A ROUTING ALGORITHM INCREASING THE TRANSMISSION AVAILABILITY IN SMART GRIDS

Francesco Buccafurri
Lorenzo Musarella
Roberto Nardone

Department of Information Engineering, Infrastructure and Sustainable Energy (DIIES)
University Mediterranea of Reggio Calabria
Via Graziella, loc. Feo di Vito
Reggio Calabria, Italy
{bucca, lorenzo.musarella, roberto.nardone}@unirc.it

ABSTRACT

Smart grid environments require highly available transmissions to support the different Smart Services and Applications. The migration from isolated network to public communication networks created challenging issues. One of them is to guarantee a resilient communication also in presence of failures and attacks. A multi-path routing approach can support the identification of different communication paths among nodes. In this paper, we exploit this technique to increase the communication availability. The main contribution of this work is to define a routing algorithm that identifies different communication paths, possibly disjointed, in order to increase the overall transmission availability. The proposed algorithm is modeled in the Stochastic Activity Network formalism and analysed by simulations by means of the Möbius tool.

Keywords: Multi-Path Routing, Path Availability, Delivery Probability, Stochastic Activity Network.

1 INTRODUCTION

Smart Grids (SGs) are born as the evolution of classical electric grids by the employment of innovative services and technologies. Indeed, SGs implement a real-time communication network connecting the grid with electricity providers and consumers. Smart grids use this communication network in order to collect demands from consumers and to reply to them by optimizing resources available on the grid (Siano 2014, Gungor et al. 2011, Yan et al. 2013). Moreover, SGs have some internal features, such as self-healing and the ability for fault detection, which allow them to continue providing the energy flow to the grid. In Figure 1 a high-level description of a smart grid implemented in the energy area is shown. There are different smart meters and a Distribution System Operator (D.S.O.) that are linked through a distribution network. In particular, smart meters are electronic devices able to record information about the consumption of electricity and used to monitor the consumer usage and to adjust prices according to the time and the season. The D.S.O. is a physical or a legal person responsible for operating and ensuring the maintenance of the network and, in general, for maintaining distribution network resilience and security.

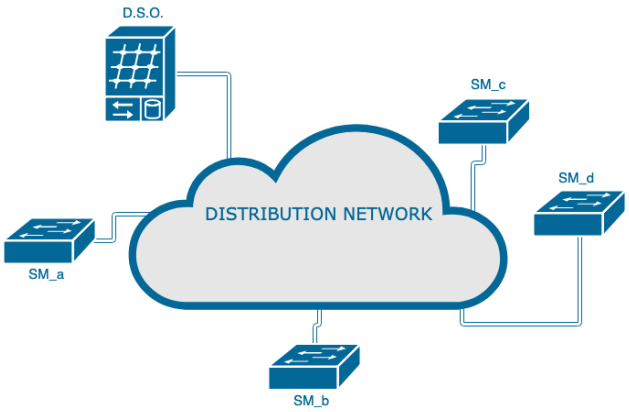

Figure 1: Example of a smart energy grid.

It is worth noting that Energy is not the only field in which smart grids are used. Another important sector in which smart grids have been implemented, is *Water* (Lee et al. 2015). Anyway, the contribution of this paper is independent of the particular sector on which smart grids are implemented. Therefore, from now, we will refer to generic SGs.

In this scenario, it appears fundamental to have a right governance of energy usage and cost reduction, as well as to ensure that messages exchanged in the communication network have, at least, a certain value of *delivery probability* guaranteed and that these communications must be secure and reliable (Hayden 2010). With the term *delivery probability*, we mean the probability that, given a message sent from a source node *src* to a destination node *dest*, it reaches correctly the destination. A path connecting *src* and *dest* is known as *routing path*. Usually, routing algorithms choose the path between a source and a destination node by minimizing the overall cost. In the case of smart grids, the main objective is to guarantee the correct delivery of messages. Consequently, existing classic algorithms does not fit completely with this scenario. For this reason, we decided to develop a new routing algorithm which saves, in each routing table, more information related to different objective functions, so that, for each application running over the grid, the appropriate function can be chosen. To achieve this goal, multi-path routing could be seen as the right solution. Indeed, the main idea on which multi-path routing is based is very simple: we transmit a replica of a single message from *src* to *dest* using multiple paths, so that the probability of reaching *dest* is increased. This approach increases the fault tolerance of SGs in the sense that, if a generic node fails, we can rely on other paths able to deliver the data to the requested destination. Clearly, re-transmission reaches its optimal result when paths are completely disjoint each other. Anyway, although it is not always possible to find or demonstrate the independence of paths, it is always provable that multi-path routing increases the delivery probability of messages in the communication network. A way to obtain the above attribute of resilience in a *justified* fashion, according to the general principles of dependability, is of course *simulation*. In this paper, besides the proposal of the routing algorithm, an important contribution is its modeling in the Stochastic Activity Network formalism and its simulation-based analysis done with the Möbius tool.

The paper is structured as follow. In Section 2 we discuss the related work. In Section 3 we first give some preliminaries concepts and some motivation and, then, we describe how our routing algorithm works, from the creation of routing tables to the recognizing and reacting to overlapped paths. In Section 4 we model our approach by using the Stochastic Activity Network (SAN) formalism. In Section 5 we report first results of simulations. Finally, in Section 6, we draw our conclusions and describe future plan.

2 RELATED WORK

Recently, smart grid is one of the topics on which the research interest has observed an outstanding increase. The authors of (Kabalci 2016) propose a complete survey about smart grids. In particular, after a clear explanation about the infrastructure, they focused their attention on smart metering and communication technologies and methods and on what is, in the smart grid scenario, security. Indeed, from this survey, it emerged that the most important area on which smart grids researchers should carry on is the one that let reliability, privacy and data protection increase. The paper (Fang et al. 2012) is one of the most appreciated surveys about smart grids in literature since authors looks at SGs from a technical point of view. In particular, they focused their studies on three different areas, such as *(i) smart infrastructure system, (ii) smart management system* and *(iii) smart protection system*. Concerning the first element of the list, it includes the information, energy and communication infrastructure on which SGs are based. Instead, the smart management system includes the tools and methods sets necessary for the advanced management of functionality of SGs that are developed on the smart infrastructure system. Finally, the smart protection system provides advanced methods for data and failure protection, reliability analysis and for facing cybersecurity issues.

In (Egert et al. 2018), authors proposes a work about different quality indicators in smart grids. In particular, their work aims at catching information from smart grids when multiple aspects, like energy level and network structure are jointly assessed. For this reason, the authors proposes an *Indicator Model* for combining a priori information about grids to estimate the service quality of the network in such a way different network configurations can be comparable. The authors of (Yigit et al. 2014) propose an approach cloud based for smart grid applications since some features required by SGs, such as flexibility, scalability and omnipresent access are guaranteed by cloud computing. Anyway, a solution based on cloud computing for SG applications needs still some additional studies and it has open research issues in terms of, among others, security, reliability and robustness. Also in some past work we focused on smart grids, discussing the trade offs among energy, performance and availability (Gentile et al. 2016) (D'Arienzo et al. 2013).

Focusing on the routing problem in smart grids, the work in (Saputro et al. 2012) represents a survey of smart grids routing protocols, highlighting that SGs open new challenges to be addressed in order to satisfy the requirements of their applications. In particular, they classifying routing protocols according to three different kind of network: Home Area Networks (HANs), neighbourhood Area Networks (NANs), and Wide Area Networks (WANs). The paper (Hong et al. 2014) proposes two multi-path routing algorithms for fault tolerant communications in SGs, both of them based on a qualitative and heuristic approaches. The two algorithms are used in order to compute different and disjointed paths between the source node and the target one and they are trade-off between running time and quality of outputs. Furthermore, they demonstrate that classic node-disjoint routing path computation algorithms do not provide a level of reliability that is enough for smart grids since, differently from more classical network, failures in SGs can be co-related. Instead, the authors of (Vaidya et al. 2012) propose a mechanism for secure multi-path routing protocol in smart grid networks and they validated the robustness of this algorithm simulating some attacks. Indeed, they suggest that security is a crucial factor for the success of SGs and that it is fundamental for avoid, or reply, to attacks such as man-in-the-middle attacks, sybil attacks and modification attacks. The resulting algorithm is a multi-path routing protocol based on a combination between AODV multiple Alternative paths and AODV version2 by providing end-to-end security and a secure route discovery. In (Chakraborty et al. 2017), the authors proposed a Pseudo-Opportunistic, Multipath Secured Routing Protocol *POMSec* for communications in SGs. In particular, they combined the advantages of both opportunistic and multi-path methods for increase the general "power" of the algorithm in a smart grid scenario. It is proposed also a trust model for let the whole algorithm be more secure by evaluating the trust level for each node of the network. This level is, in turn, computed using two different parameters, *risk* and *reputation*. The former is resulted from the past behaviour of the node, while the latter assesses a node's long term behaviour. Furthermore, if the trust level crosses a threshold value for a certain node, this node is considered as an attacker. Finally, the authors used the trust level of neighbouring nodes as a metric in order to determine the degree of routes. With

respect to the existing literature, the work described in this paper adopts the multipath routing to increase the probability of correctly delivery messages inside a network. The proposed routing algorithm is designed to work at both network layer or at application layer, in order to be compliant with existing infrastructures.

3 GENERAL DESCRIPTION OF THE ALGORITHM

This section describes the routing algorithm we propose, which exploits multi-path routing to increase the delivery probability of messages in a communication network. We exploit the possibility of using multiple paths (possibly disjointed) between the source and the destination nodes, transmitting n-times the same message over different paths. After mathematical preliminaries, which motivate our approach, we detail through pseudo-code the algorithms to be executed by each node of the network to populate its routing table. Then, we describe how the nodes recognise and react to the overlapping of paths during services, properly updating their routing tables.

3.1 Preliminaries and Motivation

Modern nodes in a smart communication network are complex components which can be easily equipped with sensors and means to monitor their own health-status, relying on different parameters such as remaining energy, failed components, internal congestion, etc. It is also feasible for each node to support the run-time monitoring of the health status of its neighbours, communicating them detected anomalies. Starting from this consideration, in our proposal we associate an availability level to each node inside the network. We denote with $A_i \in [0..1]$ the availability of the i-th node n_i. In our meaning, the node availability represents the probability that a node properly forwards a received message to the subsequent in the communication path. Hence, the node availability is equal to 0 if the node is failed, while it is 1 when the node is ensuring a proper message delivery. It is out of the scope of this paper the way to evaluate this value; we assume that each node can evaluate its availability level (possibly supported by neighbours) and can store it locally.

Starting from the node availability definition, the *path availability* between a source node s and a destination d can be calculated as the product of the availabilities of the nodes $n_1, ...n_k$ composing the path (Figure 2). In formula, $PA_{n_1,...,n_k} = \prod_{i=1}^{k} A_i$. According to this definition, the path availability represents the probability that a message sent by s is correctly delivered to d, by the path $n_1, ...n_k$. Note that the two nodes s and d are not taken into account in the path availability and excluded from the product since their correct operation is essential to the communication. Furthermore, with the exclusion of nodes with an availability equal to 1, the path availability hugely decreases with the increase in the number of hops in a path (multiplicative effects of values lower than 1). For sake of simplicity, we do not consider availabilities of communication links, but we suppose that it is taken into account in the node availability values.

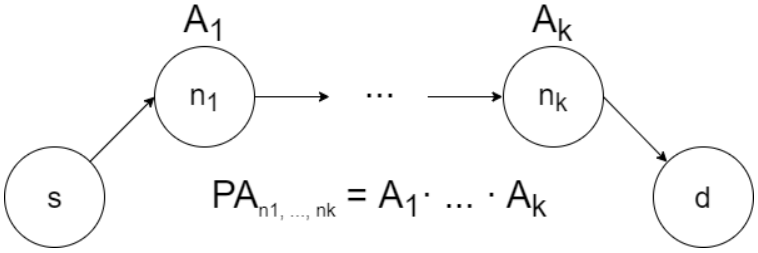

Figure 2: Evaluation of the Path Availability.

Let suppose that a source node s is connected by two totally disjointed paths $n_1,...,n_k$ and $m_1,...,m_l$ to a destination node d (Figure 3). Consequently, s could send a message to d through the first path and it can be sure that it will be correctly be delivered with a probability of $PA_{n_1,...,n_k}$ (path availability of the first path). Analogously, it can send a message through the second path and it can be sure that it will correctly delivered with a probability of $PA_{m_1,...,m_l}$ (path availability of the second path). However, s could also send a couple of messages through both the paths, and it can evaluate the probability of correct delivery as $1-(1-PA_{n_1,...,n_k})\cdot(1-PA_{m_1,...,m_l})$. In fact, this value is given by 1 minus the probability of a missed delivery on both the paths. This probability is considerably higher than the one reachable using a single path, at the disadvantage of a redundant message sent. Figure 3 depicts also the routing tables needed by the node s to be aware of the two paths. As it will be detailed in the following, for each possible destination node it has to store a table in which the three columns are: path availability, next hop and number of hops (optional). In the depicted table, we reported exemplifying values that are 0.7 for the path $n_1,...,n_k$ and 0.65 for the path $m_1,...,m_l$. Furthermore, the former path allows to reach the destination in 10 hops while the latter in 8 hops. Consequently, the delivery probability using both the paths, evaluated according to the previous formula, reaches 0.895. Moreover, note that if s and d are connected by a third disjointed path with a path availability of 0.4 (hence not very available), the overall delivery probability further increases to 0.937. Generalizing, given two nodes that are connected by p disjointed paths, each one with a path availability PA_i, we obtain that the delivery probability of a message from a node s to a node d as : $DP^p_{s,d} = 1-\prod_{i=1}^{p}(1-PA_i)$.

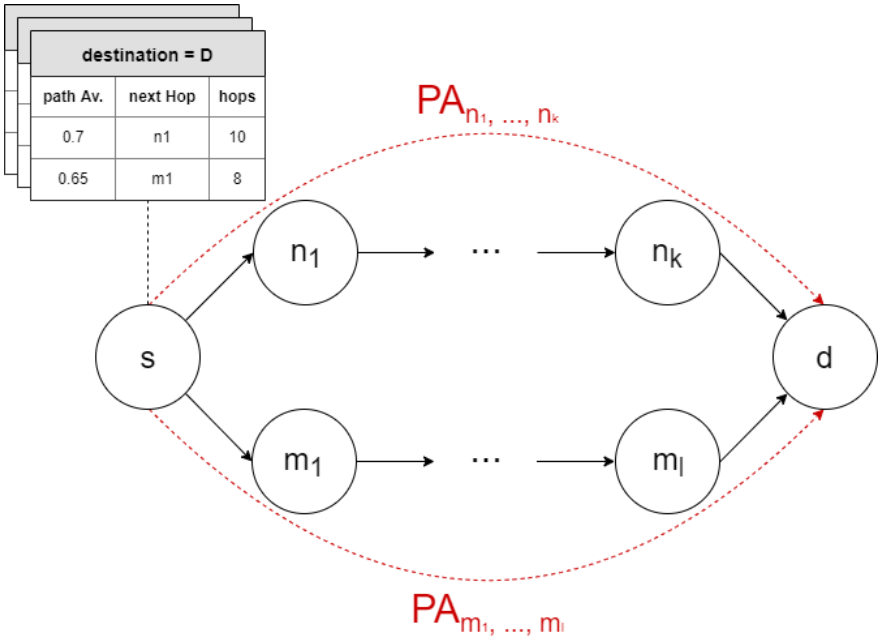

Figure 3: Evaluation of the Delivery Probability.

As it is possible to deduce from the previous example, the success probability of a message delivery could significantly increase if different disjointed paths are available between two nodes and the message is sent over such paths. Moreover, the number of needed paths needed to significantly increase the overall delivery probability is very low. Hence, the main challenge in our routing algorithm is to find disjoint paths between a couple of nodes and create the necessary routing tables, limiting the necessary information to store at the node level and the required messages to be exchanged among nodes.

As said before, our approach requires to store at node-level a set of tables, one for each possible destination (in the worst case for each node of the network), where each row represents a possible path. For each path

(row in the routing table) the needed information are path availability, next hop and, optionally, number of hops (generalizing, total cost). For each node and for each table, the maximum number of rows it has to store is given by its out degree, which is certainly higher than the number of possible independent paths towards a whatever destination. In fact, if a node has a certain number x of outgoing arcs, we can immediately deduce that no more than x independent paths can connect this node to a whatever destination. With these routing tables, if a node s needs to send a packet to d and it wants to obtain a maximum delivery probability DP', it accesses to the table related to d and it selects one or more paths from the table, according to defined decision algorithm. Examples of decision algorithms could be: (1) starting from the path with minimum cost, the node selects further paths with increasing costs up to reach a $DP_{s,d} > DP'$, or (2) starting from the path with the highest path availability, it could select further paths with decreasing path availability up to reach a $DP_{s,d} > DP'$.

Given the complexity of finding independent paths, our routing algorithm aims at individuating possible independent paths without ensuring the total independence among them. We also defined a strategy to find possible overlaps at service time and report this information to source nodes, that could react by updating the routing tables accordingly. In the following subsections, we describe both these activities.

3.2 Creating Routing Tables

As previously stated, our routing algorithm aims at constructing for each node the set of routing tables for each possible destination. For sake of simplicity, we do not allow for network clustering, but of course, this technique could be exploited to reduce the set of possible destinations, using some nodes as "gateways". We remind that for each routing table (i.e., for each destination), each row represents a possible path and contains information about the path availability, next hop and number of hops (total cost if edges can have different weights).

To create these tables, each node entering the network has to communicate information about a neighbour (recognised as a final destination node) to all other neighbours. From now on, we consider all the nodes in the network as possible destination nodes. This would represent the worst case since a limited number of possible destinations (i.e., base stations) limit the total number of exchange messages and the number of stored tables.

When a node n discovers the node i as its neighbour, n updates the table related to i, storing (in a row) that the path availability towards i is 1.0, the next hop is i and the total cost is $c_{n,i}$. Then, the node n sends this information to all other neighbours (different from i), communicating that they could reach i through n with a path availability of A_n (i.e., the node availability of n) and a total cost of $c_{n,i} + c_{j,n}$. Hence, it has to send a routing message that contains the following fields: *source* that is the sender node (n in the example), $finalDestination$ that is the final destination node (i in the example), $pathAvailability$ that is the total path availability of the path towards the final destination (A_n in the example), *cost* that is the total cost of the path towards the final destination ($c_{n,i} + c_{j,n}$ in the example). The algorithm performing this step is reported as pseudo code in Figure 4.

In the code, we consider the function $neighbour(n)$ that returns the list of neighbours of the node n while $routingTable$ is a table with a number of rows equal to the number of nodes and a number of columns equal to the out-degree of that node. Each cell of $routingTable$ contains a data structure with the 3 fields needed by the routing algorithm. To send a message to a node j, we use a shared matrix *messageList*, and we call the function *add* to add a message at row j.

When a node receives a routing message from a neighbour, it execute the algorithm reported in Figure 5.

```
1  for(node i : neighbours(n)) {
2      routingTable[i][0].pathAvailability = 1.0;
3      routingTable[i][0].nextHop = i;
4      routingTable[i][0].cost = c_{n,i};
5      //send info to neighbours
6      for(node j : neighbours(n)) {
7          if(i!=j) {
8              node source = n;
9              node finalDestination = i;
10             double pathAvailability = A_n;
11             int cost = c_{n,i} + c_{j,n};
12             Message M(source, finalDestination, pathAvailability, cost);
13             messageList[j].add(M);
14         }
15     }
16 }
```

Figure 4: Pseudo-code of the neighbour discovering step.

In details, the node n receives the set of messages by calling the function *getAll* on the row n of *messageList*. For each received message it stores the fields in variables *source*, fD (final destination), *pathAvailability* and *cost*. Then it checks if a path towards fD with *source* as $nextHop$ is already known. If this happens and the new *pathAvailability* is higher than the stored one, the node overwrites the old information with those last received; otherwise, it discards the received message. In the former case, the node communicates also its neighbours the received information, except for *source* and fD (if it is among the neighbours of n). On the contrary, if a path towards fD with *source* as $nextHop$ is not known, the node looks for the possibility to insert this value in its routing table. Hence it retrieves the minimum path availability and its column index by calling the two functions *getMinPathAvailability* and *getMinPathAvailabilityIndex*. Note that these functions return the value 0 and the related index if a cell is empty. If the minimum of the path availability is lower than the received *pathAvailability* (or if it is equal but with a higher cost), the node stores this path to the table related to fD and communicates this information to neighbours, as in the previous case.

When the described algorithm reaches the stability, each node has a routing table that, given a destination, gives information about the path availability and the total cost that could be reached by selecting a specific neighbour as a next hop. Note that, it is never possible that two paths towards the same destination share the next hop. Otherwise, it is possible that two paths are not completely disjointed, so crossing at least one common node.

3.3 Recognising and Reacting to Overlapped Paths

The objective of the routing algorithm previously described is to find, given a destination, the most available paths in order to maximize the delivery probability. Furthermore, these tables can be also used to realize the forwarding function of other nodes. However, the algorithm does not ensure that choosing two or more different paths from the table related to a specific node, these paths are completely disjointed and the total delivery probability is the one calculated with formula in Section 3.1. For this reason, without complicating the routing algorithm both in terms of number of routing messages and structure of the messages themselves, we designed a mechanism that recognises overlaps during operational time.

To implement this mechanism, we rely on a message identifier, that is an integer value that uniquely identifies a message sent by a certain source node. If a message is sent over more than one path, the sender uses the same message identifier on all the message before increasing it. Each node stores locally the last received message identifier for each source node. When receiving a message, each node forwarding it updates the local last received message identifier. If a node s sends a couple of replicated messages to d with a certain

```
1  for(Message M : messageList[n].getAll()) {
2      node source = M.getSource();
3      node fD = M.getFinalDestination();
4      double pathAvailability = M.getPathAvailability();
5      int cost = M.getCost();
6      bool alreadyKnown = false;
7      int i = 0;
8      while (i < outDegree(n) && !alreadyKnown) {
9          if(routingTable[fD][i].nextHop == source) {
10             alreadyKnown = true;
11             if(alreadyKnown && pathAvailability>routingTable[destination][i].pathAvailability) {
12                 routingTable[fD][i].pathAvailability = pathAvailability;
13                 routingTable[fD][i].cost = cost;
14                 //send info to neighbours
15                 for(node j : neighbours(n)) {
16                     if(j != source && j != fD) {
17                         Message M(n, fD, pathAvailability·A_n, cost+c_{j,n});
18                         messageList[j].add(M);
19                     }
20                 }
21             }
22         } else    i++;
23     }
24     if (!alreadyKnown) {
25         //find min path availability in the routing table
26         double minPathAv = routingTable[fD].getMinPathAvailability();
27         int indMin = routingTable[fD].getMinPathAvailabilityIndex();
28         if (minPathAv < pathAvailability || (minPathAv == pathAvailability && routingTable[fD][
               indMin].cost > cost)) {
29             routingTable[fD][indMin].pathAvailability = pathAvailability;
30             routingTable[fD][indMin].nextHop = source;
31             routingTable[fD][indMin].cost = cost;
32             //send info to neighbours
33             for(node j : neighbours(n)) {
34                 if(j != source && j != fD) {
35                     Message M(n, fD, pathAvailability·A_n, cost+c_{j,n});
36                     messageList[j].add(M);
37                 }
38             }
39         }
40     }
41     messageList[n].remove(M);
42 }
```

Figure 5: Pseudo-code performing the reception of a routing message.

message identifier, if the paths are partially overlapped, the first node of the overlapped path portion is asked to forward two messages with the same identifier. At the second occurrence, the node could block the forwarding of the second message since from it to the destination the two paths are overlapped ant this message will follow the same path of the previous one. As an example, consider the following paths from s to d: $s, n'_1, ..., n'_k, m, ..., d$ and $s, n''_1, ..., n''_l, m, ..., d$. The first portions of these paths are disjointed, while they are overlapped from m to d. When the node s selects both these paths to send a message to d, it sends to n'_1 and to n''_1 a replica of the message. During the paths, m receives two messages from s with the same identifier. So, it recognises the overlap, blocks the message forwarding and sends the information to s. The node s receives the information about the overlap and annotates the paths as overlapped. From now on, the source node s could decide to avoid the usage of both these paths to send a message towards d.

4 THE SAN MODEL

The routing algorithm described in Section 3 is here modelled in order to be simulated and to verify its performance. The adopted formalism for the model realization is the Stochastic Activity Network (SAN) formalism (Meyer et al. 1985). The SAN formalism was born to analyse concurrency, timeliness, fault tolerance and degradable performance of complex computing systems. SANs are stochastic extensions of Petri nets and use four primitives: places, activities, input gates, and output gates. The expressive power of the SAN formalism is raised up by the possibility to use C++ code in all the primitives of the modelling languages. Two types of activities are allowed in SANs: instantaneous and timed. Instantaneous activities fire immediately after their activation, while timed activities wait for a duration depending on the associated time distribution function after their activation. Input and output gates can be used to control the "enabling" condition of an activity and to change the state of the system when the activity "fires". The input predicate, which enables an activity connected to an input gate, is given as a C++ predicate, and the input and the output functions, which are executed when an activity fires, are expressed as a sequence of C++ statements. An activity is enabled when the predicates of all input gates connected to the activity are evaluated as true, and each place connected to incoming arcs contains at least one token. When an activity fires, the input and the output functions of input and output gates (respectively) are executed, while tokens of connected places are updated as in the Petri Net firings. Note that the SAN formalism allows the usage of special places, i.e., the *extended places*, enabling to handle the representation of variables, structures and arrays of primitive data-types (i.e., *short*, *int*, *long*, *float*, *double*, *bool* and *char*). Finally, SAN models can be composed by replicating and/or joining different sub-models. The composition is made basically by overlapping places, hence by sharing their state variables. The choice of the SAN formalism to model the network nodes offered the possibility to use extended places, arrays and matrices to store the complex data structures needed in the model. Moreover, the possibility to use model replication allow for creating simple models of network nodes and replicate them to build the overall network model. We preferred this approach to simulation instead of using a typical network simulator to separate our routing algorithm from the specific network layer at which it could be implemented. Let us note that our algorithm, for example, could be also realized at application layer.

The overall SAN model has been structured into different sub-models. We defined a routing and a communication model at the node level, which are properly joined and replicated for each node in the network. This replica is then joined together with the network model, that represents the communication links of the network. A first sub-model represents the node routing algorithm and is represented in Figure 6.

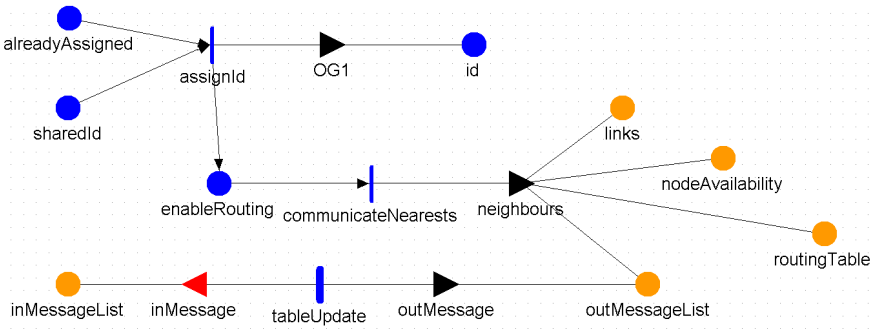

Figure 6: SAN model of the routing component in a node.

The portion at the top of the figure, made of the place *alreadyAssigned*, *sharedId* and *id* is needed to have unique identifiers (from 0 to $n-1$) for each replica of the node. In fact, the place *sharedId* is shared among replicas and initially contains a number of tokens equal to the number of nodes. The place *alreadyAssigned* is not shared and contains exactly 1 token. At the starting of the simulation, the activity *assignId* is enabled

for each replica, and it's firing puts a number of tokens equal to *sharedId* (-1 removed by the firing itself) in the place *id*. When a replica performs a firing, it is no more enabled to fire thanks to the loss of tokens in the place *alreadyAssigned*, while 1 token is put in the place *enableRouting*. This last token enables the firing of the activity *communicateNearests*, which in turn enables the execution of the output gate *neighbours*. This gate executes the code in Figure 4, so communicating a neighbour to the others. The needed information is stored in the extended gates *links*, storing the cost matrix (shared among nodes), *nodeAvailability*, storing the node availability, *routingTable*, storing the entire routing table for the node, *outMessageList* (shared among nodes), storing the message that have to be delivered by the network model. On the bottom of the figure there is another SAN portion that models the reception of routing messages. The extended place *inMessageList* (shared among nodes) stores the message to be analysed. The input gate *inMessage* checks for the presence of routing messages. If a message is present, the activity *tableUpdate* fires and activates the output gate *outMessage*, which executes the code in Figure 5. If some new messages need to be sent, they are stored in the extended place *outMessageList*.

Figure 7 depicts the communication model. For sake of simplicity, this model performs just an evaluation of the paths that have been found for a specific destination (it corresponds to the maximum number of replicated messages the node can send). Indeed, after the presence of a token in the place *endRouting* that represents the routing stability condition, the transition *sentMessage* is enabled and the output gate *send* chooses non deterministically a possible destination and sums to the extended place *numMessages* the number of replicated messages that could be sent to that destination.

The SAN network model is depicted in Figure 8. It is a simple model that performs a step-by-step simulation. Indeed, it waits for the complete consumption of the messages in the current step (input gate *waitMessageConsumption*) and, when no more messages have to be consumed by the nodes, copies the messages produced in the current step, stored in the extended place *outMessageList*, in the extended place *inMessageList*, so enabling the execution of a new step.

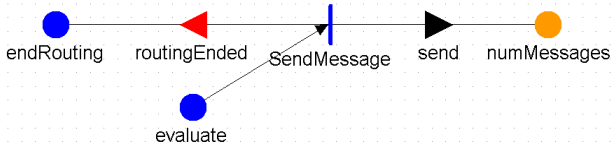

Figure 7: SAN model of the communication component in a node.

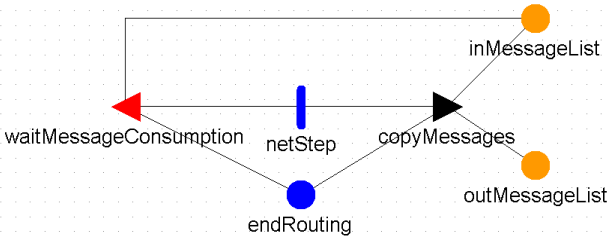

Figure 8: SAN model of the network.

5 SIMULATION RESULTS

In order to describe the potentialities and the performance of the proposed algorithm, in this section we report the results of some analysis, conducted by simulation on the SAN model. We evaluated the scalability of our algorithm, i.e., we analysed the number of routing steps needed by the routing algorithm to reach stable routing tables for each node in the network. We varied the number of nodes in the network from 5 to 25, and we generated random topologies with a maximum degree for each node varying from 3 to 5. Hence, we analysed a total of 63 configurations. Note that, a random topology has been generated starting from the line topology and adding arcs up to reach the desired maximum outgoing degree. For each configuration, we performed a total of 10.000 simulations in order to calculate the average value.

Figure 9 depicts results as a graph, where the x-axis reports the number of nodes in the network and the y-axis reports the number of steps. On the graph, the three lines reports the measured average number of

steps needed from the network to end the routing algorithm. It is possible to notice that, with a maximum node degree of 2, the system does not scale (note that possible topologies with degree equal to 2 are line and ring). By increasing the maximum degree to 3 and to 4, the scalability is better preserved. In fact, the configurations with a maximum degree of 4 reaches the stability in a lower number of steps with respect to the configurations with a maximum degree of 3. We also performed an analysis on the same 63 configurations to check the mean number of paths available among a whatever couple of nodes in the network. The results are depicted in Figure 10. It is possible to note that, except the configuration with a maximum node degree equal to 2, the mean value of available paths is almost high.

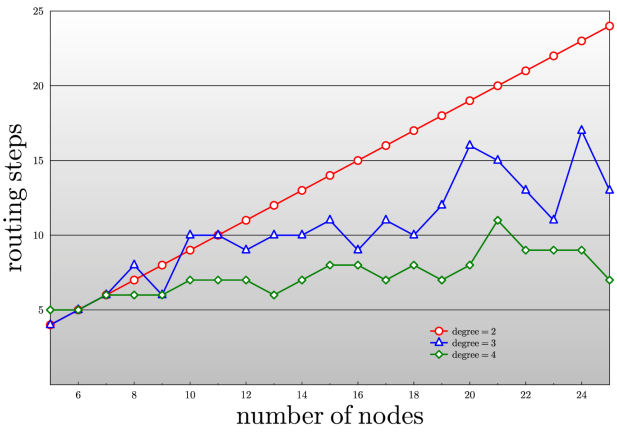

Figure 9: Scalability analysis of the algorithm.

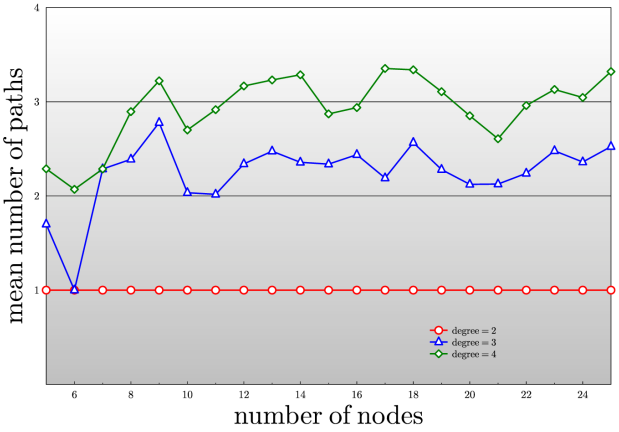

Figure 10: Performance analysis of the algorithm.

6 CONCLUSIONS AND FUTURE WORK

This paper proposes a routing algorithm that aims at individuating different communication paths in a smart distribution network. The main advantage of this algorithm is that nodes can exploit multi-path routing to increase the delivery probability of messages. We also define a mechanism to identify possible path overlaps. The proposed algorithm is then modeled in the Stochastic Activity Network formalism and first results demonstrate a good scalability level and good performance. As future work, we plan to apply the proposed algorithm in smart appliances in order to analyse the benefits at application level. Furthermore, we plan also to extend the routing algorithms to find paths with an increased value of trust.

REFERENCES

Chakraborty, M., N. Deb, and N. Chaki. 2017. "POMSec: Pseudo-opportunistic, multipath secured routing protocol for communications in smart grid". In *IFIP International Conference on Computer Information Systems and Industrial Management*, pp. 264–276. Springer.

D'Arienzo, M., M. Iacono, S. Marrone, and R. Nardone. 2013. "Petri net based evaluation of energy consumption in wireless sensor nodes". *Journal of High Speed Networks* vol. 19 (4), pp. 339–358.

Egert, R., A. Tundis, S. Roth, and M. Mühlhäuser. 2018. "A Service Quality Indicator for Apriori Assessment and Comparison of Cellular Energy Grids". In *International Conference on Sustainability in Energy and Buildings*, pp. 322–332. Springer.

Fang, X., S. Misra, G. Xue, and D. Yang. 2012. "Smart grid—The new and improved power grid: A survey". *IEEE communications surveys & tutorials* vol. 14 (4), pp. 944–980.

Gentile, U., S. Marrone, N. Mazzocca, and R. Nardone. 2016. "Cost-energy modelling and profiling of smart domestic grids". *International Journal of Grid and Utility Computing* vol. 7 (4), pp. 257–271.

Gungor, V. C., D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke. 2011. "Smart grid technologies: Communication technologies and standards". *IEEE transactions on Industrial informatics* vol. 7 (4), pp. 529–539.

Hayden, E. 2010. "There is No SMART in Smart Grid without secure and reliable communications". *Energy & Utilities, white paper*.

Hong, Y., D. Kim, D. Li, L. Guo, J. Son, and A. O. Tokuta. 2014. "Two new multi-path routing algorithms for fault-tolerant communications in smart grid". *Ad Hoc Networks* vol. 22, pp. 3–12.

Kabalci, Y. 2016. "A survey on smart metering and smart grid communication". *Renewable and Sustainable Energy Reviews* vol. 57, pp. 302–318.

Lee, S. W., S. Sarp, D. J. Jeon, and J. H. Kim. 2015. "Smart water grid: the future water management platform". *Desalination and Water Treatment* vol. 55 (2), pp. 339–346.

Meyer, J. F., A. Movaghar, and W. H. Sanders. 1985. "Stochastic Activity Networks: Structure, Behavior, and Application". In *International Workshop on Timed Petri Nets*, pp. 106–115. Washington, DC, USA, IEEE Computer Society.

Saputro, N., K. Akkaya, and S. Uludag. 2012. "A survey of routing protocols for smart grid communications". *Computer Networks* vol. 56 (11), pp. 2742–2771.

Siano, P. 2014. "Demand response and smart grids—A survey". *Renewable and sustainable energy reviews* vol. 30, pp. 461–478.

Vaidya, B., D. Makrakis, and H. Mouftah. 2012. "Secure multipath routing for AMI network in Smart Grid". In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, pp. 408–415. IEEE.

Yan, Y., Y. Qian, H. Sharif, and D. Tipper. 2013. "A survey on smart grid communication infrastructures: Motivations, requirements and challenges". *IEEE communications surveys & tutorials* vol. 15 (1), pp. 5–20.

Yigit, M., V. C. Gungor, and S. Baktir. 2014. "Cloud computing for smart grid applications". *Computer Networks* vol. 70, pp. 312–329.

AUTHOR BIOGRAPHIES

FRANCESCO BUCCAFURRI is a Full Professor of Computer Science at the University Mediterranea of Reggio Calabria, Italy. His research interests include information security and privacy, social networks, deductive-databases, knowledge-representation and non-monotonic reasoning, model checking, data compression, data streams, agents, P2P systems. He is in the editorial board of international journals, and played the role of PC chair and PC member in many international conferences. His email address is bucca@unirc.it.

LORENZO MUSARELLA is a PhD Student at the University Mediterranea of Reggio Calabria. He received his M.D. in Computer Engineering and Telecommunications Systems from the same university in 2017. His research includes network and availability analysis, trust and cybersecurity with a particular focus on social and communication networks. His email address is lorenzo.musarella@unirc.it.

ROBERTO NARDONE is an Adjunct Professor at the University Mediterranea of Reggio Calabria. His research interests include quantitative evaluation of non-functional properties, with a particular focus on dependability and performability assessment and threat propagation analysis. He has been involved in research projects with both academic and industrial partners. His email address is roberto.nardone@unirc.it.