relaying*. Standard EN 13757-5. European Committee for Standardization, 2015.

[8] European Committee for Standardization. *EN 13757-7 Communication systems for meters - Part 7: Transport and security services*. Standard EN 13757-7. European Committee for Standardization, 2018.

[9] P. Masek et al. "Communication Capabilities of Wireless M-BUS: Remote Metering Within SmartGrid Infrastructure". In: *Distributed Computer and Communication Networks*. Ed. by V. M. Vishnevskiy and D. V. Kozyrev. Springer International Publishing, 2018, 31–42.

[10] Zaib Ullah et al. "Applications of Artificial Intelligence and Machine learning in smart cities". In: *Computer Communications* 154 (2020), pp. 313–323. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.02.069. URL: https://www.sciencedirect.com/science/article/pii/S0140366419320821.

[11] Fadi Al-Turjman et al. "Network Experience Scheduling and Routing Approach for Big Data Transmission in the Internet of Things". In: *IEEE Access* 7 (2019), pp. 14501–14512. DOI: 10.1109/ACCESS.2019.2893501.

[12] Fadi Al-Turjman, B. D. Deebak, and Leonardo Mostarda. "Energy Aware Resource Allocation in Multi-Hop Multimedia Routing via the Smart Edge Device". In: *IEEE Access* 7 (2019), pp. 151203–151214. DOI: 10.1109/access.2019.2945797. URL: https://doi.org/10.1109/access.2019.2945797.

[13] R. M. Jacobsen and P. Popovski. "Data recovery using side information from the wireless M-Bus protocol". In: *IEEE Global Conference on Signal and Information Processing*. Dec. 2013, 511–514.

[14] R. M. Jacobsen and P. Popovski. *Reliable Reception of Wireless Metering Data with Protocol Coding*. 2013.

[15] Daniele Buonocore et al. "Mesh Overlay for wM-Bus Network". In: *2022 IEEE International Symposium on Measurements & Networking (M&N)*. 2022, pp. 1–6. DOI: 10.1109/MN55117.2022.9887687.

[16] F. Abate et al. "A low cost smart power meter for IoT". In: *Measurement* 136 (2019), pp. 59–66. ISSN: 0263-2241.

[17] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. "Energy-Efficient Communication Protocol for Wireless Microsensor Networks". In: *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*. HICSS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 8020–. ISBN: 0-7695-0493-0. URL: http://dl.acm.org/citation.cfm?id=820264.820485.

[18] David B Johnson, David A Maltz, Josh Broch, et al. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks". In: *Ad hoc networking* 5.1 (2001), pp. 139–172.

[19] Rosario Culmone and Fabio Pagnotta. "Energy Efficient Light Routing in Utility Network". In: *Web, Artificial Intelligence and Network Applications - Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications, AINA*. Ed. by Leonard Barolli et al. Vol. 927. Advances in Intelligent Systems and Computing. Springer, 2019, pp. 745–754.

[20] Fabio Pagnotta et al. "NARUN: noise adaptive routing for utility networks". In: *International Journal of Web and Grid Services* 18.4 (2022), pp. 384–410. DOI: 10.1504/IJWGS.2022.126118. eprint: https://www.inderscienceonline.com/doi/pdf/10.1504/IJWGS.2022.126118. URL: https://www.inderscienceonline.com/doi/abs/10.1504/IJWGS.2022.126118.

[21] Fabio Pagnotta, Leonardo Mostarda, and Alfredo Navarra. "NARUN-PC: Caching Strategy for Noise Adaptive Routing in Utility Networks". In: *Advanced Information Networking and Applications*. Ed. by Leonard Barolli, Farookh Hussain, and Tomoya Enokido. Cham: Springer International Publishing, 2022, pp. 31–42. ISBN: 978-3-030-99587-4.

[22] Rehmat Ullah, Yasir Faheem, and Byung-Seo Kim. "Energy and congestion-aware routing metric for smart grid AMI networks in smart city". In: *IEEE access* 5 (2017), pp. 13799–13810.

[23] S. Spinsante et al. "Evaluation of the Wireless M-Bus standard for future smart water grids". In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. July 2013, 1382–1387.

[24] Z. Kuder and R. M. Jacobsen. "Feasibility of Wireless M-Bus Protocol Simulation". In: *Elektrorevue* 3.3 (2012), 1–5.

[25] Z. Kuder. "Routing Protocols For Lossy Wireless Networks". MA thesis. Czech Republic: Faculty Of Electrical Engineering And Communication Department Of Radio Electronics, Brno University Of Technology, 2012.

[26] P. Masek et al. "Wireless M-BUS: An Attractive M2M Technology for 5G-Grade Home Automation". In: *Internet of Things. IoT Infrastructures*. Ed. by B. Mandler et al. Springer International Publishing, 2016, 144–156.

[27] P. Masek et al. "Communication Capabilities of Wireless M-BUS: Remote Metering Within SmartGrid Infrastructure". In: *Distributed Computer and Communication Networks*. Ed. by Vishnevskiy, V. M. and Kozyrev, D. V. Springer International Publishing, 2018, 31–42.

[28] K.-I. Hwang and S.-W. Nam. "Bitmap-wise wireless M-Bus coordination for sustainable real time energy management". In: *Sustainability* 6.7 (2014), 4326–4338.

[29] Joydeep Tripathi, Jaudelice C. de Oliveira, and J.P. Vasseur. "Proactive versus reactive routing in low power and lossy networks: Performance analysis and scalability improvements". In: *Ad Hoc Networks* 23 (2014), pp. 121–144. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2014.06.007. URL: https://www.sciencedirect.com/science/article/pii/S1570870514001243.

[30] Chiara Petrioli et al. "ALBA-R: Load-Balancing Geographic Routing Around Connectivity Holes in Wireless Sensor Networks". In: *Parallel and Distributed Systems, IEEE Transactions on* 25 (Mar. 2014), pp. 529–539. DOI: 10.1109/TPDS.2013.60.

[31] Gopalakrishnan Iyer et al. "Performance analysis of wireless mesh routing protocols for smart utility networks". In: *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 2011, pp. 114–119. DOI: 10.1109/SmartGridComm.2011.6102301.

[32] John Moy. *OSPF version 2*. Tech. rep. 1997.

[33] Thomas Clausen and Philippe Jacquet. *Optimized link state routing protocol (OLSR)*. Tech. rep. 2003.

[34] Yakubu Tsado et al. "Multiple metrics-OLSR in NAN for Advanced Metering Infrastructures". In: *2016 IEEE International Smart Cities Conference (ISC2)*. 2016, pp. 1–6. DOI: 10.1109/ISC2.2016.7580740.

[35] Government Arts College et al. "PSA-HD: Path Selection Algorithm based on Hamming Distance to Enhance the Link Stability in Mobile Ad-hoc Networks". en. In: *International Journal of Intelligent Engineering and Systems* 11.1 (Feb. 2018), pp. 259–266. ISSN: 21853118. (Visited on 10/03/2021).

[36] Zehua Wang, Cheng Li, and Yuanzhu Chen. "PSR: Proactive Source Routing in Mobile Ad Hoc Networks". In: *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. 2011, pp. 1–6. DOI: 10.1109/GLOCOM.2011.6133636.

[37] M Deva Priya and P Priyanka. "PPCLSS: probabilistic prediction coefficient link stability scheme based routing in MANETs". In: *Int J Comput Sci Eng Technol* 6.4 (2015), pp. 246–256.

[38] Harjeet Kaur, Varsha Sahni, and Manju Bala. "A survey of reactive, proactive and hybrid routing protocols in MANET: a review". In: *network* 4.3 (2013), pp. 498–500.

[39] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. *RFC3561: Ad hoc on-demand distance vector (AODV) routing*. 2003.

[40] Chenxi Liu et al. "An Improved Multi-Channel AODV Routing Protocol Based on Dijkstra Algorithm". In: *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. June 2019, pp. 547–551. DOI: 10.1109/ICIEA.2019.8833838.

[41] Sagnik Dutta, Rituparna Chaki, and Nabendu Chaki. "Optimal Reactive Routing Protocol (ORRP): A New Reactive Routing Protocol for the Shortest Path in Ad Hoc Networks". In: *2006 Annual IEEE India Conference*. 2006, pp. 1–4. DOI: 10.1109/INDCON.2006.302850.

[42] Soma Saha and Tamojay Deb. "Study and Improvement on Optimal Reactive Routing Protocol (ORRP) for Mobile Ad-Hoc Networks". In: *International Journal of Computer Science & Emerging Technologies (IJCSET)*. Vol. 1. 2010, pp. 134–138.

[43] P. Digeser et al. "Management of Routed Wireless M-Bus Networks for Sparsely Populated Large-Scale Smart-Metering Installations". In: *Recent Trends in Computer Networks and Distributed Systems Security*. Ed. by Thampi, S. M. and Zomaya, A. Y. and Strufe, T. and Alcaraz Calero, J. M. and Thomas, T. Springer Berlin Heidelberg, 2012, 385–395.

[44] A. Sikora, R. Werner, and J. O. Grafmüller. "Design and implementation of an energy aware routing extension for energy autarkic Wireless M-Bus networks". In: *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2013, 1446–1451.

[45] A. Sikora et al. "Design, Implementation, and Verification of an Energy Autarkic, RF-based Water Meter with Energy Aware Routing". In: *Energy self-sufficient Sensors; 7th GMM-Workshop*. Feb. 2014, 1–5.

[46] Tung-Linh Pham and Dong-Seong Kim. "Lossy link-aware routing algorithm for ISA100.11a wireless networks". In: *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. July 2013, pp. 624–629. DOI: 10.1109/INDIN.2013.6622956.

[47] Y.-M. Xie et al. "An EAODV routing approach based on DARED and integrated metric". In: *Wireless Networks* 20.8 (2014), 2455–2467.

[48] M. Jayakumar, N. Ramya Shanthi Rekha, and B. Bharathi. "A comparative study on RIP and OSPF protocols". In: *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. 2015, pp. 1–5. DOI: 10.1109/ICIIECS.2015.7193275.

[49] Y. Bazlov. *Coding Theory, Part2: Hamming Distance - 2010*. Lecture from The University of Manchester, School of Mathematics, Manchester. 2010.

[50] Chansook Lim. "Improving Congestion Control of TCP for Constrained IoT Networks". In: *Sensors* 20.17 (2020). ISSN: 1424-8220. DOI: 10.3390/s20174774. URL: https://www.mdpi.com/1424-8220/20/17/4774.

[51] Adam Brentnall. "Discrete-Event System Simulation (International Edition)". In: *Journal of Simulation* 1.3 (Aug. 2007), pp. 223–223. DOI: 10.1057/palgrave.jos.4250022.

[52] Diletta Cacciagrano et al. "Energy-Efficient Clustering for Wireless Sensor Devices in Internet of Things". In: *Performability in Internet of Things*. Cham: Springer International Publishing, 2019, pp. 59–80. ISBN: 978-3-319-93557-7. DOI: 10.1007/978-3-319-93557-7_5. URL: https://doi.org/10.1007/978-3-319-93557-7_5.

[53] Leonardo Mostarda and Alfredo Navarra. "Distributed Intrusion Detection Systems for Enhancing Security in Mobile Wireless Sensor Networks". In: *International Journal of Distributed Sensor Networks* 4 (Apr. 2008). DOI: 10.1080/15501320802001119.

[54] Naranker Dulay et al. "PICO-MP: de-centralised macro-programming for wireless sensor and actuator networks". In: *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE. 2018, pp. 289–296.

[55] Oliver Hahm et al. "Operating systems for low-end devices in the internet of things: a survey". In: *IEEE Internet of Things Journal* 3.5 (2015), pp. 720–734.

[56] O Irulegi, A Serra, and R Hernández. "Data on records of indoor temperature and relative humidity in a University building". In: *Data in brief* 13 (2017), pp. 248–252.

[57] Dr. Uthman Baroudi, Ahmad Shawahna, and Md. Enamul Haque. *Efficient Energy Harvesting in Wireless Sensor Networks of Smart Grid*. 2019. arXiv: 1911.07621 [cs.NI].

[58] R. U. Anitha and P. Kamalakkannan. "Enhanced cluster based routing protocol for mobile nodes in wireless sensor network". In: *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*. 2013, pp. 187–193. DOI: 10.1109/ICPRIME.2013.6496470.

[59] R. U. Anitha and P. Kamalakkannan. "Energy efficient cluster head selection algorithm in mobile wireless sensor networks". In: *2013 International Conference on Computer Communication and Informatics*. 2013, pp. 1–5. DOI: 10.1109/ICCCI.2013.6466154.