A NOVEL SECURE AND MULTIPATH ROUTING ALGORITHM IN WIRELESS SENSOR NETWORKS

Bayrem Triki, Slim Rekhis and Noureddine Boudriga
Communication Networks and Security Research Lab, University of the 7th of November, Carthage, Tunisia

Keywords: Secure multipath routing, k-x-connectivity, Wireless sensor networks.

Abstract: Multipath routing in Wireless Sensors Networks (WSNs) is used in order to tolerate node failures and improve the reliability of data routing. To make a multipath routing algorithm adaptive to the sensitivity of the used application, communicating nodes in the network should be able to specify to which extent the established paths are disjoint. In this paper, we propose a novel routing algorithm for Wireless Sensor Networks (WSN) entitled Secure Multipath Routing Algorithm (SeMuRa). We extend the concept of k-connectivity to k-x-connectivity where x is the value of threshold representing the maximal number of nodes shared between any two paths in the set of the k established paths. The proposed algorithm: a) is based on on-demand routing; b) uses labels in the datagrams exchanged during the route discovery to carry the threshold x; and c) is secure thanks to the use of threshold signature. A simulation is conducted to prove the efficiency of the algorithm and estimate the additional overhead.

1 INTRODUCTION

Wireless sensor networks (WSNs) consist of a large number of small devices called sensor nodes which are equipped with a radio transceiver and an antenna, a processor, a memory, and a short lifetime battery. These sensors are constrained in terms of computational, and communication capabilities . WSNs are mainly used for collecting data and sensing samples, and sending them to a (central) site in order to be analyzed. Unlike wired networks, sensor nodes can be deployed in hostile environment, making them vulnerable to physical and logical security attacks .

WSNs are characterized by their infrastructureless nature, ease of deployment, and independence to any pre-existing architecture which make the design of routing protocols for WSNs a challenging problem. In fact, routes may be easily broken due to nodes mobility and interference occurrence. Moreover, links between nodes may have limited bandwidth and nodes may lack resources. To alleviate these problems, algorithms for multipath routing were developed as an alternative solution to increase the efficiency of the routing process. They exploit resource redundancy and diversity to enforce fault tolerance, load balancing, and end-to-end delivery delay minimization(Law et al., 2007).

k-connectivity, is a concept used by multipath routing. It defines a minimum number of *k* different paths between two communicating nodes. Before sending data, a sensor node should be sure that a set of *k* paths are available between itself and the base station (*BS*). Then it sends duplicated copies of data over the alternative paths to decrease the probability of communication failure (H.M. and A.E., 2009). Several propositions were made by the literature to design multiple routing algorithms. For instance, Multipath On-Demand Routing Algorithm (MDR) (Dulman et al., 2003) ensures the establishment of disjoint paths between the source and the destination. It splits the original data packet into k parts, and sends these new subpackets instead of the whole packet across available paths. The destination, receiving one of the route request messages, will only be aware of the existence of a path. It returns a route reply containing the number of hops it traveled so far. Each node that receives a route reply, increments the hop count of the message and then forwards the message to the neighbor from which it got the original route request. This solution may cause a lot of overhead in the network. In dynamic multi-path source routing (DMSR) (Yang and Huang, 2008), each node writes its bandwidth into forwarded packets in order to find better paths based on the available bandwidth. Best Effort Geographical Routing Protocol (BEGHR) exploits nodes position to forward data, and requires the use of a positioning system such as Global Position Systems (GPS). However, the demands for resources at the different nodes are quite high, which affect battery lifetime.

Depending on the number of available nodes, established routes may not be totally disjoints. While most of the multipath routing algorithms have taken interest to the establishment of maximal disjoint paths, node performance may be severely affected and resources may be quickly exhausted, especially if the routes include many hops and the nodes density is high. To cope with this issue, it would be useful to extend the multipath routing algorithms so that a disjointness threshold will be defined and exploited during the routes discovery to create a trade-off between fault-tolerance and performance.

In this paper, the concept of k-connectivity is extended to *k*-*x*-connectivity where *x* is a disjointness threshold representing the maximal number of nodes shared between any two paths in the set of *k* established paths. This concept is applied by a secure on-demand routing algorithm, called SeMuRa. It uses labels in the datagrams exchanged during the route discovery to carry the threshold *x*. SeMuRa is secure thanks to the use of elliptic threshold signature and exploits the watchdog concept to tolerate several types of routing attacks.

The paper contribution is five-fold. First, the proposed routing algorithm is adaptive. It allows a source node to tune the disjointness threshold to a suitable value before establishing a path and sending data. The value of the threshold may depend on the sensitivity of the message to be sent, or the rate of broken routes during previous communications. Second, thanks to the use of threshold signature, the algorithm is secure and does not require an extensive number of stored keys per sensor node. The technique of sur-signature could be used for the generation of evidences, which could be used by a digital investigation scheme to prove the identity of malicious nodes, and trace and analyze the attack. Third, the algorithm is tolerant to a large set of routing attacks such as wormhole. Fourth, SeMuRa takes into consideration the characteristics of WSNs, in terms of architecture, nodes resources limitation, and categories of attacks. Fifth, the variation of the network topology could make nodes unable to establish multiple disjointness paths to the destination. If the WSN is used for some application requiring a high level of tolerance to nodes and link failures, by tuning the value of the disjointness threshold, nodes could cope with topology variation and continue benefiting from a degraded level of tolerance.

The remaining part of the paper is organized as follows. The next section describes the requirements to be fulfilled by a secure and multipath routing algorithm. Section 3 describes the proposed routing algorithm. Section 4, presents security mechanisms used by the proposed algorithm. In section 5, a validation of the proposed algorithm is addressed. Section 6 presents and discusses simulation results. The last section concludes the work.

2 REQUIREMENTS OF K-X-CONNECTIVITY

This section describes the requirements to be fulfilled by a multipath routing algorithm based on the concept of *k*-*x*-connectivity. First, the proposed algorithm should be tolerant to attacks on routing protocols. In wormhole attacks, for instance, a node can perform a high-powered transmission of a route request datagram to a non neighbor node, forcing the routing algorithm to include it in the established routing path. This attack, makes the malicious node to appear as a highly connected node, while, in reality, it is connected to few number of nodes. The proposed algorithm should verify that packets are properly forwarded in the networks and identities of intermediate nodes are appended securely to the routing requests. Second, the algorithm should include the generation of evidences regarding the identities and behavior nodes involved during the establishment of the multiple routes. This is of utmost importance if a digital investigation scheme is used to traceback an occurred attack, locate the malicious nodes, and prove the existence of fake routes.

Third, it would be better that the algorithm be reactive rather than proactive. In fact the source node is usually the node that specifies the disjointness threshold value. This value may depend on the sensitiveness of transmitted data. Fourth, the algorithm must be distributed where intermediate nodes should start learning and gathering information regarding potential available path as soon as the route request datagrams propagate. Since nodes in WSNs have limited energy, and consequently limited network lifetime, it is essential to share routes computation tasks between nodes. In addition, distributed computation have a better chance to withstand failure in the case of attacks. Fifth, the routing algorithm must preserve the network performance. Especially, the overhead caused by the storage of information regarding potential usable routes, and the distributed computation of the routing paths by intermediate nodes in the networks, should be reduced.

3 DESCRIPTION OF THE PROPOSED ROUTING ALGORITHM

The proposed multipath routing algorithm, SeMuRa, extends the Dynamic Source Routing (DSR) (Johnson and Maltz., 1996) algorithm, which is a reactive routing approach, widely used as a basis for a large set of extended routing protocols. We consider a tracking-based WSN where a set of sensor nodes are randomly deployed in the network, and are in charge of sensing activities from the surrounding environment and sending the collected data to a *BS*. Sensor nodes communicate with the *BS* by forwarding datagrams through their neighbors. Compared to sensor nodes, the *BS* is typically endowed with sufficient computational resources, and high storage capacity.

3.1 SeMuRa Phases

This protocol includes two steps: route discovery and route maintenance, which allow to discover and maintain possible multiple paths between any sensor source node, say *S*, and the *BS*.

Route Discovery: is the mechanism used when *S* wants to establish a set of paths with the *BS*. Route Request datagrams, say *RReq*, are sent by *S* when it does not already have a route to *BS*. The entirely on-demand properties allows SeMuRa to minimize the overhead and specify the path-disjointness threshold value. After receiving list of potential paths, *S* computes all paths to the destination which satisfy the specified threshold, chooses the list of paths to be used, caches the remaining ones, and starts sending the data. Keeping information regarding unused paths, allows the reaction to routes modification to be rapid and decreases the overhead related to the generation of a new *RReq*.

Route Maintenance: is the mechanism used to let *S* update the list of paths in use if the network topology changes, or some routes are broken due to an attack or sleeping cycles of nodes. This mechanism is based on letting intermediate nodes use the watchdog concept for every packet they forward (Lee et al., 2007) to detect the identities of misbehaving nodes or detect routes errors. If the next hop appears to be broken, a route error packet, say *REtr*, is generated and sent to *S* in order to decrease the number of possible path to the destination. *S* will consider all the path as broken and attempts to use another route that go over the non responding stored in its cache which allows to maintain the *k*-*x*-connectivity. If none backup route to the *BS* is in the cache, the source node invokes again the Route Discovery mechanism.

SeMuRa can be easily extended to the context of Adhoc networks, where the set of features performed by the *BS* would be supported by any node in this network, and the destination to which a node would generate multiple routes, would be any one of them.

3.2 SeMuRa Route Discovery

Five kinds of datagrams are used by SeMuRa during the route discovery phase: (a) Route Request: is the first packet to be broadcasted by a node who wants to establish a multipath route to the *BS*. Every intermediate node exploits this datagram to discover incomplete routes in the network. It also appends its identity in the *RReq* and broadcasts it to its neighbors. (b) Route Response: is sent back by the *BS* upon reception of the *RReq*. This datagram contains the optimal path and is source routed to the node which generated the *RReq*. (c) Notification : used by the destination node to ask intermediate nodes to forward the information they learned regarding the routes to the source node. These information would have been invisible by the *BS* when it received the *RReq* datagram. (d) List forwarding: used by intermediate nodes to forward the information they stored regarding the existing paths in the network. (e) Route Error: sent by an intermediate node to the source node when it detects a route failure. It also lets the source node update the set of paths it uses to reach the *BS*.

Network Discovery. When a node, say *S*1, joins the network, it broadcasts a two-hop HELLO message, which includes its identity and has a Time To Live (TTL) value equal to 2. Any node, say *S*2, which hears the message includes the identity of *S*1 in its list of one-hop neighbors, sets the TTL value of the HELLO message equal to 1 lower than its received value, and forwards the datagram. Any node, say *S*3, that hears the message includes the identity of *S*2 in its list of one-hop neighbors, and *S*1 in its list of two-hop neighbors, sets the TTL value of the HELLO message equal to 1 lower than its received value, and discards the datagram. To be considered as active, every node should periodically send a two-hop HELLO message and follow the above described process. This allows each node to maintain two up-to-date lists: its neighbors list, and the neighbor list of its neighbors. The two lists will support the detection of routing attacks (described later in section 4).

Route Request Generation and Forwarding. A node which wants to establish a path to the *BS*, initiates the route discovery by generating a *RReq* datagram to the *BS* and broadcasting it in the network. Every generated *RReq* includes a three-tuple information: $\langle Seq, RRec, Dt \rangle$. *Seq* stands for a sequence number which should be different for every new generated *RReq*. The sequence number, together with the IP address of the sender, allow to uniquely identify the *RReq* and associate it to the subsequent generated responses. *Dt* is the disjointness threshold which is set by the sender to specify the maximal number of nodes that could be shared by any two paths among the set of paths to establish with the *BS*. The value of *Dt* remains unchanged during the forwarding of the *RReq* to the destination. *RRec* is a route record which is used to include the path followed by the *RReq* to reach the *BS*. In fact, when a node in the network receives a copy of this datagram for the first time, it appends its identity to the *RRec* field, and broadcasts it to its neighbors. Every node, say *N*, including the *BS*, which receives a second copy of the datagram, extracts the content of the *RRec* field. The latter provides a path from the sender to the node *N*. The node *N* will append the content of *RRec* together with the value of *Seq* to a list, entitled *RP*, which stands for list of Received Paths. Then it discards the datagram.

Route Response Generation and Forwarding. Once different copies of the *RReq* datagram reach the destination (i.e., the *BS*), the latter will generate a Route Response datagram, say *RRep*, to the source. It includes three-tuple information $\langle Seq, R, RNC, RPBS \rangle$ where *Seq* represents the value of the sequence number that appeared within the received *RReq*, *R* is the route, which is composed of the sequence of nodes identities representing the shortest path (between the source and the *BS*) among those that were received within the different copies of the *RReq*s. *RCN* stands for the remaining number of common nodes. It is initiated by the *BS* to the value of *x* received within the *RReq*. This value is decreased by 1 every time the *RRep* is routed by a node which contains a non empty list of received paths for the same value of the sequence number. The *RRep* datagram will be source routed to the sender based on the content of *R*. Moreover a list, say *RPBS* ,containing the list of all the routing paths connecting the source node to the *BS* is added to the *RRep*. When the *RRep* packet is routed to the source, the latter and all intermediate nodes will discover a route to the *BS*, store it in their cache, and use it as an alternative path if some link error will potentially occur. We remind the reader that any different routes requests, to be received by the *BS*, may share some nodes. Every one of them can be written as a series of nodes shared with other routes, followed by a series of distinct nodes.

Notification Datagrams Generation and Forwarding. In the case where the *BS* has discarded a copy of the *RReq*, it generates a notification datagram, say *NP*, containing the three-tuple information $\langle Seq, RCN, L, RPBS \rangle$ composed of the sequence number (*Seq*) and the value of *RCN* received in the *RRep*, in addition to a list, say *L*, containing the identities of neighbor nodes from which a received copy of the *RReq* was previously discarded (i.e., the identities of these neighbors stand for the last nodes in the routing paths provided by *RP* and related to the sequence number *Seq*). If the case where the *NP* is sent by the *BS*, the list *L* will be set to the identities of neighbor nodes from which a copy of the *RReq* was received. *RPBS* is a list containing the set of routing paths connecting the source node to the *BS*, including the shortest path. These routes are collected from the copies of the *RReq* received by the *BS*. The *NP* is sent to the source node, broadcasted but treated only by nodes existing in the list *L*.

In the case where some node in the network, say X, receives the *RRep* two situations may happen. It X has already discarded, at least, a copy of the related *RReq*, it forwards it after decreasing the value of *RCN* and generates an *NP* containing the three-tuple information $\langle Seq, RCN-1, L, RPBS \rangle$. If it is not the case, it simply forwards the datagram to its neighbors. When an intermediate node, say X, receives an *NP* for the first time, two situations may happen. If X has not previously discarded any copy of the related *RReq*, it simply forwards the *NP* to its neighbors. If X has already discarded, at least, a copy of the related *RReq*, it forwards it after decreasing the value of *RCN* and replacing the value of *L* by the identities of neighbor nodes from which a received copy of the *RReq* was previously discarded. When a node receives a second copy of the *NP*, it simply discards it. If the value *RCN* becomes equal to 0 after decreasing it by one, the notification packet will be rejected before sending it.

List Forwarding Datagrams Generation and Forwarding. Every node which decreases the *RCN*'s value of the *NP*, generates a list forwarding datagram containing the sequence number (already received in the *NP*) and a list obtained from *RP* (the sequence number associated to *RP* should be the same as the one received in the *NP*) after applying two filters, say $F1$ and $F2$, consecutively. The list forwarding datagram is sent to the source node (i.e., the node which initiated the *RReq*). The first filter $F1$ eliminates from *RP* any path that has more than $RCN-1$ shared nodes with any path existing in *RPBS*. The second filter $F2$ locates in the output of $F1$ groups of nodes that share more than $RCN-1$ nodes. It replaces each one of these groups in the *RP* list by the shortest path. When the source node specifies an x equal to 0 (i.e., all the discovered paths must be disjoint), the *NP* will be sent with a value of *RCN* equal to 0. Intermediate nodes receiving this latter and have an empty *RP*,will forward the packet to their neighbors. If it is not the case, they drop the *NP*. Each time a node sends its *RP* list to the source node, it eliminates this list from its memory to preserve storage resources. If there is no additional space in the node memory, a solution consists in using the neighbor memory. Two categories of nodes can be used: nodes with high storage capacity and nodes with limited storage capacity. Each node knows the category of its neighbors. A node with a low storage capacity has the possibility to send parts of the data it stores only to neighbor nodes with high storage capacity. In fact, the receiving node should send back the data to the sender before it goes out of its coverage or sleeps. If the sender memory is still full, the receiving node should find a neighbor node, which is also a neighbor of the sender and has a high storage capacity, transfer the data to that node, and inform the sender about its identity.

3.3 Reconstruction of Routes by the Source

The aim of the reconstruction process is to construct k paths satisfying the disjointness threshold x. If several combinations are possible, the sender could select the one which uses the minimal number of shared nodes, or select the one which uses the shortest k possible paths. The first alternative could be chosen if availability is more sensitive than delivery delays. We remind, that x's value specified in the *RReq*, has prevented intermediate nodes to forward useless list of paths, which if assembled together by the source node, would generate routes that include a number of shared nodes higher than it is expected by x.

To reconstruct the set of paths, the following algorithm is executed. Let L_{cp} be the list of complete paths satisfying the threshold x to generate by the algorithm, and L_p be the series of path received in the different list forwarding datagrams sent by intermediate nodes in the network. $L_p =< p_1, \ldots, p_n >$ where every $p \in L_p$ represents any path, if the form of $< S_1, \ldots, S_{bl} >$, where: a) S_1 represents the identity of the node, which initiated the route discovery; and b) S_{bl} represents the identify of any intermediate node in the network including the *BS*, which discarded a second copy of the *RReq* datagram.

Starting with a FIFO queue, say $\mathcal{Q}$, containing the set of paths in *RPBS* received by the *BS*. Until $\mathcal{Q} = \langle\rangle$ or $L_p = \emptyset$, the algorithm do: Let $R =< S_1^{'}, \ldots, S_n^{'} >$, be the first routing path in $\mathcal{Q}$, where $S_1^{'}$ stands for the source node and $S_n^{'}$ stands for the *BS*. Starting from a value of i equal to n, and until i becomes equal to 1 (i.e., i is decreased by 1 in every loop), the algorithms checks for every $p \in L_p$ if $S_i \in R$ corresponds to the last node $S_{bl} \in p$. If it is the case, the source node will generate a path equal to $\langle S_1, \ldots, S_{bl}, S_{i+1}^{'} \ldots, S_n^{'} \rangle$, appends it to L_{cp} and $\mathcal{Q}$, and deletes p from L_p. If i becomes equal to 1, the algorithm deletes R from $\mathcal{Q}$. For the particular case where the *BS* has generated an *RPBS* containing paths which share more than x nodes, the L_{cp} needs to be filtered by keeping the shortest path from those having more than x nodes shared with paths in the *RPBS* . Based on the content of L_{cp}, the source node will be able to select the best combination of paths satisfying the values of k and x.

3.4 Route Maintenance

A path can fail due to collisions, and/or nodes mobility. It is essential to recover broken paths immediately to ensure the reliability of data. In SeMuRa, after routes establishment and during data forwarding, when a node fails to send the packet to the next hop, or detects that a neighbor is not forwarding the datagram, it considers the route as broken and sends a *RErr* to the source node to inform it about the identity of the unavailable node. This mechanism is strengthened by applying a watchdog mechanism described in the section 4. Upon reception of this *RErr* packet, the source node deletes any alternative route in the k established path that uses the broken node. It tries to replace it by one of the available routes in its cache, and verifies if the set of k paths to use still have, at the maximum possible, x common nodes. If it is not the case, the route discovery step is initiated again.

Lemma 1. *All the paths reconstructed at the source node do not share a number of nodes higher than the pre-defined disjointness threshold with the shortest path.*

Proof. We remind that a route response uses the shortest path to reach the source node and contains the pre-defined disjointness threshold x of shared nodes. Given a node, say n, whose identity is part of the shortest path, and which has already discarded a duplicated copy of the *RReq*. If node n forwarded that packet, the latter would have reached the *BS* using the same nodes existing between n and *BS* within the shorted path, and the *BS* would have obtained two routing paths sharing a number of nodes, say *snn*, composed of all nodes between n and *BS*. The process of generation and forwarding of the *RRep*, which includes the decrementation of x's value, allows to detect whether the value of *snm* exceeds this threshold.

In addition, given a node, say n', whose identity is part of the shortest path, and which has broadcasted the *RReq* to its neighbors, where one of them is part of path, say p, different from the shortest path. As the path p will share with the shortest path the set of nodes between the source node and the node n'. Applying filter $F2$ on the output of the filter $F1$ will take into consideration shared node between the source node and n'. Consequently paths that share more than x nodes with the shortest path will not be sent to the source node. As a result this path will not be used in the reconstruction process. □

Lemma 2. *Any path in a node's RP, related to the same route request, can be described as the concatenation of two series. The first series can be shared by another path in RP while the second series never occurs in another path in RP.*

Proof. The proof of the first series is conducted as follows. We remind that a *RReq* contains the identities of nodes through which the datagram has been routed. During the route discovery, a node broadcasts the *RReq*, and all its neighbors receive a copy. Given two distinct nodes, which have received the same copy from a neighbor. Each one of them appends its identity to the copy it received and forwards it. Consequently, the routes that will be generated in the *RP* within the other nodes could start with a series of shared nodes. The proof of the second series is based on the fact that a node, which receives a second copy of some *RReq*, does not forward it. Consequently, a node could not append two paths to the RP which share some nodes. □

3.5 Example Applying *k*-*x*-connectivity

In the example presented in Figure 1, the source node A needs to establish three ($k = 3$) routes to the *BS* sharing at maximum two nodes between them ($x = 2$). In the proposed topology, it is hard to respect three disjoint paths without tuning the threshold x. A generates and broadcasts a *RReq* with a disjointness threshold x equal to 2. In the Figure, nodes are represented by circles, and an edge connects two nodes if they are able to directly communicate together. The *RP* lists stored by each node are represented by rectangles. Node B receives the first packet directly from A and receives a second copy though two other paths which are $< A,C,B >$ and $< A,C,E,B >$. As each time node B drops a duplicated packet, it stores the routing path used by this copy in its *RP* list, the routing path *RP* will be equal to $<< A,C,B >< A,C,E,B >>$. The *RReq* is forwarded to neighbor nodes until it reaches the *BS* on the shortest path $R_{sp} =< A,B,D,F,BS >$.

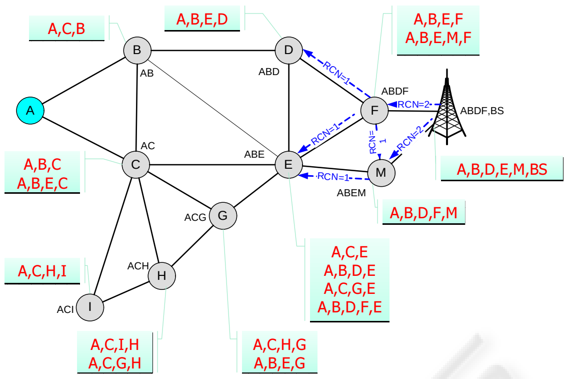

Figure 1: Response process.

Based on the number of minimum common nodes *RCN* which is set to 2 in the *NP*, the RP lists are only sent by nodes F, M, D and E. In fact, at the response step, when Node F receives the *NP* from the *BS*, it decreases the value of *RCN* to 1 and forwards this packet to its neighbors. This node applies the filter $F1$ on its *RP* list based on the content of the *RPBS* list received for the *BS*. Note that this *RPBS* contains the two following paths $< A,B,D,F,B,S >$ and <A,B,D,E,M,BS>. Using the filter $F1$, paths $< A,B,E,F >$ and $< A,B,E,M,F >$, in the *RP* list of the node F,will be eliminated because they have more than $RCN - 1 = 1$ shared node with the *RPBS*. As the *RP* list of node F is empty, its *RP* list will be sent back to the source node. When node M receives the *NP* from the *BS*, it decreases *RCN* o 1 and applies the two filters $F1$ and $F2$ on its *RP* list. By applying $F1$, node M, will discard the path $< ABDFM >$ from its *RP* list, because it has more than $RCN - 1 = 1$ shared nodes with the first path $< A,B,D,F,BS >$ of the *RPBS* list. When the node M receives a second copy of the *NP* from the node F it will discard it.

When nodes D and E receive the *NP* from node F, they decrease the value of *RCN* to 0. By applying the two filters $F1$ and $F2$, node D will not send the path $< A,B,E,D >$stored in its RP list. In fact, the use of filter $F1$ lets node E keep only paths $< A,C,E >$ and $< A,C,G,E >$ in its *RP* list. However, $< A,C,E >$ and $< A,C,G,E >$ share node C between them and node E accepts only $RCN - 1 = 0$ common nodes. Therefore, when it applies the filter $F2$, node E is forced to keep only the shortest path, which is $< A,C,E >$. When the node E receives a second copy of the *NP* from the node M, it discards it. When the neighbors of nodes D and E receive the *NP* with an *RCN* set to 0, they discard this packet without sending any information. At the end, the source node A, has a list of paths available to the *BS* which is equal to $L_p = \{< A,C,E >\}$. The *RPBS*, which is extracted from the *RRep*, together with the list of paths L_p will be used to determine the set of routes, characterized by $k = 3$ and $x = 2$, to the *BS*. By applying the reconstruction algorithm described in Subsection 3.3, the final list of reconstructed paths L_{cp}will be equal to: $L_{cp} = \{< A,C,E,F,BS >< A,B,D,F,BS >< A,B,E,M,BS >\}$.

4 SECURITY OF SEMURA

To provide a secure routing algorithm, against a set of attacks, that prepare for the investigation, three main properties should be satisfied. First, nodes should be able to authenticate each others during the process of routes establishment. Datagrams generated with forged information should be discarded before reaching the *BS*. Second, every node should not only be in charge of generating and forwarding datagrams to the *BS*, but also of controlling the behavior of its neighbors. In this context, the watchdog technique is used to detect nodes that do not forward the datagrams as expected. A node which uses the watchdog technique is able to determine whether its neighbor nodes are forwarding the datagram they receive or not. If the packet is not forwarded within a certain period, this neighbor is considered as malicious(Lee et al., 2007). Every node should maintain two lists: a list of one-hop neighbors and a list of two hop neighbors. The two lists are created by letting every node periodically perform a two-hop broadcast of a Hello Message (i.e., by setting the TTL equal to 2). A node, say n_1 which receives a generated Hello message by a node, say n_0, with a TTL equal to 2, appends the identity of n_0 to its list of neighbors, appends it own identity (i.e., n_1), decreases by one the TTL, and forwards the packet. A node, say n_2, which receives a datagram with a TTL equal to 1 from the neighbor node n_1, appends the identity of the sender (i.e., n_0) to its list of two hop neighbors, and marks this node as being reachable through the immediate sender n_1. Third, when a node detects a malicious neighbor, both the source node and the *BS* should be informed.

We use a signature scheme to authenticate nodes and guarantee the integrity of the information they exchange. We suppose that every node joining the network is authenticated by the *BS*. Intermediate verification of packets signature allows discarding compromised packets before they reach the destination nodes, which optimizes the used energy and communication resources, and reduces the overhead of the signature verification process performed by the *BS*. Every node which generates or forwards the *RReq*, appends its identity, the identity of the next receiving nodes, and sur-signs the route record. A node which receives the forwarded message verifies whether the last appended signature is correct, checks if it is the presumed destination, determines the immediate sender (the neighbor node from which the packet is being forwarded) of that datagram and makes sure that it is a neighbor. If it is the case, it appends its identity, the identity of the possible next hops and sur-signs the datagram. To perform signature, the elliptic threshold signature provided by (Sliti et al., 2008) is used. It allows to generate for a public key, say k_{pub}, n associated secret keys, say $k_{pr_1},...,k_{pr_n}$. Every signature created using one of the private keys, say k_{pr_i}, can be checked using k_{pub}. In the SeMuRa , every node, including the sensors and the *BS*, uses its own private key for signature, while the same public key is used by all of them for the purpose of signature verification. Such techniques increases the resilience of WSNs against node capture by: (a) using threshold signature to authenticate packet content and discarding invalid datagrams (b) the identification of captured node based on applied watchdog mechanism and intermediate signature (c) to be tolerant to the discard of compromised paths involving captured node by authorizing x shared node, which assures that, even when part of the nodes have been captured, the rest of the network remains secure. Based on threshold signature, if a node is duplicated, and its key is used by a malicious node, the *BS* will notice the attack by detecting that the same key was used by several nodes. To do so, the *BS* checks whether two nodes having the same identities have participated in forwarding the *RReq*. If it is not the case, for each signature appended to the *RReq*, the *BS* identifies the identity of the signer node, and verifies whether it could really produce this signature if its private key was used (the *BS* is assumed to know all the sensors private keys). Note that, the use of the public key is not sufficient to authenticate the nodes, because it does not allow detecting whether the same private key was used several times to generate the sur-signed *RReq*. When the *BS* detects that a node has used a key of another node, or a node has participated several times in the same *RReq*, it forwards an alert containing the identity of the compromised nodes, asking the remaining nodes to reject any packet sent from that node in the future. When a node receives a second copy of the *RReq*, it signs and stores the received path in the *RP* list, where each identity, in the received path, is signed by intermediate nodes. If a malicious node wants to modify the *RP* list, it must uses signatures of all nodes involved in the modified path to re-sign each identity which is impossible. In addition when an intermediate node eliminates a received *RP* list instead of forwarding it, the watchdog mechanism used by neighbor nodes will detect such behavior.

SeMuRa is also protected against a set of routing attacks such as the wormhole attack (Triki et al., 2009), where a malicious node receives packets, tunnels them to another location in the network, and resend them. In the case of an out-of-band channel establishment, a malicious node may collude with another node, which is typically located near the *BS*, to make the routing paths, that go over them, look shorter than expected. Such behavior could compromise the discovery $k-x$-connected paths. By relying on watchdog mechanism and intermediate signature, the wormhole attack will be detected. In fact, a malicious node could forward a packet to non neighbor node using a high powered transmission. Since the node should append its identity and the identity of immediate receivers to the route record and sur-sign them, two situations could happen:(a) If the malicious node, specifies a correct identity of the immediate receiver, the watchdog neighbors, verify the signature in the datagram and detect that the packet was forwarded to a non neighbor node. When the malicious datagram is forwarded to the *BS*, together with the alert generated by the watchdog nodes, the latter could use the signature as an evidence to prove the identity of the malicious node (b) the malicious node specifies the identity of a neighbor node when it signs the route record, but forwards the datagram to a non neighbor node. In that case, the neighbor nodes, using watchdog technique, will detect that the node has appended an identity of a non neighbor node, which will receive the packet. The watchdog nodes will broadcast an alert, containing the identity of the malicious node.

5 VALIDATION OF SEMURA

In SeMuRa, the construction of paths is based on a distributed mechanism where information regarding available links is collected by the *BS*, or stored by intermediate nodes, and copies of a same datagram are discarded. The correctness of the routes generation is vindicated by the fact that exchanged information are signed and verified by neighbors. In addition, the *NP*s generated by the *BS*, or by nodes incorporate the disjointness threshold related to the set of the k established paths. Since this value is decreased whenever the *NP* is forwarded by a node which maintains a path to the source node, all pairs of paths that have more than x shared nodes will not be generated. This allows that nodes to store and forward only the useful information, and reduce the network overhead.

The completeness of SeMuRa is satisfied by the fact that the source node is able to reconstruct all existing and shortest paths to the destination. In fact, SeMuRa broadcasts the *RReq* over all nodes in the network, and makes all the nodes, including the *BS*, able to save all information regarding possible routes from the source node to themselves. In addition, *NP*s are broadcasted to all neighbor nodes, which have discarded a copy of the *RReq*, to let them send to the source node the list of paths they discovered. Using SeMuRa the network overload may increase depending the number of nodes and the value of the threshold x. Especially, a high number of lists of paths may be generated and the number of list of paths datagrams may increase. The traffic overload will be estimated in the next section. In WSN, if nodes are in sleeping state they will not be involved in the route discovery. If the node changes its state at the end of this process it will be considered as a novel node joining the network and will be involved in the next route discovery.

The security of SeMuRa is based on the use of watchdog mechanism and digital signature. The watchdog mechanism allows to capture several types of routing attacks including for instance, the wormhole attacks. In fact, bi-directional links should be used to communicate between sensors, in order to let a node detect whether its neighbor is forwarding the datagram received from another node. False positives may occur if a node detects that its neighbor is malicious because it has not forwarded the datagram, while, in reality, it happens due a collision. False negatives may occur if the state of some node, involved in the route to the *BS*, becomes sleeping or runs out of energy. Such node, which is not detected as inactive yet, may be considered as malicious and leads to the generation of false negative alerts.

The values of the two parameters x and k are highly correlated and both of them depend on the node density. Typically, if the source node chooses a high value of k, it should tend to decrease the value of x to guarantee the establishment of all the paths. It is worth to notice that, for a fixed value of k, the more the node density is low, the lower will be the value of x. Conversely, if the nodes density is high, the value of x could increase with regard to the latter situation.

Two particular topologies could reduce considerably the performance of SeMuRa. The first is obtained when nodes are so close to each other and all of them are located around the *BS*. In this topology, a *RReq* generated from any node will reach all nodes in the network. As a result, a node could receive the same copy of datagrams from all nodes in the network. The nodes memory will be overloaded due to the highest number of paths to store in the *RP* list. The second, is obtained when nodes are not deployed with a sufficient number, and most of nodes do not have more than two neighbors. The routing paths to generate will contain a large set of intermediate nodes. While the memory occupation rate in nodes is highly reduced with regard to the previous topology, a multipath will require a high delay to be established.

6 SIMULATION RESULTS

6.1 Memory Overhead Estimation

When the *RReq* datagram is forwarded, every node, which receives a copy, stores the route record in its list of received paths. Since this list is temporary stored within the sensor node memory, we estimate the average number of stored paths in each node in terms of the number of nodes. We consider a network area of 72×72 length units. The nodes communication radius is set to 10 length units and the *x*'s value is set to 2. Based on these values, the optimal number of deployed nodes is given by the following formula:

$$\frac{Total\,Network\,Area}{(3 \times Node's Coverage\,Area)}$$

and is approximately equal to $50 \backsimeq 3 * 72^2/(10^2\pi)$. In this simulation nodes positions are computed using a uniform distribution of random values. The use of uniform distribution is very common (Karl and Willig, 2007) as it allows to distribute nodes over the whole network area and ensure a homogeneous coverage. If the nodes would have been deployed in non-flat surface containing valleys, rivers and lakes, the use of another distribution such as normal distribution (Krupadanam and Fu, 2008) would be considered.

Figure 2 shows the variation of the average number of stored paths per node, with respect to a number of nodes ranging from 10 to 150. This average is computed during the route discovery phase. It is equal to the $Number\ of\ stored\ paths\ in\ all\ nodes$ divided by the $Number\ of\ deployed\ nodes$. The simulation results indicate that the more the nodes communication radius is, the more will be the memory overhead. Starting from a value of deployed nodes equal to 10, and until this number reaches the optimal value (i.e., 50), the number of received copies of the same *RReq* increases which slowly increases the number of stored paths in each node. In fact, since the node number is lower than the optimal value, the number of unreachable nodes is important. The node, which do not receive any copy of *RReq*, will have a number of stored paths equal to 0. Starting from a number of nodes higher than 50, all the nodes become reachable and the average of stored paths linearly increases as long as the number of nodes in the network increases.

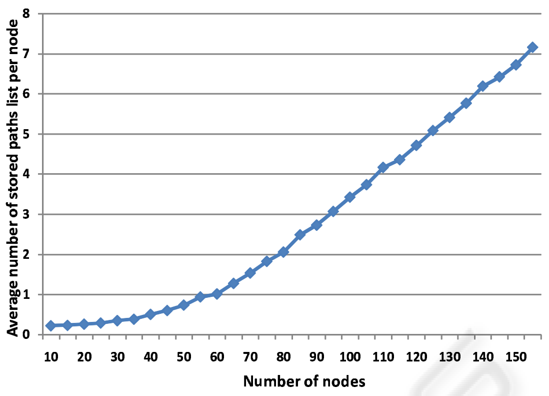

Figure 2: Influence of nodes density on length of stored paths list.

6.2 Communication Overhead Estimation

In order to estimate the communication overhead generated by SeMuRa, we consider a network area equal to 39×39 length units, which holds a variable number of randomly distributed nodes. The nodes communication radius is set to 10 length units and the value of *x* is set to 2. Based on these values, the optimal number of deployed nodes is equal to 15. The curve depicted by Figure 3 shows the estimated average time, per node, required to generate a set of routes satisfying the $k-x$ connectivity scheme, in terms of number of nodes in the network and number of hops separating the source node to the *BS*. The number of nodes was varied from 5 to 195, and the estimated time was computed for different possible hop values.

The simulation shows that the estimated average time, per node, to establish a path, initially increases as the number of nodes increases from 5 to approximately 30. In fact, the network becomes more and more covered, and nodes far from the *BS* become able to establish a path, which increases the average of the estimated time. As the number of nodes becomes far from the optimal value, the estimated average time becomes approximately constant. This is due to the fact, that datagrams generated by the algorithm are always routed through the shortest path. Even during the route establishment phase, the copies of the *RReq* datagrams, which arrive to the *BS*, are the ones which were forwarded through the minimal number of nodes. The simulation shows also that the highest values of hops are obtained for a number of nodes ranging from 20 to 35. In fact, since the network is not sufficiently dense, nodes far from the *BS* will require a high number of hops to reach the *BS*. As the network becomes dense, these nodes become able to reach the

BS using shorter paths. Finally, it is also noticed that as the number of hops increases by 1, the estimated average time regularly increases with approximately 5 periods of time due the static value of the waiting and processing time of datagrams in nodes.

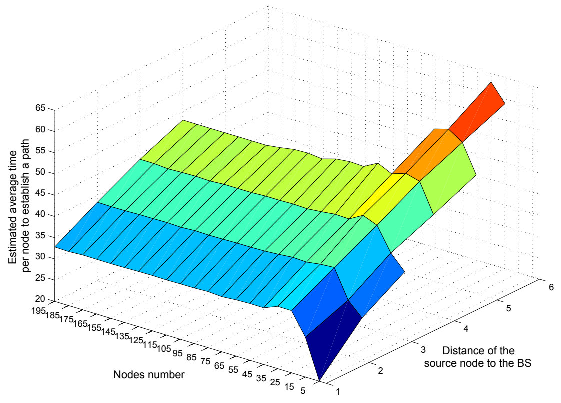

Figure 3: Influences of nodes density on the overhead and the length of paths.

7 CONCLUSIONS

In this work, we described a routing algorithm, called SeMuRa, for maintaining *k*-*x*-connectivity, where *k* stands for the number of paths that should be available between a source and a destination before sending data, and *x* is the maximal number of nodes shared between any two paths among the *k* paths. SeMuRa uses a threshold signature algorithm to authenticate exchanged packets. A simulation is conducted to analyze the memory and communication overhead. Directions for future works include the study of the digital investigation of security attacks on multipath routing algorithms. SeMuRa is used for WSNs but it can also to be adapted to Adhoc networks.

REFERENCES

Dulman, S., Wu, J., and Havinga, P. (April 2003). An energy efficient multipath routing algorithm for wireless sensor networks. *IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2003), Pisa, Italy*.

H.M., A. and A.E., K. (June 2009). On the minimum k-connectivity repair in wireless sensor networks. *in the proceedings of the IEEE International Conference on Communications (ICC).*, pages Page(s):1 – 5.

Johnson, D. B. and Maltz., D. (1996.). Dynamic source routing in ad hoc wireless networks. *In Imielinski and Korth, editors, Mobile Computing, Mobile Computing*, volume 353.

Karl, H. and Willig, A. (2007). *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 1st Edition.

Krupadanam, S. and Fu, H. (July 2008). Beacon-less location detection in wireless sensor networks for non-flat terrains. *International Journal of Software Engineering and Its Applications*, Vol. 2, No. 3,.

Law, Y., Yen, L., Pietro, R., and Palaniswami, M. (2007). Secure k-connectivity properties of wireless sensor networks. *IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems*.

Lee, J., Lee, Y., and Syrotiuk, V. R. (May 2007). The performance of a watchdog protocol for wireless network security. *International Journal of Wireless and Mobile Computing*.

Sliti, M., Hamdi, M., and Boudriga, N. (2008). An elliptic threshold signature framework for k-security in wireless sensor networks. *Electronics, Circuits and Systems, ICECS 2008. 15th IEEE International Conference on*.

Triki, B., Rekhis, S., and Boudriga, N. (2009). Digital investigation of wormhole attacks in wireless sensor networks. *nca, pp.179-186, Eighth IEEE International Symposium on Network Computing and Applications,*.

Yang, P. and Huang, B. (2008). Multi-path routing protocol for mobile ad hoc network. *International Conference on Computer Science and Software Engineering*.