Using Trust for Key Distribution and Route Selection in Wireless Sensor Networks

Nathan Lewis, Noria Foukia, and Donovan G. Govan,
Information Science Institute,
University of Otago PO Box 56, Dunedin New Zealand
{ndlewis, nfoukia, dgovan}@infoscience.otago.ac.nz

Abstract— **This paper presents a new approach of dynamic symmetric key distribution for encrypting the communication between two nodes in a Wireless Sensor Network (WSN). The distribution of a shared key can be performed by any sensor node and does not always require that it is performed by the base station (BS). Each node can be selected by one of its neighbor nodes in order to distribute a pair-wise key for a communication between two nodes. The selection is based on the local computation of a trust value granted by the requesting nodes. This scheme considerably reduces the cost of communication between the BS and the nodes when setting up pair-wise keys between neighboring nodes. This paper also describes a dynamic route selection mechanisms based on trust and cost, that each node performs to route data to neighbor nodes and to the BS.**

***Index Terms*— Key Distribution, Trust, Wireless Sensor Network, Route Selection.**

I. INTRODUCTION

A Wireless Sensor Network (WSN) consists of spatially distributed autonomous nodes called sensors that monitor physical or environmental conditions, such as temperature or pressure at different locations [1]. They are used in a variety of applications, such as climate sensing and control in office buildings. The privacy and security issues posed by WSNs are currently crucial issues of WSN research [2]. There are different ways of compromising sensitive information in WSNs. For example, an attacker can capture traffic from individual sensor nodes, re-program nodes, or introduce her own sensor nodes in the network to be accepted as legitimate nodes. After taking control of sensor nodes, an attacker can alter the sensor data and/or extract private information. Because of the physical characteristics of a sensor network that enables large data collection [2] and makes the sensor information available from remote access, an attacker does not require to be physically located at a sensor node, thereby aggravating the security problem. To minimize this security issue, this paper proposes an efficient dynamic distribution mechanism of pair-wise keys based on the notion of local trust. Any node in the WSN can be selected by one of its neighbor nodes in order to distribute a pair-wise key for encrypting the communication between two nodes. Establishing a trust context will ensure that only trusted nodes within the WSN can share sensed information.

While this method attempts to ensure the reliability of the WSN, it does not guarantee that recorded data cannot later be compromised. For this reason, our approach based on sensor nodes allocating keys should only be used in WSNs where it is less important to keep historical data secret and more important that current communication is reliable: that is detecting possible intruders or weak nodes under attack.

Moreover, in order to send data to the base station (BS), a sensor node will select the best route to the BS via one of its neighbors (that it shares a key with). This paper describes a dynamic route selection mechanism based on trust, that each node performs to route data to neighbor nodes and to the BS.

The structure of the paper is as follows: Section II describes our shared-key distribution protocol and explains how our approach is different from existing approaches of pair-wise key distribution performed by the BS. Section III explains the dynamic trust-routing protocol that we propose. Section IV describes how trust is maintained in the trust-based routing protocol., and finally in Section V, we provide a description of future work and discussion.

II. THE SHARED-KEY DISTRIBUTION PROTOCOL

A. Neighbor-based shared-key distribution

1) BS vs. neighbor-based shared key distribution

There are many ways of defining trust [7][8]. We define the notion of two "trusted" nodes as two nodes that share a private unique symmetric key and as far as each of them are aware, both nodes have not been compromised. Trusted nodes can be immediate neighbors in the radio range of the requesting node or multi-hop nodes not in the radio range. There are already several papers about using the BS to create keys that pairs of nodes share to talk to each other [3][4][6]. In all of these works, the basic idea is that each node has a unique symmetric key with the BS and every time that a node A wants to talk to node B, it asks the BS for some information about node B. For example, the BS may return a message confirming that node B

is trusted and a token[1] that node A can show to node B so that node B knows that it can trust node A. Kerberos is one such protocol [5].

From our reading of the different works dealing with pair-wise key distributions in sensor networks, it has been observed that they all require the nodes to communicate with the BS. This means that in terms of communication cost, if each node of the sensor network has *d* neighbors (*d* is the average degree[2] of nodes in the sensor network) and there are N nodes in the sensor network, then the cost to set up all pair-wise keys between all neighboring nodes will be $N \cdot d$ communications between the BS and the nodes. This creates a large amount of network traffic which is not desirable in a WSN. An argument that differentiates our approach is that not every node has to check with the BS before it can set up a pair-wise key with a new neighbor, if it can get the key directly from one of its already trusted neighbors. Consider the case in Fig. 1 where, the BS already trusts A, B and C. A and B are allocated a private unique pair-wise symmetric key (K_{AB}) by communicating with the BS, as are A and C (K_{AC}). Instead of requesting the BS to provide a key when B and C want to communicate, they can be allocated their new pair-wise symmetric key by communicating with A, without the need to check with the BS.

The way it work is as follows: Node B wants to communicate with node C. B sends a message to all of its trusted immediate (in the radio range of the requesting node) neighbors asking them if they trust node C. If a trusted neighbor A replies that it does, B then asks A to allocate them a new pair-wise symmetric key K_{BC}, as described above with the direct communication with the BS. If no neighbors reply, B has no option but to ask the BS (perhaps through multiple hops) to allocate the key.

In the present scheme, the cost of communications is improved by a factor of d with N communications with the BS needed instead of $N \cdot d$ as in the previous scheme. Initially, each node needs to establish a pair-wise key with one other neighbor via the BS. Later, a pair of nodes can establish a pair-wise key via any trusted neighbor that they have in common. Moreover, considering the physical distribution of nodes, most nodes are far away from the BS. Rather than having the majority of nodes using multi-hop communications to establish keys with each of their immediate neighbors, they only once need to use long distance multi-hop communication to set up their first local trust relationship and then all further trust relationships are established with single-hop communications (immediate neighbor).

[1] A token is a packet of information provided by a node that can show evidences of authority, validity or identity.

[2] The number of overall immediate neighbors of a node is called its degree *d*.

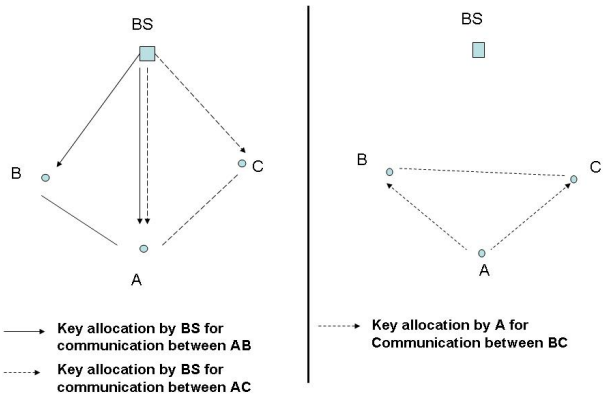

Fig. 1. BS-based key-distribution vs. Neighbor-based key distribution

2) Protocol description

To calculate the initial trust, suppose that we have 3 sensor nodes A, B and C. Node A is the allocator of the key and has trust values $TN_{A\rightarrow B}$ and $TN_{A\rightarrow C}$. B has a trust value $TN_{B\rightarrow A}$. C has a trust value $TN_{C\rightarrow A}$. B can calculate the new trust value $TN_{B\rightarrow C}$ using $TN_{B\rightarrow A}$ and $TN_{A\rightarrow C}$ using some local function. In our approach we simply multiplying the values together, i.e. $TN_{B\rightarrow C}=TN_{B\rightarrow A}.TN_{A\rightarrow C}$. Multiply works well for limiting cases, resulting in $TN_{B\rightarrow C}=0$ if either $TN_{B\rightarrow A}=0$ or $TN_{A\rightarrow C}=0$ and resulting in $TN_{B\rightarrow C}=1$ if $TN_{B\rightarrow A}=TN_{A\rightarrow C}=1$. It also results in sensible initial trust values with other values for $TN_{B\rightarrow A}$ and $TN_{A\rightarrow C}$.

At the beginning of shared-key distribution protocol, each node sends a *Hello* message. Nodes that hear a *Hello* message record the address of the sending node in their Neighbor table. A BS that receives a *Hello* message will immediately send a *Key* message to that node. This *Key* message informs the node that the BS is within radio range. Once a node has a route to a BS, it begins to look through its list of neighbors for any neighbor that it does not yet have a shared pair-wise key. In a standard Kerberos-based [5] system, the node would then ask the BS for a pair-wise key that it can use to securely communicate with that neighbor (the "*target*"). Here instead, the *requester* node will send out an *Allocator Request* message. If any immediate neighbor shares pair-wise keys with both the *requester* and the *target* then that neighbor (the "*allocator*") can reply with an *Allocator Reply* message. If no neighbor replies to the *Allocator Request* message before a timer has expired then the *requester* has no choice other than to send a *Key Request* to its BS (note that each node must have at least one key allocated to it via a BS before it can begin requesting keys with other neighbors). However if an *Allocator Reply* message is received then the *requester* can send a *Key Request* message directly to *allocator*. When a node or BS receives a *Key Request* message, it generates a new random key and sends it to the *requester* in a *Key Reply* message. The *requester* receives and records the new pair-

wise key and creates a *Key* message that it sends to the *target*, including a copy of the token that was in the *Key Reply* message. The Key message is received by the *target* and it can use the token to verify that the new pair-wise key came from the trusted *allocator*.

B. Simulation, results and analysis

The neighbor-based shared-key distribution has been simulated using the OMNet++ simulator [9] in C++ (see Fig. 1). To measure the efficiency of the algorithm, simulations were run to record the total number of packets sent by all nodes. These include *Hello* messages, *Allocator Request* and *Reply messages* (of which there are none when the BS performs all key allocations), *Key Request* and *Key Reply* messages (counted once for each hop from the source to the destination) and *Key* messages.

We compared the neighbor-based shared-key distribution to distribution of pair-wise keys performed only by the BS (see Fig. 1). In our simulations, we had one BS in the center of a square region which contained randomly scattered nodes (see Fig. 2). The size of the area was increased in proportion to the number of nodes to keep a consistent average node density. The experiments were repeated with average node densities of 10 and 15. Each simulation ended when all nodes had acquired pair-wise keys with every one of their radio-range neighbors. The results are the averages of each of the 20 iterations.

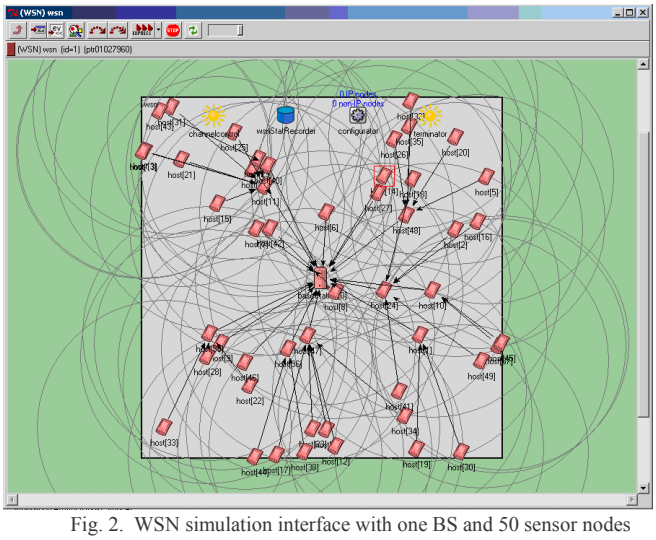

Fig. 2. WSN simulation interface with one BS and 50 sensor nodes

The objective of using the neighbor-based shared-key distribution scheme was to reduce the number of transmissions required while forwarding additional *Key Request* and *Key Reply* messages to and from the BS. The cost of this is the number of transmissions required to find out if an immediate neighbor would be able to perform the key allocation, rather than having to ask a BS. We found (see Fig.3) that when there are a small number of nodes, the overhead of *Allocator Request* and *Allocator Reply* packets exceeded the savings from reduced *Key Request* and *Key Reply* forwarding. Nodes that are only a few hops away from the BS send more transmissions with the proposed scheme than with the simple BS distribution scheme. But the proposed scheme scales well with increasing network size. As the number of nodes increases, the proportion of nodes that are far away from the BS increases. Beyond a certain network size, our method may be able to give a substantial performance increase which is appropriate for many application of WSNs. The threshold and savings depend on the average node density. From these results, we can see that the lower the density, the smaller the network size needs to be, before we begin to see the benefits of using our algorithm. Also, the rate of increase in number of transmissions per node slows down as the number of nodes increases when using our algorithm.

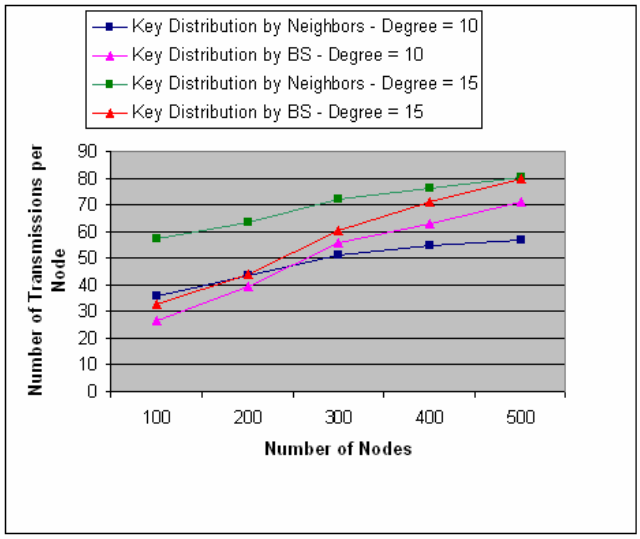

Fig. 3. Number of transmissions/node required for all nodes to share a key with their trusted neighbors.

III. Dynamic Trust-Based Routing Protocol

A. Routing behavior

In a simple WSN, data is routed from the nodes to the BS and global maintenance messages are flooded from a BS to the nodes. When a node sends a request for information to the BS a route must be maintained to allow the reply to be sent back. In more complex situations a node may wish to send a message to a specific node, perhaps for data aggregation.

We expect our algorithm to exhibit the following properties:

- If a route will not succeed (there is some node on the route that has an attributed trust value of 0.0) then nodes will not use that route (trust value is between 0.0 and 1.0).
- The algorithm should avoid loops when a node is selecting a new route.
- The algorithm should minimize the overhead of route and

trust information that nodes are transmitting to maintain trusted routes to the BS.

B. Computing the trust and cost of a route

Node A use Formulas (1) and (2) to computes the trust $TR_{A\to B\to BS}$ and cost $CR_{A\to B\to BS}$ of a route through node B to BS (the destination may also be any node other than the BS) by combining the trust $TN_{A\to B}$ (trust that node A grants to node B) and $CN_{A\to B}$ (cost to transmit directly to B) that it has for node B with the trust $TR_{B\to BS}$ and cost $CR_{B\to BS}$ that node B has broadcast (see Fig. 4).

$$TR_{A\to B\to BS} = \min(TN_{A\to B}, TR_{B\to BS}) \quad (1)$$

$$CR_{A\to B\to BS} = CN_{A\to B} + CR_{B\to BS} \quad (2)$$

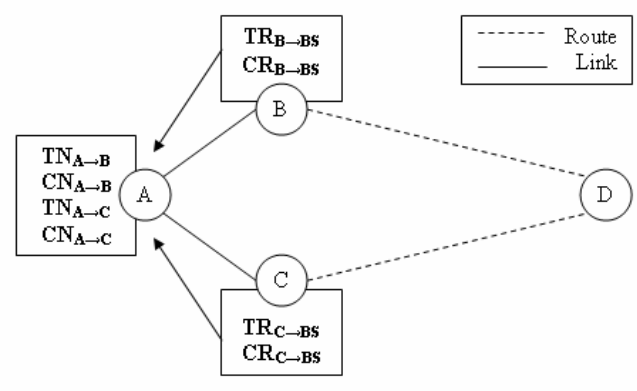

Fig. 4. Trust and cost parameters to compute the trust of routes.

1) Choosing between two routes

A node chooses a route according to the cost of a route as well as its trust of the route. Cost can have any value ≥ 0. For simplicity, the cost for transmission from one node to an immediate neighbor is 1 in our simulations, but any metric may be used, such as latency or the radio power required to reach the neighbor. A node wishes to minimize the cost while maximizing the trust of the route it chooses to use.

Suppose that node A is given the choice between two routes (to the BS or any other node). For simplicity we call these routes R_B and R_C with trust values TR_B and TR_C and costs CR_B and CR_C respectively; there are four cases to consider:

- If TR_B=TR_C then choose the cheaper route.
- If CR_B= CR_C then choose the more trusted route.
- If one route is more trusted and cheaper than the other route then choose that route.
- If $TR_B > TR_C$ and $CR_B > CR_C$ then the node must have some way of deciding if it is more important to find a more reliable route or a cheaper one. The choice should depend on both the cost and the trust.

According to Equation (3), we propose to take the trust and cost of two routes and compare the cost of trying and failing with one route vs. trying and failing with the other. Let E_B be the Expenditure of choosing R_B. Then E_B is calculated as:

$$E_B = CR_B + (1-TR_B).CR_C \quad (3)$$

We assume that $(1-TR_B)$ is proportional to the probability that using route R_B fails.

Similarly,

$$E_C = CR_C + (1-TR_C).CR_B \quad (4)$$

Comparing E_B to E_C, the route with the smaller Expenditure is chosen by A.

We can test the formula and see that we get the same result as earlier:

- If TR_B=TR_C then the cheapest will always have a smaller Expenditure.
- If CR_B=CR_C then the most trusted will always have a smaller Expenditure.
- If one route is more trusted and cheaper than the other route then the Expenditure will always be smaller for this route.

2) Choosing between more than two routes

We can compare all "pairs" of available routes using the method listed above. There is a potential problem: what if node A is given the choice between three routes R_B, R_C and R_D where $E_B < E_C$, $E_C < E_D$ and $E_D < E_B$? In this situation there is no overall best route. We have tested this empirically and found that for all possible values of TR_B, TR_C, TR_D, CR_B, CR_C and CR_D there is always one route that has a smaller Expenditure when compared with the other two routes, except in trivial cases such as when two or more routes have the same TR and CR or when $CR_B = CR_C = CR_D = 0$.

In order to compare all the routes to find the best one, a node doesn't need to compare all pairs. It only needs to compare two routes and discard the route with the higher Expenditure until it has eliminated all but one route, which is its best choice.

3) Avoiding loops

Our method of selecting a route to use does not guarantee that loops do not occur. When node A changes the trust or cost for a neighbor or route, it would immediately choose a new best route from all those it has available. But the best available route may contain node A. Such situations occur because the change in trust or cost needs to be transferred to node A's neighbors, which would then be aggregated into their route trust and cost values and then transmitted back. Even then there is a potential for loops to occur due to

synchronization. To prevent these problems, we had the node send a *Route Test* message before using or broadcasting an updated route. The *Route Test* message is sent to the destination using the route to be tested. If a node or BS receives a *Route Test* message then it replies with a *Route Confirm* message. If a *Route Confirm* message is received then the node can start using that route and transmit a *Route Update* message to all neighbors. If a *Route Test* message comes back to the node that it originated from then the node knows that using that route will cause a loop, so it excludes that route and chooses a new best route from those it has available.

Fig. 5 show the simulation interface of the dynamic trust-based routing with four BS and 50 nodes tracing their route to their respective BS.

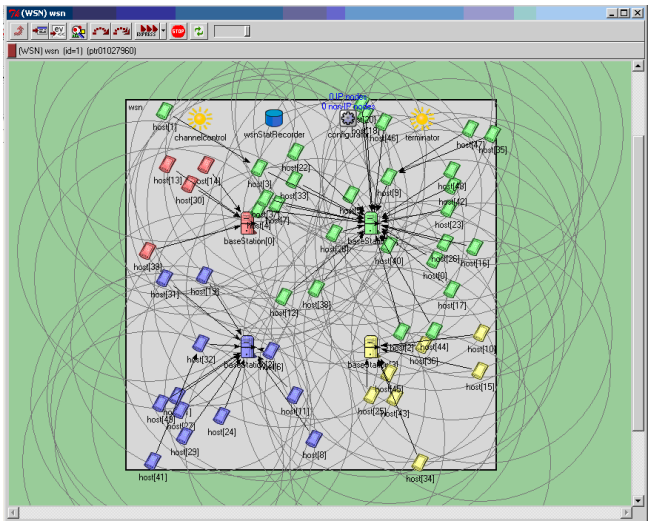

Fig. 5. Dynamic trust-based routing with four BSs.

C. *Simulation, results and analysis*

We implemented our algorithm and ran a number of simulations. We occasionally observed short (two-hop) loops or routes that were obviously not optimal (e.g., node A routes through B which then routes through C, when node A could have routed directly through node C) but these were caused by routing messages failing to reach nodes (due to collisions) and we found that the network rapidly returned to a loop-free and efficient state.

Table.1 and Table.2 show the average number of transmissions per node to perform routing updates in two situations. The first simulations are with no mechanism for change in the trust and cost values of nodes, thus routing updates only occur when shorter routes are detected during the initial key setup. In the second test, each time a node sent a certain number of messages (500), that node increased the cost for other nodes to route through it to the BS by sending a *Route Update* message with an increased cost (CR increased by 1). Each simulation (topology with 100, 200 and 300 nodes) has been run 20 times and the results have been averaged over these 20 iterations.

TABLE I
AVERAGE NUMBER OF MESSAGES PER NODE
WITH DIFFERENT SIZE OF WSN (100, 200, 300) AND NO CHANGE IN COST AND TRUST

Type of Messages	100	200	300
CreateRoute TestMessage	1.92	2.01	1.89
ForwardRoute TestMessage	3.67	5.9	7.42
ForwardRoute ConfirmMessage	3.64	5.83	7.29
CreateRoute UpdateMessage	0	0.01	0.04

TABLE II
AVERAGE NUMBER OF MESSAGES PER NODE
WITH DIFFERENT SIZE OF WSN (100, 200, 300) WITH COST ADJUSTED

Type of Messages	100	200	300
CreateRoute TestMessage	2.91	3.82	13.26
ForwardRoute TestMessage	5.72	11.69	64.72
ForwardRoute ConfirmMessage	5.15	10.31	43.06
CreateRoute UpdateMessage	0.25	0.69	3.62

IV. MAINTAINING TRUST

This section explains other mechanisms that have been added to the trust-based routing protocol to adjust the trust based on the following criteria:

- The the age of the trust values that a node has for each neighbor (TN).
- The direct behavior of a neighbor node.
- The reputation of a node provided by neighbor nodes.

A. *Decay of trust*

If a node does not receive any new information from a neighbor for a significant period of time (such as a missing *ack* packet acknowledging that a packet has been received), it should become less confident in its estimate of the trust value granted to this neighbor with increasing time: high trust values should decrease, and low trust values should increase (gradually correcting a mistake in labeling benign node as malicious or acting in an inappropriate way). For this reasons, trust in a given node (named *T* here instead if *TN* for simplicity) slowly decays exponentially towards an intermediate default value according to Equation (5),

$$T = T_0 + (\alpha - T_0) \cdot \left(1 - e^{\frac{-t}{\tau_\alpha}}\right) \quad (5)$$

where T is a trust value at a given time t, T_0 is the initial trust value, α is the default trust value (α could chosen based on different characteristics and configurations of the WSN such as the type of applications, the type of sensor nodes, the topology of covered areas, etc., and τ_α is the time constant for the decay by 0.63 (i.e., $1–e^{-1}$) towards the default trust value.

B. Reward and punishment

A good behavior is defined as a behavior that improves the efficiency and integrity of the network, while a bad behavior is the opposite. An example of good behavior is when a node is forwarding consistent packets of data received from neighbors to other nodes. An example of bad behavior is when a node is failing to forward packets of data (either through its own fault or the fault of other nodes further down the route), indicating a malfunction or potentially an attack.

Good or bad behavior add or subtract certain amounts of trust depending upon the expected frequency and severity of the events according to Equation (6). In particular, we should try to arrange so that:

$$p_{bad|good} \cdot \Delta T_{bad} = p_{good|good} \cdot \Delta T_{good} \quad (6)$$

where $p_{bad|good}$ is the probability that a node is deemed to have misbehaved when it was really being good (false alarm rate for this node), $p_{good|good}$ is the probability that a node is deemed to have been good when being good; ($p_{bad|good}$ and $p_{good|good}$ sum to 1), and ΔT_{bad} is the amount that trust is reduced when bad behavior is suspected (punishment), and ΔT_{good} is the increase in trust if good behavior is suspected (reward). One can see that if Equation (6) holds, and there is no attacker, then trust will maintain a constant level around the default value α, on average. Therefore, when choosing our parameters, we suggest a larger reward (ΔT_{good}) than required to maintain the intermediate trust value of α, and $p_{bad|good}$ should be quite small (this is consistent with what we expect in reality), so that ΔT_{bad} is much greater than ΔT_{good}. Consequently, trust will slowly climb to a relatively high value through the course of normal interactions between nodes. If bad behavior from a real malicious node is detected, then trust granted by its neighbors will quickly fall to zero.

V. Discussion and Future Work

A. The neighbor-based shared-key distribution

One security issue about our scheme it that a node that allocates keys will have access to that key and may use it maliciously. Our scheme makes the assumption that if a node is trusted it should not act maliciously by storing keys that it has distributed, but it will immediately forget them.

Obviously, if a node A is compromised, any pair-wise key it allocates from that point on will be compromised. However, this effect can be minimized by each node reducing the trust of every node that it witnesses allocating a key.

The sensor nodes can also negotiate pair-wise keys with malicious nodes that may be a long way out of the normal radio range supported by the senor node, but again we may minimize this effect by reducing the trust of nodes who set up keys.

There is also the possibility that an attacker can record some conversations then later compromise a node. For instance, suppose that A allocates the key K_{BC}. A immediately "forgets" the key K_{BC} (A is currently trustworthy and so erases K_{BC} from memory immediately after transmitting it to B and C). Later, if A is compromised, an attacker may be able to use the keys K_{AB} and K_{AC} to decode the messages that A sent when allocating K_{BC}, thus being able to read any message transmitted using the link BC, including any key allocation messages that are sent via BC.

This highlights an important fact. If we are not using public key encryption then we must assume that any message sent across the network may be readable by an attacker, except those sent between the nodes and BS using keys that were pre-distributed and those between nodes where the BS allocated the keys. This effect can be minimized by revoking keys allocated by un-trusted nodes. But as it has been mentioned at the beginning this idea of nodes allocating keys should only be used in sensor networks where it is less important to keep historical data secret and more important that current communication is reliable: that is detecting possible intruders or weak nodes under attack.

Further improvements of the method may include:

- node B asks all of the nodes it trusts (even those at multiple hop range) if any of them trust C. However, in a WSN, it is sufficient for nodes to securely communicate with their immediate neighbors and the BS, so the list of immediate neighbors is probably similar to the list of all trusted nodes. A common trusted node is likely to be within radio range of both nodes anyway. Also, using multi-hop communication incurs additional resource cost that is undesirable.
- B could ask A for its list of trusted nodes instead of only asking A if it trusts C. This is not secure because an attacker could then work out who in the WSN trusts who and can use that information to her advantage.

- If we choose to use only single-hop communication and B is unable to find a trusted immediate neighbor who trusts C, then B could delegate the task of finding a common trusted neighbor to C. C would then use the same process and ask its immediate neighbors if one of them trusts B. If B asks all trusted nodes if they trust C and finds none that do, then there is no need for C to do the same because it should come to the same conclusion (if A tells C that it trusts B then A should have told B that it trusts C).

We must also consider the case of an attacker compromising nodes and trying to impersonate as many identities as it compromises. By recording all keys it distributes it can read all communications that use those keys. It may also impersonate the nodes at either end of the links it has established[3], but only to communicate along the links it established.

An attacker could try to set up pair-wise keys with every node in the network. Then whenever a node tells its neighbors that it wishes to establish a new pair-wise connection, the attacker could volunteer to assign the key. For this reason, each node should only check with its immediate neighbors when trying to establish connections and not ask multi-hop neighbors. This will minimize the effect that an attacker could have in this situation.

B. The dynamic trust-based routing protocol

In our WSN protocol, nodes immediately send an acknowledgement packet (*ack*) after they receive a packet of data. If C wants to send data to BS via B then it transmits and waits for the *ack* from B. If it gets the *ack* then it can increase its trust of B. If it doesn't get the *ack* then it could C temporarily set its trust of B to 0. When it hears from B it will return its trust of B back to the previous value minus some amount based on how long B was absent according to the decay of trust Equation (5).

Another thing to consider is that a very cheap and unreliable route may actually have a lower Expenditure than a more reliable, costly one. First instinct is to discard a very unreliable route, perhaps by using a minimum trust threshold. But if we choose to use the cheap route and it fails, we haven't lost much (low cost) and we will punish the route further so that next time its Expenditure will be higher. If it succeeds then we have just successfully found a more efficient route and we will have increased our trust of that route in the process (assuming we increase the trust of a route whenever it is used successfully). So there is no reason to have any type of threshold for the routes that we are considering using.

[3] We say that a node A has established a link between a node B and C if A distributed the key for the secure communication between B and C.

REFERENCES

[1] K. Römer and F. Mattern, The Design Space of Wireless Sensor Networks, in IEEE Wireless Communications 11 (6): 54-61, December 2004.

[2] H. Chan and A. Perrig, *Security and Privacy in Sensor Networks*, In IEEE Computer, 36(10), October 2003, pp103-105.

[3] S. A. Camtepe and B. Yener, *Key Distribution Mechanisms for Wireless Sensor Networks: a Survey*, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York, USA, TR-05-07, 2005.

[4] W. Du, J. Deng, Y. S Han, S. Chen, and P. K. Varshney, *A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge*, in IEEE INFOCOM, 2004.

[5] B. C. Neuman and T. Tso, *Kerberos: An authentication service for computer networks*, in EEE Communications, vol. 32, no. 9, pp. 33–38,September 1994.

[6] G. Gaubatz, J. P. Kaps and B. Sunar, *Public Key Cryptography in Sensor Networks - Revisited,* In Proceedings of ESAS'04, C. Castelluccia and H. Hartenstein (Eds.), LNCS 3313, pp. 2-19, Springer, Heidelberg, 2004.

[7] H. Zhu, B. Feng, and R. H. Deng, *Computing of Trust in Distributed Networks*, Cryptology ePrint Archive: Report 2003/056, Available at: http://eprint.iacr.org, 2003.

[8] D. Artz and Y. Gil, *A Survey of Trust in Computer Science and the Semantic Web,* Information Sciences Institute, USC, USA, March 15, 2007.

[9] OMNeT++: http://www.omnetpp.org/