Abstract

Wireless sensor networks are rapidly becoming a ubiquitous technology, particularly in relation to the monitoring and control of critical infrastructure such as public transport networks, fuel distribution systems, power generation facilities, and areas of protected ecological development. In light of this, it is a matter of deep concern that there remain a number of serious unaddressed, or at least unsolved, security issues within the field.

Traditional security measures in wireless sensor networks rely heavily on cryptography, and a number of innovative and effective advances have been made in this field. Nonetheless, wireless sensor nodes are by their nature very computationally limited, and almost all cryptographic operations which are tractable on these nodes are trivially breakable on laptop-class hardware. In response to this, a large quantity of recent research has emerged in the field of remote attestation, secure remote reprogramming, misbehaviour detection, and intrusion mitigation and analysis.

This paper presents a proof-of-concept implementation of a novel, configurable solution for a security layer which can enforce faithful forwarding, cooperative misbehaviour detection, asymmetrical link control, and reliable delivery of individual packets – not something which has typically been the focal point of wireless sensor security applications. In addition to this, a novel, truly homogenous monitoring and reporting topology is proposed as an exploration of the extent to which node profusion and density may be used to identify and defend against attacks. The system is intended as a step towards a technology which would allow reliable multihop attestation and reprogramming of nodes in a homogenous mesh network.

Table of Contents

Introduction

Motivation

Security within a wireless sensor network (WSN) environment poses a unique challenge. Due to the scale of the systems in question, which can be in the range of millions of sensors [1], in combination with the vulnerability and often disposability of the individual nodes, sensor hardware is extremely constrained in terms of cost, and therefore resources. The Berkeley MICAz mote is very widely used as an example and a test platform in the available literature [2]–[11], and has the following hardware specification [8]:

ATmega128 microcontroller with:
- 16MHz 8-bit processor
- 512Kb Data memory (flash)
- 128Kb Program memory (flash)
- 4Kb SRAM
- 4Kb EEPROM

2.4GHz 802.15.4 radio
- 100m maximum range (outdoor)
- 30m maximum range (indoor)

2 x AA batteries

Figure 1: A Berkeley MICAz mote

It is worth discussing the standard memory architecture of these devices, at this stage. As WSNs are almost invariably constructed of devices which are designed to be low both in cost and in computational capacity, it is usual to use low-cost microprocessors, such as the ATMega128L, which frequently have a Harvard architecture, due to the reduced cost [12]. There is an additional advantage to this, which is that the separation of the memory into separate regions for data and instructions makes it harder for an adversary to utilise buffer-overrun attacks to execute arbitrary code, although this does remain possible. This is much more of a concern in Von Neumann architecture devices, as it makes self-propagating worms, which are trivially capable of taking control of an entire network, much easier to implement [13].

WSNs are, by their nature, especially vulnerable to a number of varieties of attack. The limited nature of the hardware means that a large number of straightforward security solutions are intractable, or introduce an unacceptable overhead [14]. The large number of sensors in a network means that the authority overseeing the network cannot realistically equip them with powerful computational abilities, whereas an adversary can easily introduce to the network a small number of more powerful nodes, or even a laptop computer, on which many of the encryption solutions available to the motes can be trivially broken [9]. Additionally, while traditional networks can rely to some extent on infrastructure security and denial of access, wireless sensor networks are trivial to gain access to (a perfect example of this would be a crowd monitoring network, the utility of which would be severely compromised if the public were to be denied access). In light of this, it is necessary to design WSN security measures with the expectation that nodes will be captured, moved, modified, and compromised [15], and to invest a great deal of effort into the development

and analysis of methods for the initiation, propagation, and maintenance of trust [10]. This unique combination of challenges and requirements has created an extremely vigorous and fertile area of research.

In spite of these concerns, these devices are routinely used for critical operations where security is a primary concern, and lives are frequently at stake [16]. Typical applications for this type of network can be found in medicine (patient monitoring, drug administration monitoring), industry (monitoring the health of structures and machines), transport (traffic density tracking, accident detection), environmental monitoring (forest fire and flood detection, biocomplexity monitoring) and the military (command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting (C4ISRT) systems) [17]. This great discrepancy between the levels of computational capability and functional criticality of WSNs imbues this area of research with a certain urgency, especially as their adoption is becoming rapidly more widespread, with some market research estimating up to a billion devices in deployment by 2017 [18]. To quote Hu et al:

> "The hard-learned lesson from the PC industry is that ignoring security at the outset leads to huge pain when the technology becomes ubiquitous." [9]

The initial aim of this project was to address the assumption of adversarial silence (detailed below under the review of attestation literature) by having each node listen during attestation time for any kind of unauthorised traffic, and report back to the base station. This turned out to be a plausible project, and would have been implementable on the existing communication protocols. However, when this was examined more closely, it became clear that it would not add any kind of provable security. It would be possible for malicious nodes to drop or modify the reports about attestation time traffic and the attack could go undetected. In light of this, a study was made of various peer monitoring and intrusion detection systems.

This paper presents the claim that, by combining a number of existing technologies in innovative ways and performing a reasonable amount of original research and development, it would be possible to construct at least a proof-of-concept implementation of a holistic distributed security system for WSN monitoring – a route-centric, rather than exchange-centric, anomaly and misbehaviour detection system which makes it possible to detect not only when packets are dropped, but when they are modified, mis-routed, repeated, worm-holed, fabricated, and when nodes are impersonated, as well as being able to provide detailed information to the central monitoring authority regarding topology changes and broader-spectrum issues which affect larger areas of the network.

Objectives

The main objectives of this research project were

1. To perform an in-depth analysis of the attack vectors and vulnerabilities pertaining to multihop communication in a wireless sensor network.

2. To formulate a set of security features which will neutralise as many of the identified attack profiles as possible.

3. To develop a proof-of-concept implementation of a distributed, cooperative system which implements as many of the above features as is practicable in the time-frame

Contributions

In addition to the proposed analysis and specification, a system has been developed whereby collector nodes within a homogenous mesh network may be instructed to monitor and report upon the behaviour of the nodes within their reception radius. This system is designed to be modular and highly configurable, and implements the following security measures.

- Tamper-proofing. The payload of a packet is inspected by its peers at each routing step, and any mutation or corruption of that payload will trigger responsive action.

- Signature-based localisation. The signal strength of each neighbouring node is known, and compared to the strength of each received or overheard packet. Anomalies are reported, and differential link strength variations may be used at the base station to diagnose geographical transmission inconsistencies.

- Link control. During deployment, each node gains a knowledge of the inter-node communication links possible both at itself and at each of its neighbours. Communications from nodes which were not present at this time are reported, and must be approved before the messages can be accepted.

- Faithful forwarding enforcement. Each node attempts, through the application of internal routing and behavioural heuristics, to analyse the routing choices of the nodes around it. If a node fails to forward a packet, or forwards a corrupted packet, or mis-routes a packet, the incident will be reported.

Background and related work

In his layer-based classification [19], Saxena seems to base his analysis of the different areas of research on the OSI model, and splits the areas of research into a number of broad categories which include key management mechanisms, physical layer security, MAC layer security, network layer security, high-level security mechanisms, and DoS attacks. In addition to these, secure routing mechanisms are deemed an important enough subtopic that they are granted a full section of their own. Buttyàn, who is a member of the WSAN4CIP[1] initiative, takes a more functional view of the available countermeasures. In his article [16], the first addressed area is sensor node architecture, which covers the TinyIDS intrusion detection system (naturally enough, a WSAN4CIP project), operating systems, and code attestation methods. The author does not feel that the current remote attestation mechanisms are sufficient, being vulnerable to return oriented programming, and makes reference to a longer article detailing the issues with these [6]. This article, and its later refutation, will be covered later, in the section on attestation. The article then goes on to cover dependable networking protocols, including node deployment, where the assumption of random node distribution is questioned, and traffic analysis protection. It concludes with an analysis of the central role of service dependability, including the importance of graceful degradation under adverse conditions such as a jamming or service denial attack, along with the criticality of network recovery in the wake of such. In particular, this paper shall address the importance of network recovery later, as it is relevant to the field of node attestation and secure remote reprogramming.

Of these topics, not all are relevant to the current investigation, which has developed a strong focus on remote attestation methods and intrusion detection systems. An exploration shall be made into the subject of key management, which is fundamental to the area, including a summary of the work recently performed by Tan et al [2] into the production of a TPM breakout board for performing complex cryptographic calculations, followed by an in-depth look at the current state of the art in the area of remote attestation, with a description of various protocols and a discussion of the current controversy in this area. The section will conclude with a brief overview of a number of distributed, cooperative intrusion detection systems.

Key management

This area is related to the distribution and management of keys within the network. Cryptography within a wireless sensor network is a particular challenge because mote-class hardware modules (wireless sensors, such as the Berkeley MICA2) are invariably impoverished in terms of power and computational ability [14]. Wireless sensor network (WSN) key management protocols can be placed in a number of broad categories, as follows.

Network-wide Keys

In this scheme, a single symmetrical key is used for the encryption of data across the entire network. In the context of a WSN, this scheme is catastrophically insecure, as either a simple brute-force attack by a more computationally powerful opponent, or the capture and compromise of a single node, would result in the attacker gaining access to and potentially control over the entire network.

Master Key and Link Keys

1 Wireless Sensor and Actuator Networks for Critical Infrastructure Protection

In this scheme, each node is initialised with a copy of a master key. Once the node is activated, it uses the master key to obtain a set of link keys corresponding to the nodes with which it will need to communicate. Once this process is complete, the master key is deleted [19]. This idea of a setup or initialisation phase during which the network can be guaranteed to be secure is common to many of the ideas which shall be detailed in this document, as it can be used to perform secret-sharing operations in a computationally inexpensive manner. While having the advantage of being less vulnerable to node capture attacks than a network-wide shared key scheme, this scheme has the flaw that new communication links cannot be initialised once the setup phase is complete, meaning that all potential group reorganisations and node additions must be planned in advance [15]. This issue, however, is partially one of classification. While pure master-and-link systems cannot initiate new links, this is partially because the inability to generate new links is part of the definition of a master-and-link system. It is possible to combine this management scheme with, for example, a bootstrapping key mechanism, as described later, to allow new links to be created after the initialisation phase.

Preconfigured Link Keys

In this scheme, each potential link between two nodes is assigned its own unique symmetric key during the setup phase. This system has the advantage of having an extremely low overhead (with no requirement for key generation beyond rekeying operations during runtime) and high flexibility (as all links between nodes are possible). However, it necessitates a very large number of keys (n(n-1)/2) to be stored in each node, and makes it impossible to add nodes to the network [19].

Bootstrapping Keys

In brief, the bootstrapping key system allows new link keys to be generated on demand within a network. Simpler implementations of this system involve a link key being issued by the base station when requested by a given pair of nodes [19], but this suffers from a number of serious security flaws. Firstly, the base station must keep a database of link keys, and so becomes a single point of failure (although this is also a serious issue in many other security approaches for WSNs), and secondly it allows the reauthorisation of compromised nodes. Both of these issues are addressed by the SAKE and trustedFleck systems, which will be discussed later.

Asymmetric cryptography

This scheme is based around public-key cryptography, which allows secure key establishment and exchange over an insecure channel, through a method based upon the intractability and irreversibility of certain mathematical operations. The first practical implementation of public-key cryptography was developed in 2004, and achieved public key generation within 34 seconds on a MICA2 mote [20]. While this latency is clearly prohibitive for frequent routine communication, the authors suggest that it may be suitable for shared secret distribution, such as changing a symmetrical shared key on a regular basis.

An improvement upon this system was designed and implemented by by Hu et al [2], [9], who developed trustedFleck, a trusted platform module (TPM) breakout board, the purpose of which is to handle cryptographic functions. While this allows the mote to perform RSA-based PPK operations with vastly reduced latency and energy consumption (see table 1), asymmetric cryptography remains prohibitively expensive in terms of energy consumption for routine communication, and is still suggested solely as a method for securing the rekey process for XTEA encryption. It is theorised by the authors of the paper that this form of security enhancement may

not see widespread adoption unless a commodity sensor model becomes available which incorporates the TPM hardware used in the project. As the private keys are stored in the TPM, which are tamper-proofed, the use of the trustedFleck platform can go a long way towards protecting the network against node capture attacks.

Platform	Current (mA)	Time (μs)	Energy (μJ)
RSA (software, $e = 65{,}537$, 2048-bit key)	8.0	219,730	7,030.0
RSA (hardware, $e = 65{,}537$, 2048-bit key)	50.4	27	5.4
XTEA (software, 128-bit key)	8.0	18	0.6

Table 1: trustedFleck (RSA and XTEA) Encryption Energy Consumption for 1 bit of Data [9]

This work is a significant step towards making strong cryptographic operations tractable on mote-class hardware, but still relies upon analytically verifiable protocols for key exchange, software attestation, data storage, and node recovery. While this system represents a large step forward in the field of sensor network security, it does not by itself constitute a complete solution. In an environment where node capture is likely, it is unwise to rely fully on the tamper-resistance of hardware [21], and so further work into procedural and unified solutions is required.

Another interesting development relating to asymmetric cryptography in wireless sensor networks is the SAKE protocol, which allows the secure establishment of keys between nodes with no prior shared secrets, and which does not rely upon the node being uncompromised prior to the establishment process [22]. The security of this method relies on the ICE (indisputable code execution) system, an earlier work from the same authors, and therefore shares the same major vulnerability (an attacker's devices remaining present in the network during the attestation process) [23]. This system, and its associated complications and criticisms, will be discussed in a later section.

Remote Attestation

The area of remote attestation is a deeply complex and extremely interesting one. There is an ongoing debate as to whether indisputable remote software attestation is in fact even possible, which shall be addressed in this section, along with a brief explanation of a number of different protocols.

SWATT

SWATT was one of the first practical implementations of remote software attestation for low-end embedded devices. It produces a checksum of the program memory in the device by processing program words in a pseudo-random order, and returns that checksum to the node which demanded the attestation procedure. The system relies upon very tight timing requirements to prevent on-the-fly manipulation, redirection, or falsification of the checksumming routine [24]. This system has a number of vulnerabilities and limitations, which are detailed in Castelluccia's article “On the difficulty of software-based attestation of embedded devices” [6]. These include the fact that the attestation routine does not check data memory or external memory, the fact that the security mechanism is reliant upon the attacker not being able to optimise the attack routine beyond a certain point, and the fact that SWATT is implemented only for the older ATMega163L microcontroller, which is present only in older Berkeley Mica motes. Castelluccia et al construct and present a number of attacks on and criticisms of this protocol, among others, which have since been refuted [7], and then reasserted in the face of that refutation [25]. This area of research is the subject of ongoing debate, and even controversy, which has made the process of gaining a cohesive overview of the state of the art difficult. This controversy, and the ensuing difficulties, shall be further detailed at the end of this section.

SCUBA

SCUBA is another remote attestation method, which provides stronger guarantees than SWATT. While SWATT only guarantees the correctness of the program memory contents, SCUBA provides guarantees regarding untampered code execution [23]. These guarantees are provided contingent upon the usage and correctness of the ICE (indisputable code execution) set of primitives, also developed by Seshadri et al, the operation of which takes place in the following three stages:

1. Check the integrity of the executable.

   In brief, the ICE verification function checksums itself in such a way that any alteration of the execution of the program will result either in an increased runtime or an incorrect checksum. The soundness of this stage is based upon the assumptions that the checksumming routine is non-optimisable, and that any alteration of the routine cannot be performed quickly enough to fall within the stringent timing benchmarks. There is some debate as to the soundness of these assumptions [6], [7], [25].

2. Set up an execution environment in which the execution of the executable is guaranteed to be atomic.

   Once the executable has been verified, all maskable interrupts are disabled, and dummy handlers are instated for non-maskable interrupts. In this way, no program is able to take control of the hardware until such a time as the ICE-enabled process voluntarily

relinquishes control.

3. Invoke the executable to execute in the untampered execution environment.

The ICE system is designed to be carried out in an “expanding ring” sequence, as the protocol only works over a single hop. What this means is that the initiating base station can verify the nodes within one hop of it, after which each of these nodes goes on to verify all the nodes within its range, and so on, until all the nodes have been verified. With the addition of localisation and mapping, a particular node can be verified in a much more efficient way, simply by verifying a line of units between the base station and the node in question. The authors of this system decline to offer a solution for situations where the base station has been compromised, on the basis that this situation represents the compromise of the entire network – a situation from which recovery cannot be reasonably expected.

One major concern regarding the ICE-based methods is that they are vulnerable to attacks where the adversary maintains computationally-powerful hardware within the network. In the proxy attack, detailed in the original paper, it is possible for a compromised node to hand off computation of the checksum to a nearby device of this sort, and use the reduced processing time to hide the lost time due to transfer. This is clearly a concern, and a situation to which a number of solutions present themselves, prominent among which is the idea of suppressing and monitoring network activity during the attestation process. This idea will be explored further in a later section.

The particular implementations presented in the original paper are vulnerable to ROP (return orientated programming). This is a technique where existing blocks of code (referred to as 'gadgets') are chained together to perform arbitrary computation. Proof-of-concept implementations of this exist where parts of the TinyOS bootloader are chained together to place a hook in the attestation routine which, in the event of an incoming attestation request, will delete all of the operational malware from the memory. It then allows the attestation routine to complete, having modified the return address on the stack to point to the malware installation routine, hidden in a part of memory which is not checked by the attestation routine (in this case, the EEPROM). Return oriented programming is a serious concern in remote attestation, as it is one of the major methods by which attestation routines may be foiled. Once again, there is considerable debate as to whether this is an inevitable difficulty in the field, but it does not appear to be disputed that it is currently an issue, and that preventing it constitutes a serious challenge.

Perhaps of more concern are the axiomatic difficulties with this paradigm, which have been raised by Castelluccia et al. The main thrust of the criticism of this scheme is that, in essence, it is very difficult to create an indisputable security system which is based upon timing requirements, particularly in a distributed environment, as this depends upon the code being maximally optimised, a characteristic which is mathematically non-provable. In short, the unreliability of network links and transmission media require a certain degree of leeway in the timing requirements, these timing requirements are necessarily based upon the assumption that the original developers have produced a finished product which cannot be further optimised, and history has shown the malicious programming ('black hat') community to consistently excel in the arenas of code runtime optimisation and the exploitation of small error tolerance margins [26], [27].

PIV

PIV (program integrity verification) is another method of remote attestation, comparable to those above, but with a differing provenance and methodology. Like the above programs, PIV relies upon a checksum of the contents of the sensor module's memory, although it processes the code blocks sequentially, and processes all of the memory locations in the device.

Like the above methods, PIV is vulnerable to ROP (return orientated programming), and especially to hardware modification (in that the addition of extra memory would give malware a place to conceal itself during the attestation process). The attestation routine, however, does not implement a verified and uninterruptable operation environment. This leaves the possibility of dynamic attacks, where the checksum for an uncompromised environment is falsified on the fly in a compromised environment. In order to prevent caching of precomputed responses to challenges, PIV utilises a variety of keyed hash functions, and uses these in combination with a unique secret and the memory contents in order to produce each checksum.

DataGuard

All of the above attestation routines share a common issue: they rely on static code analysis. This is a method wherein the contents of the device's memory are analysed at a single point in time. While the authors of the SCUBA protocol add a time component to the attestation procedure, it is applied only to the procedure which is currently taking place – there is no attempt to address the *history* of the data which has been stored in the memory. The DataGuard project addresses this admirably [28]. The fundamental principle of operation is this: at the time the program is transferred into memory, nonrecoverable data sequences are placed at the boundaries between areas of memory. In the case of any buffer-overrun type operation, at any place in the memory, the data guard will be overwritten. Once this has occurred, the device becomes permanently unable to return the correct response to an attestation challenge.

This fixes one of the fundamental issues with static data analysis, which is the TOUTOA (time of use time of attestation) gap. However tight the timing thresholds get, or how carefully and thoroughly the memory is scanned, techniques which depend solely upon static analysis will always have a blind spot when it comes to the history of program execution on a device. This work is a generalisation of the Stackguard implementation, which uses similar data artefacts (known as “canary words”) to detect return addresses in the stack which have been overwritten by means of an overflow [29]. This method obviously has the limitation that it cannot detect data corruption which does not result in a data field being overflowed, such as exploitation of a *format* in *printf*-type functions. In addition to this, it disallows certain kinds of pointer-type access, and cannot offer fine-grained protection of array elements. Nonetheless, as a method to be used in conjunction with an advanced static analysis technique, it greatly reduces the choice of attack vectors available to an adversary, rendering a large number of wireless-based attacks non-useful. The most obvious remaining attacks require physical access to the remote, such as JTAG access (which can be manually disabled prior to deployment, although it frequently is not [6]).

Intrusion detection systems

The area of study concerned with the creation and improvement of intrusion detection systems (IDSs) is an unavoidably complex and varied one, especially where it is concerned with wireless sensor network monitoring. Intrusion detection systems, at the most basic level, detect misbehaviour within a network by comparing network activity against some definition of correct behaviour. A formal categorisation of this field can be seen in Figure 2.

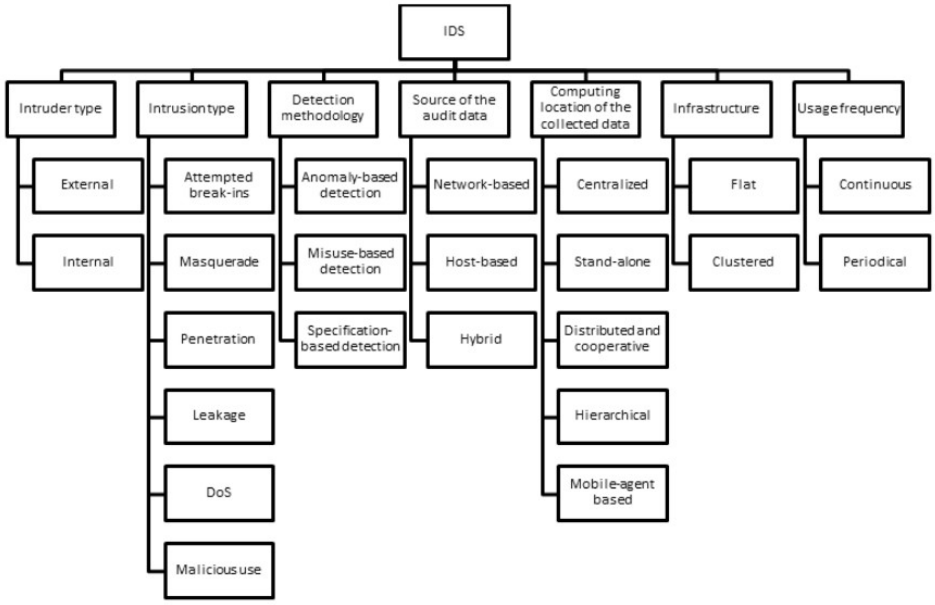


Figure 2: Classification of Intrusion Detection Systems (IDSs) [31]

Of these categorical differentiations, the most obviously pertinent to the field of WSN security is that of network topology. In a clustered topology, all traffic can be guaranteed to pass through a defined subset of the nodes (the cluster heads or routing nodes). This means all behaviour can be monitored from these nodes, a situation which is relatively similar to traditional IDSs, such as might be found in an enterprise network, where all network traffic can be monitored from a router or server within the network [30]. In a flat (or mesh) network, however, traffic is routed between homogenous nodes in a way which may be more or less predictable dependent upon the routing algorithm which is being utilised, and so these individual nodes must be relied upon to either make judgements upon the behaviour of the nodes around them, or to report that behaviour back to some central point for analysis.

There follows a review of a selection of IDSs which are either currently available or proposed for usage within WSNs, expressed largely in terms of Butun's classification of these systems [31], above. This study focuses on systems designed specifically for WSNs, rather than the comparable but distinct field of mobile ad-hoc networks (MANETs), as the additional dimension of node mobility creates a fundamentally distinct, and often incompatible, set of basic design requirements [32].

Two Level Clustering

In their 2010 paper [33], Shin et al propose a method of intrusion detection based upon the two-level hierarchical clustering of nodes into monitors and cluster heads, both of which are under the control of a gateway node, which acts as a routing node. This innovation is suggested as an approach to solve the issue whereby, in single tier hierarchical networks (to wit, networks with

monitor nodes which report directly to gateway nodes), nodes which are more than a single hop away from the gateway node cannot be directly monitored. With this in mind, the infrastructure suggested by the authors is strongly limited to monitor nodes being a single hop away from their cluster head, and cluster heads which are a single hop away from their gateway nodes. This paper introduces a number of interesting improvements upon traditional IDSs, including the idea of distributed intrusion detection and mitigation (attacks are detected and reacted to primarily at the individual cluster head level), and the use of rule- or specification-based misuse detection at the nodes, which is very economical in terms of energy and computational power, and easily configured by the end-administrator to suit a particular deployment.

The stated aim of this research is primarily to improve the monitoring systems for wireless industrial sensor networks (WISNs), which is a scenario very well suited to the topological limitations of the design. In an industrial setting, the monitored environment is likely to be:

- Limited in size
- Relatively static, topologically
- Environmentally consistent, if not hospitable
- Access-controlled
- Populated by personnel who, by and large, are cooperative with being monitored

The limited range, relatively inflexible topology, and comparatively high requisite density of gateway nodes makes this configuration unsuitable for environments which are:

- Geographically extensive, e.g. oil pipelines
- Prone to rapid and violent change, e.g. battlefields
- Mobile, e.g. transport networks
- Lacking in infrastructure, e.g. disaster zones

This model is focused on rule- or specification- based intrusion detection, and all information collation and analysis is done at the cluster heads. This, of course, introduces a single point of failure, as the subversion or destruction of a cluster head can compromise or throw into disarray a large part of the network, respectively. This means it would be risky to deploy a system of this kind in an area with uncontrolled access, such as an exterior urban setting, or a controlled setting with a potentially uncooperative population, such as a prison or a hospital.

Cellular organisation

Wang et al have proposed what they describe as a lightweight defence scheme [34], and which combines many of the features of a distributed intrusion defence system and an ad-hoc routing algorithm. During the initialisation phase, the nodes gain information about their own absolute location via GPS, and share this information with their neighbours. They then organise themselves into regular hexagonal cells, as illustrated in Figure 3, and a single node within each cell remains awake while all others sleep until such a time as they are required to take on the duties of the active cell node.

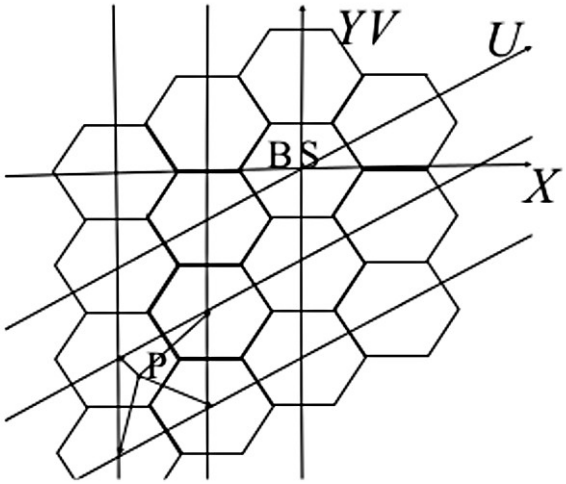


Figure 3: Determining the hex cell to which node P belongs [35]

In this system, packets are routed between discrete cells, and are accompanied by 'event packets', which notify the surrounding nodes of packet origination, forwarding, and acknowledgements, and instruct the other nodes within the current and surrounding cells to monitor the packet exchanges to watch for any misbehaviour (the focus of the study is selective forwarding attacks). In the event of a detected instance of malicious behaviour, one of the monitor nodes will reroute the packet to avoid the suspicious node, and action will be taken to rehabilitate or excise the offender.

This system of operation is ground-breaking in a number of ways – in particular, the use of monitor nodes as buffering and routing nodes is intriguing, as it allows for the density of the network to be used against the adversary. In many distributed security systems, the isolation of the individual devices is treated as a weakness to be compensated for, whereas this system actively leverages it to drastically increase the difficulty of an undetected subversion within the network. Concerns have been raised, however, regarding the rigidity and predictability of the network structure, and the limited numbers of watchdog nodes in use at any one time [31]: with the hex cells and monitoring processes so closely defined, it might be possible for an adversary to gain control of a number of complete adjacent cells, a scenario which would be very difficult to detect or repair. This concern leads unavoidably to the concept of a system where the same or similar measures are implemented within an amorphous or topologically flexible mesh network with ad-hoc routing strategies.

Design

Problem Specification

The initial intuition for the project was to address the assumption of adversarial silence within the field of remote attestation. The obvious solution to this, as suggested in the literature [23], [35], was to implement a method of ensuring that there is silence across the network during attestation time, and to report any on- or off-network traffic taking place during this period to the base station or a cluster head, where the traffic could be analysed; any systematic overt or covert communication with the prover (node under attestation) could be identified; and the problem could be addressed (either by remotely reprogramming or excising the compromised node, or by triangulating the position of the malicious device and dispatching a mobile agent to retrieve, deactivate, or subvert it.) This naturally led to an analysis of the possible behaviours of any compromised nodes (either under attestation or not) during this process. For the sake of simplicity, we are assuming that the entire network is set to monitor network traffic during attestation time. Take, for example, the following layout:

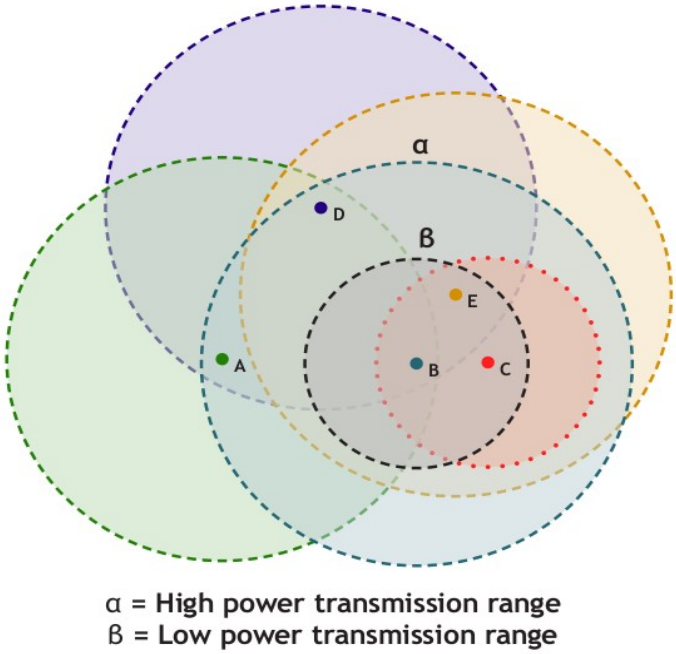


Figure 4: A sample mesh network topology

In the above diagram, node A is attesting node B, which is compromised, and being assisted by a malicious device C. Nodes D and E are acting as 'watchdogs' [36]. All nodes have a maximum broadcast and reception radius of α, but are capable of broadcasting with reduced power at a radius of β. The transmission radius of each node is illustrated as a dotted line in the same colour as the node. The sequence of events is as follows.

1. A issues an attestation request to B, broadcasting with maximum power.

2. B acknowledges the request, broadcasting at maximum power. This broadcast is received by A, D, E, and C, and initiates 'attestation time'.

3. D and E begin logging traffic.

4. B forwards the attestation request to C, using reduced power. E overhears this request.

5. C solves the request, and delivers the answer to B, using reduced power. E overhears this request.

6. B forwards the answer to A, using full power.

7. A signals the end of attestation time, causing all watchdog nodes to deliver their reports.

8. D has nothing unusual to report. E has a suspicious exchange to report, so it formulates an activity digest, and transmits it to the verifier, A. As there is no single-hop connection between A and E, the report must be forwarded. As node B is currently under suspicion, the report must be forwarded through node D.

9. D forwards the report to A, resulting in the conclusion that the attestation result is invalid.

10. Loss of trust in B is established, and action is taken, in line with the suggestions in the previous paragraph.

This protocol is reliable under the assumption that there is only one compromised node in the network at any one time, but it is unacceptably vulnerable if that assumption turns out to be unwarranted. If, for example, node D is compromised, it can either modify or fail to forward the report from E, dependent upon what would constitute faithful behaviour under the utilised protocol. In addition to this, a compromised node E could fail to report suspicious traffic. The latter of these problems could be solved, or at least mitigated, by increasing the network density, and thereby the number of nodes which would need to be compromised in order to allow such an exchange to take place undetected, but the former provides more of a challenge. While an increased network density would increase the number of possible routes to the verifying node, there is a class of attack dedicated to encouraging traffic to be routed through a node or set of nodes under the control of the adversary [37]. It would be possible to ensure delivery of the reports by monitoring traffic behaviour during the reporting phase and reporting on that in turn, but in order to ensure the delivery of the reports on the delivery of the initial reports, that phase would also need to be monitored, resulting in a potentially infinite number of messages being generated from a single event. In order to obviate this, an exhaustive analysis of the ways a mote can misbehave in a flat network was made (see appendix 1). In light of this study, a set of features was formulated which could, in a network with a certain minimum density and a certain minimum percentage of corrupted nodes, defend against these attacks, while attempting to guarantee delivery of uncorrupt and untampered packets through a valid and sensible route.

Feature definition

1. Anomaly reporting. Nodes have the ability to report anomalous behaviour both to the nodes in their immediate vicinity, and to the base station.

2. Checksumming. As packets are transmitted, a checksum of the payload is calculated and inserted into the packet. This can be checked by the receiving node, to detect mis-transmission, corruption, or interference.

3. Cooperative traffic monitoring. Each node behaves as a watchdog for the surrounding nodes. If a node receives a transmission for which it is not the ultimate destination, and does not forward it, this act will be reported. If a node originates a packet for which it is not the stated original source, this will be reported.

4. Packet buffering. Each node keeps a limited-duration copy of messages which it has overheard or transmitted. The time limit on these buffers is a generous estimation of the maximum time it should take for the receiving node to forward or acknowledge the transmitted packet. Nodes which overhear the transmission have a slightly longer timeout period than the node which sent it, to allow a limited number of retransmission attempts.

5. Packet comparison. All nodes which heard the previous hop of a message can compare the contents and header data of the buffered version with the freshly received version. If the message has been tampered with, the receiver can be informed to prevent the packet being forwarded further into the network. At this stage there are two options: Either inform the predecessor of the transmitter that the message was tampered with, triggering a re-routing of the message; or the overhearing nodes can vote on which version of the message is correct (if there is any disagreement), and a copy of the correct version can be transmitted onwards.

6. Location signature logging. Each node keeps a list of the expected signal strengths of transmissions from all the nodes from which it has heard, low-pass filtered over time.

7. Identity verification. By a mixture of 2 and 4, it is possible for individual nodes to make informed guesses about the identity of the node which sent a particular message. If, for example, a good signal strength is received from a node which has not been heard from previously, or a node is heard from with a signal strength which is greatly different from that which is expected, or a node transmits with the same ID as the receiving node, the identity of the transmitter is brought into doubt, and the incident is reported.

8. Full-path monitoring. When a packet is overheard, with the features above, it can be determined whether the predecessor should have been heard, and whether the successor should be heard. This allows detection of inserted and dropped packets, which can then be reported.

9. Routing analysis. With the integration of a transparent and predictable routing mechanism, individual nodes can make judgements about the routing decisions of the transmitting node. When given information about the transmitter, intended recipient, and ultimate destination of the packet, each node can examine its routing table and determine whether it would have routed the same packet in the same way. If not, the incident can be reported to the base station for analysis.

10. Link approval control. During deployment, the nodes go through a beaconing and logging stage in which they keep a log of all the nodes they can 'hear' transmissions from, and mark

them as 'approved' in their own row in a link permissions table. All other node numbers are marked as 'unknown'. At the end of this phase, nodes broadcast their own permissions row to the surrounding nodes, resulting in each node having a permissions table for each of its neighbours. When a packet is transmitted to a node from a source which is marked as unknown, verification is sought from the base station, which will make a decision based upon any localisation information it has, as well as information about any nodes which have been recently added to the network. It then returns this decision to the querying node, which will broadcast the result to its neighbours, who then update their link permissions tables.

11. Blacklisting. In addition to the above, links may be marked as 'forbidden'. This can be done locally by individual nodes which detect behaviour meeting certain pre-defined criteria, and also centrally, by the base station. The routing algorithm may be configured to avoid these nodes.

12. Central analysis. The analytical information collected through the network is collated and analysed at the base station by a more computationally rich system. With the addition of a standard network localisation and mapping system, for example, this would allow differentially anomalous signal strengths to be collated, providing the physical location of a malicious device. In addition to this, probabilistic judgements can be made about the validity of error reports. For example, in a network with density M, a node is likely to have on average N neighbours. If N/3 nodes report non-optimal routing of a packet, the routing is probably valid, and if 3N/4 reports are received, the routing is probably malicious.

Attack vector studies

There follows an analysis of each of the attack vectors from the attack trees in appendix 1, addressing the ways in which a generic packet transmission may be compromised.

Objective A: Cause a packet to be dropped

A1: Fail to forward a packet.

> In the event that a node fails to forward a packet, the previous node in the routing chain will fail to hear a retransmission of the packet. In this instance, the previous node will re-attempt the send a predetermined number of times before declaring it to have failed, reporting the failure, and sending the packet via an alternative route. In addition to this, the watchdog nodes surrounding the exchange will note the failure to forward and report it.

A2: Retransmit a packet to be forwarded via a unidirectional antenna, such that the previous node in the routing chain believes the packet to have been forwarded.

> In this instance, the watchdog nodes (or at least a subset thereof) will hear the previous, omnidirectional transmission, but not the subsequent, unidirectional transmission. This will be interpreted by these nodes as a packet drop, and reported. At the base station, reports of packet droppage will be received from the watchdog nodes, but not from the previous node in the routing chain. This will allow this event to be distinguished from (A1).

A3: Broadcast malformed packet, such that it is dropped by the next node in the routing chain

> This action has two potential outcomes. If the packet is badly enough malformed that it is not recognisable as being the same packet, or is not accepted by either the firmware or software layers of the receiving mote, it can be assumed that the previous node in the routing chain and the watchdog nodes will interpret it in the same way, if the network is homogenous, and the incident will be indistinguishable from a packet drop (A1). If the packet is received and identifiable as the successor to the previous packet, it will be compared to the buffered versions of the packet, and the mutation will be detected. In this case, the packet will be rerouted, and the incident will be reported.

A4: Transmit malformed packet to the next node in the chain using a unidirectional antenna, while transmitting an uncorrupted version to the previous node.

> In this case, the watchdog nodes will be unable to hear the retransmission of the packet, and the incident will be responded to in the same way as (A2).

A5: Forward the packet to an invalid node ID, using a standard broadcast antenna.

> This mode of misbehaviour will be noticed in two ways. Firstly, both the preceding node and the watchdog nodes will check the routing path against their list of approved links, see that the link is unknown, and report the incident. Secondly, all the aforementioned nodes will check the routing decision for the packet against their own internal heuristics, and report the discrepancy.

A6: Transmit a version of the packet routed towards an invalid node ID forwards using a unidirectional antenna, while transmitting a version routed to a legitimate destination backwards towards the previous node in the routing chain.

> As in the previous attacks involving two unidirectional antennae, watchdog nodes will detect this as a failure to forward the packet on the part of the corrupted node. In addition to this, any watchdogs in a position to intercept the onwards transmission will detect the malicious routing as in (A5), and report it in the same manner.

A7: Forward the packet to an out-of-range node.

> The watchdog nodes and the previous node in the routing chain will be able to compare this link to their table of link permissions and see that it has not been approved, and to their routing heuristics, which will detect the decision disparity. These will be reported.

A8: Transmit a version of the packet routed towards an out-of-range node ID forwards using a unidirectional antenna, while transmitting a version routed to a legitimate destination backwards towards the previous node in the routing chain.

> As in the previous attacks involving two unidirectional antennae, watchdog nodes will detect this as a failure to forward the packet on the part of the corrupted node. Any watchdog nodes in a position to intercept the forward transmission will compare the link to their link permissions tables, and the routing decision to their routing heuristics, and report the discrepancy.

A9: Cause a collision on another packet by producing a simultaneous transmission.

> This attack is effectively identical to (A1), and will be treated in the same way, except that a different node will be reported as having dropped the packet. The primary distinction between the two is that, in this attack, there is a chance that some of the watchdog nodes will hear an uncorrupted version of the transmission, due to being out of range of the collision packet. If the base station has localisation knowledge of the network, this can be used to help diagnose the issue and locate the offending node.

Note: In all the above attacks, and those below, a packet which is dropped, or for which the mode of retransmission is determined to be malicious, will be retried a set number of times, and then rerouted. Even in cases of localised jamming, or large areas of network failure, the packet can be delivered if any route to the destination remains. It may be favourable, however, to mark packets with a time-to-live counter which is incremented at point of reception, to prevent catastrophic congestion in badly challenged networks.

Objective B: Modify the contents of a packet

B1: Transmit interference over a packet transmission so that the payload becomes garbled.

> In this attack, at least some of the watchdog nodes will receive the garbled version of the packet. This will be interpreted as a payload mutation, and result in a report, and in the message being retried a finite number of times and then rerouted. In the case that some of the watchdog nodes are out of range of the interference and receive the uncorrupted version, they will not report corruption, and this may potentially be used as diagnostic information at the base station. The receiving node will calculate the checksum, and conclude that the packet has not been transferred correctly.

B2: Modify the payload of a packet prior to forwarding it.

> In this case, relatively straightforwardly, the previous node in the forwarding chain and the watchdog nodes will check the contents of the packet, detect the discrepancy, and react accordingly.

B3: Using unidirectional antennae, transmit a modified version of the packet to the next node in the routing chain, while transmitting an unmodified version towards the previous hop in the routing chain.

> As with the other cases involving the use of unidirectional antennae, the watchdog nodes will either, depending upon their position: receive the backwards transmission; receive the forwards transmission; or receive no transmission at all. Those that receive no transmission will report a droppage, and those that receive the forward transmission will (if they have a buffered record of the previous iteration of the packet) report a mutation.

In future attack profile analyses, the possibility of the attacker using a number of directional antennae to broadcast different packets will be omitted, as the solution is formulaic, and illustrated sufficiently in the preceding section.

Objective C: Interfere with the routing of a packet

C1: Route the packet upstream (away from the destination)

In this case, all the nodes able to hear the transmission will check the routing choice against their own routing heuristics. Assuming that the failure of a link is accompanied by local nodes updating their routing tables (which is the case in the routing algorithm implemented with this system, as explained in the routing section), a node routing a packet away from its destination is a suspicious action. This will be reported, and appropriate action can be taken.

C2: Route the packet to an invalid destination.

This action is functionally identical to action (A5), and is dealt with in the same way.

C3: Non-optimally route a packet.

In this case, all nodes which hear the transmission will consult their own routing tables and heuristics, and decide whether they would have routed the same packet the same way. If not, the incident will be reported. This is a subtle attack in many ways, and difficult to detect. In a network with a density such that each node has N neighbours, in this case, non-optimal routing reports from N/4 nodes probably indicate that the node is acting faithfully, whereas non-optimal routing reports from 3N/4 nodes probably indicate that the node is behaving selfishly or maliciously.

C4: Route packet through another malicious node.

If the malicious node in question is along a valid and correct routing path, then this will not be detected, and if it is not then it will be detected under one of the above categories. Once the packet reaches the malicious node, the behaviour of that node may be analysed as a new incident, with the exception of reports being raised by the previous node in the routing chain.

C5: Attract traffic by lying about routing information.

In the modification of the AODV routing algorithm used in this project, updates to a node's routing table are advertised locally. In this case, there are two distinct options.

1. The node advertises a falsely short route through a node which is, in fact, nearby and within range

2. The node advertises a short route through a node which is far away, or which does not exist

In case 1, at least some nearby nodes will have routing information for the proposed next node in the routing chain, and if the information advertised by the compromised node is false, it will clash with the cached information held by the nearby nodes (including, probably, the advertised node itself). When the routing information and heuristics are checked, the discrepancy will be detected, a report will be sent, and appropriate action will be taken.
In the second case, there will be no actively conflicting information in the nearby nodes in the routing table, but the link between the compromised node and the proposed next hop in the routing chain will be recorded as either unknown or invalid in the watchdogs' link

permissions tables. This will be reported, and appropriate action taken.

C6: Maintain a fast back-channel route to the destination.

This is a common kind of attack known as a wormhole attack. The compromised node maintains a link across some faster or longer-ranged medium to another compromised device, typically near a base station or cluster head. This allows the node to legitimately advertise a favourable route, and thereby cause a majority of local traffic to be routed through it.
In the system proposed in this document, this will be detected as a packet drop (attack profile A1) at the first device, and a packet insertion (attack profile D2) at the second node. Reports will be generated regarding both of these events, and collation of these reports at the base station will allow the type of attack and the location of the compromised nodes to be inferred.

C7: Be a node at a natural bottleneck in the network.

In many ways, this is not so much an attack profile as an attack characteristic, and as such it is very difficult to detect. While this allows the attacker access to a large amount of routed data, the advantage is lost if the node begins to misbehave in a detectable way, or if the network topology changes. While this is a vulnerability, it requires intimate knowledge of the network topology and geography, and a very high degree of precision from the attacker. In addition to this, in a homogenous, amorphous mesh network such as is generated by the AODV protocol discussed under the 'Routing' section, the routing topology is liable to change frequently and, to a certain extent, unpredictably.

Objective D: Insert a packet

D1: Transmit a packet while impersonating a node within range.

> In this case, the watchdog nodes will compare the link strength of the transmission to a locally logged record of the expected link strength from that node. If the location of the transmitting node varies dramatically from the logged value, then this will be reported. If the impersonated node receives the transmission, it will report the incident.

D2/7: Transmit a packet while impersonating a far away or nonexistent node.

> When this packet is received, the watchdog nodes consult their logs of expected signal strength and link permissions. If there is no recorded expected signal strength for the transmitting node, or if the link between it and the intended recipient of the packet has unknown or forbidden status, the incident will be reported. It is worth noting that, in the presented implementation, all links between nodes that were not in range of one another at deployment time are marked as unknown.

D3: Originate a packet with malicious contents.

> In this scenario, the adversary compromises a node and instructs it to, for the sake of argument, report bogus sensor readings. It is very difficult to react to this on a cooperative, distributed level without running the risk of allowing the nodes to prohibit the reporting of information which is 'surprising'. This is not an attack which is addressed by the system proposed in this research, but by restricting the adversary to modifying data solely from nodes which have been individually compromised, rather than having the freedom to tamper with data from faithful nodes *en route*, the attack cost has been raised dramatically.

D4: Originate a packet which has allegedly been forwarded from a nearby node.

> Nodes which overhear this broadcast will check the expected signal strength for the alleged previous node in the routing chain. If the signal strength is above a certain threshold, the watchdog will conclude that it ought to have heard the preceding packet, and report the incident. In addition, if the node which is recorded as having transmitted the packet to the compromised node receives the transmission, the incident will also be reported as an impersonation.

D5/6: Originate a packet which has allegedly been forwarded from a far away or nonexistent node.

> The watchdog nodes overhearing this packet will check the alleged receiver against their lists of expected signal strength and link permissions. When the link between the currently transmitting node and the supposed previous hop in the routing chain is found to be unknown or forbidden, the incident will be reported.

Summary

Analysis of the initially identified attack vectors and solutions, as described above, suggests that the deployment of the proposed set of features within a homogenous wireless sensor network would severely limit the activity scope of a compromised node within the network. The distributed and cooperative nature of the monitoring system means that any externalised misbehaviour will be reported. Externalised misbehaviour is defined in this case as misbehaviour which has an effect visible from outside the node. For example, modifying the contents of a packet routed through the node is a form of externalised misbehaviour: the discrepancy between the packet contents before and after routing is visible to an outside observer. Reporting bogus sensor readings, however, is internalised misbehaviour. The true values that the sensors report are tangible only within the confines of the node, and so no outside observer can prove the discrepancy. Leaking information from the network, through a channel which is intangible to the network infrastructure, is also internalised behaviour, in that the difference between the required and enacted behaviours is not visible from the point of view of the observers, i.e. the peer nodes.

In this way, while certain behaviours may not be prevented by this system, including the examples illustrated in the previous paragraph, the compromised nodes must continue to perform their prescribed actions or risk detection. This carries the obvious cost of loss of confidentiality and loss of reliability for the data reported directly from the compromised nodes. However, in a scenario where the network has a high enough density and a low enough proportion of compromised nodes (see the limitations section, later), and where the attacker instructs the compromised nodes to avoid detection, the network can continue to route collected and useful data through the compromised nodes, to continue to use them as part of the network, while maintaining a relatively high level of certainty regarding data integrity and accessibility.

Program structure

The naïve implementation

The TinyOS programming idiom is deeply unintuitive to anybody who is not used to the idiosyncrasies of embedded systems programming. The operating system, written and programmed in network embedded C (nesC), is event driven, being partially designed to real-time operation standards, and so does not support multithreading in any 'traditional' sense. Events trigger actions, which are then run linearly and sequentially. While an event is being handled, interrupts are disabled, and system interrupts can only be buffered for a very short time. Operations can also be specified in terms of threads, which are sets of operations which are run at a lower priority level than the command and event operations. This leaves the system, essentially, with two usable threads – the non-interruptable system and event thread, and the task thread, which is handled using the 'spare' cycles remaining when the former's demands have been satisfied.

There is also no firm separation between kernel space, user space, and driver space. All software objects are specified in terms of modules, with defined interfaces by which they communicate. Interfaces are directional, provided by one module and used by the other, and may contain commands – calls made within the interface using modules which trigger code execution in the providing module, and events, which are the converse. By specifying connections between the users and providers of these interfaces in configuration files, they can be wired together to create complete programs and systems of behaviour. The entire operating system is constructed out of these units, and it can be unnerving at first to investigate the underlying mechanism behind an operation and suddenly find oneself staring directly into the implementation of the task scheduler.

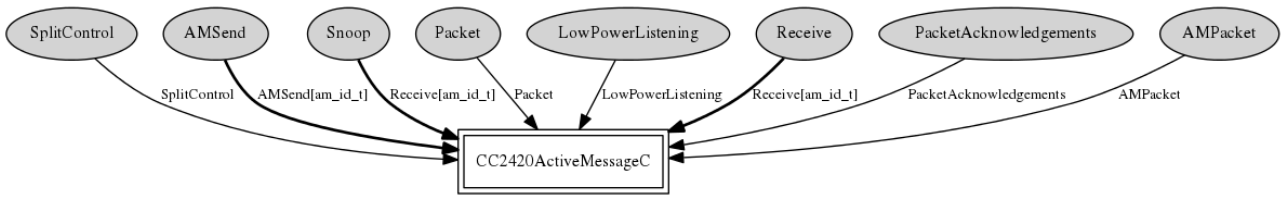


Figure 5: Connections to ActiveMessageC, CC2420 implementation

The first attempt at constructing a set of components which would perform the necessary processing, copying, buffering, timing and monitoring to provide the above tasks was not formulated with a satisfactory understanding of the implications of the above information. The first design relied, for example, upon the SubReceive.receive event (the event triggered by the reception of a correctly typed message by the underlying ActiveMessageC, as pictured above), sequentially constructed a digest of the message, queried the routing system for a destination, modified the information in the protocol header, and instructed the underlying sending mechanism to transmit the message to the appropriate address. The AMSend.sendDone event, which is triggered when a send is completed, was responsible for copying the message into the listening buffer and setting the timer. The net result of this was that, although the implementation functioned, an enormous number of packets were dropped due to the interrupts signalling a reception occurring during a non-interruptable operation. The control flow of this initial implementation is included in appendix 2.

The idiomatic implementation

In order for a system of this level of complexity to function satisfactorily, it is necessary to implement the message processing pipeline asynchronously. When a message is received, as little direct processing must be performed in the event handler as possible. In order to achieve this, a large amount of refactoring was performed. The set of necessary operations was reorganised into a set of processing stages, each with its own associated queue. As an example, messages received through the subreceive component are first copied into the reporting queue for analysis, and the reporting task is posted. The same message is then either copied into the verification queue if the link permission status is unknown, or the routing queue if they must be forwarded to another node, or the reception queue if they have reached their final destination. Each of these tasks, of course, is executed with system interrupts enabled, so as not to interfere with the continued running of the system. Certain operations must be done uninterruptably, such as copying or comparing message contents, checksumming payloads, and building message reports (various parts of which rely on aspects of global system state, meaning that the analysis must be done during a period when the state does not change). However, with a well-informed and planned approach to the operating principles of the platform, these can be minimised and major system disruptions can be eliminated. The control flow of this version of the system is illustrated in appendix 2.

Routing

In order for the system to remain as versatile and configurable as possible, a large number of the features of the CAMUnit (checksummed active message unit) have been retained as separate modules, with interfaces designed for maximum interoperability. The routing module, for example, has been kept completely separate, as it is the most likely one to be replaced between different kinds of deployment. Most freely available TinyOS implementations of multihop routing protocols have been built into a cohesive networking layer as part of a complete implementation, and so utilise the standard AMSend and Receive interfaces. In order to separate the routefinding operation set from the delivery operation set, a new interface was designed with the following commands and events:

- getNextHop(destination) – a command which requests the next hop towards a destination
- hopFailed(source, destination) - a command which instructs the routing engine that a route has failed
- nextHopFound(nextHop, destination) - an event which signals that the next hop towards a destination has been found
- checkrouting(message) - a command which requests the routing engine to evaluate the routing decisions made in the transmission of a message by a nearby node

Firstly, in order to observe more clearly the sequence of events which take place when a message is forwarded, a fake routing algorithm was constructed, which simply routes messages linearly along a sequence of consecutive node IDs, i.e. a message from 0 to 4 would take the route 0 → 1 → 2 → 3 → 4. This routing algorithm was of great utility during the development and testing of the other features. In this version, the implementation of each of the above commands was trivial.

The next routing algorithm to be integrated was Kim Junseok's implementation of the AODV protocol for TinyOS [38], which was integrated into a complete communications stack as described above. The AODV protocol is an ad-hoc, on demand protocol, which both makes it very resilient,

and makes it very difficult for the adversary to predict the resulting topology in any level of detail. Without going into a full exposition of the workings of the protocol in this report, which would seem redundant, the mechanism is based upon report flooding and aggregation. Each possible route through the network receives a routing request. Any of these which reach the intended destination trigger routing replies back along the same route. Both of these actions have the potential to update the routing tables at each node they pass through. It is possible to report routes as being broken by transmitting a route error report back to the source of the message, which clears the route from the cache of all nodes it passes through.

In the received implementation, the routing request flood was triggered by a request for a message to be sent to a particular node, and modifying this to support the routefinder interface was a simple case of calling the same function with similar arguments from the handler of the getNextHop function. Similarly, when a route reply was received, the implementation was modified such that this triggered nextHopFound, rather than a subsend event. The hopfailed command was configured to trigger a route error transmission, and the checkrouting command was configured to check the existing routing tables without triggering any update requests.

While both of these routing algorithms work, and the routefinder interface has proven to be a useful standard interface for modular routing algorithms, at this stage of the implementation, the AODV request and reply communications stack remains unsecured, which is a severe vulnerability. This issue is discussed in more depth in the 'Further Work' section.

Link strength logging

Each transmission received by the CC2420 chip results in locally-stored metadata regarding the strength and nature of the transmission, which is relatively easily accessible. In order to keep a log of expected signal strengths, it was sufficient to maintain an array of values, large enough to hold the number of neighbours that a node might be expected to have, and update the value using a simple low-pass filter whenever a transmission from that node was received. The threshold values of what constitute acceptable signal strength and abnormal signal strength discrepancy can then be tuned manually.

It would be desirable, in a field deployment, to allow these values to be determined automatically, and for the trigger threshold for strength discrepancy to be related to the variance of the value, rather than simply the current value. This issue is discussed in more depth in the 'Further Work' section.

Link control

The link permissions control system implemented for this project is based upon a set of valid links developed through a beaconing phase at deployment time, combined with a method of requesting permission to form new links from the base station, and the ability to revoke link permissions at any time if a node begins behaving in a way which is deemed unacceptable. Each node keeps an N x N record of link permissions, where N is the maximum expected number of nodes (including itself) of which it is expected to have knowledge.

The beaconing phase takes place in two parts. During the first phase, each node repeatedly broadcasts lightweight beacon messages. Once a node has heard a beacon from another node during this period, it marks that node as having a valid link in its own row of the permissions table. During the second phase, the nodes repeatedly broadcast their own rows from their own permissions tables.

When a node receives one of these packets, it updates the row in its permissions table which corresponds to the sender of the packet. This allows nodes to have knowledge of all the links which involve a node that is within transmission range, rather than just links between themselves and all nodes within transmission range. These methods of updating permissions function only during the beaconing phase. This method of storing permissions allows for asymmetrical links: it is possible to have connections within the network across which traffic is only permitted in one direction. It would be trivial to add further states to the permissions table, to define, for example, links across which only a certain type of traffic is permitted.

During normal operation, if a node receives a transmission across a non-approved link, the message is buffered into a validation queue while approval is sought from the central authority – an exchange which takes place through the protected message transit system described in this document. Nodes which overhear the transmission consult their routing tables to determine how long to expect the forwarding exchange to take. If the link is unknown, the watchdog nodes expect forwarding to take longer than it otherwise would. This makes the assumption that there must be a route back to the central node from a node which is already part of the network but which does not pass through the newly added node – an assumption which, as things stand, presents an issue when reconnecting parts of the network which have been temporarily isolated. This issue is discussed in more depth in the 'Further Work' section.

Message format

The messages transport layer within this system is provided by the TinyOS active message protocol, an unreliable, single-hop transport protocol. Within the transport layer wrapper, a header format has been designed which contains the necessary information for an overseen, reliable, multihop protocol, such as the one implemented in this research. This format is referred to throughout this document as the checksummed active message (CAM) format. The headers format for this protocol contains information about the enclosed message type and size, the original source and ultimate destination of the message, the previous hop in the routing chain and, as the title suggests, a message checksum.

Reporting

The main set of design decisions relating to the reporting procedure were to do with what information to include in the reports, and how these reports should be triggered. As the analysis of the message is performed during report generation, it makes intuitive sense for the report to be first generated, and then compared to the rules about which kinds of events are worth reporting. After a review of the attack vector studies detailed in the design section, the following specification was decided upon.

Digests: Message digests are simply information about the contents of a packet, not the manner of its transmission. These can be generated from buffered information for packets which have been dropped. They contain the following fields:

- reporter – the ID of the node sending the report
- h_src – the transmission source as listed in the AM header
- h_dest – the transmission destination as listed in the AM header
- h_len – the length of the message as listed in the AM header
- src – the original source of the message as listed in the CAM header
- prev – the previous hop in the routing chain, as listed in the CAM header

- curr – the current hop in the routing chain, as listed in the CAM header
- next – the next hop in the routing chain, as listed in the CAM header
- dest – the ultimate destination of the message, as listed in the CAM header
- type – the type of the enclosed message, as listed in the CAM header
- id – the message sequence ID, as listed in the CAM header
- len – the length of the enclosed message, as listed in the CAM header

Analytics: These are digests of analytical information regarding the transmission mode of the packet. They can, of course, only be generated from active packet transmissions, and not from an absence thereof. As such, dropped packets may only result in a message digest being delivered to the base station, whereas any other manner of anomaly may result in a full report (digest + analytics) being delivered. The analytics packets consist of the following fields:

- lqi: the link quality indication (essentially signal strength) of the received transmission
- headers_agree: whether the information shared between the AM and CAM headers agrees
- valid_len: whether all the length values are valid (this is important, as overly long length values can be used to exploit buffer overrun vulnerabilities in TinyOS [5]).
- anomalous_lqi: whether the lqi differs significantly from the expected value
- first_time_heard: whether this node is being heard for the first time
- checksum_correct: whether the checksum is in accordance with the payload
- impersonating_me: whether the transmitting node is impersonating the reporter
- checksum_matches: whether the checksum matches the cached version, if any
- payload_matches: whether the payload matches the cached version, if any
- valid_routing: how the routing decision compares to the choice the local routing algorithm would have made (unknown, upstream, non-optimal, okay)
- link_status: the permission status of the link from the transmitter to the receiver.

These reports are then filtered by characteristic to determine whether an event has occurred which must be reported to the central authority. Due to time constraints, the filtering mechanism currently contains only three determinants: whether a node is impersonating the reporter, whether a payload has been tampered with, and whether a node is broadcasting with an anomalous LQI. Suggestions for improvements on this are detailed in the 'Further Work' section.

Cooperative monitoring

The system of cooperative monitoring is relatively simple in a network of this type. As all nodes except the base station have approximately the same performance profile (i.e. the network is homogenous), each node can infer what constitutes 'faithful behaviour' for the surrounding nodes by examining its own behaviour. Messages are buffered in message queues which have been designed for this purpose, and have been implemented in both first-in-first-out and earliest-alarm-time-first varieties.

When a transmission is heard, the message is buffered, and an alarm time is calculated from the appropriate maximum estimated interval and the current system time. By this time, either a forward, a retransmission, a rerouting attempt, or an acknowledgement of the packet should be heard. In order to avoid the cost of running multiple timers, an earliest-alarm-time-first queue is used, and every time a message is added to or removed from the queue, a task is run which retrieves the earliest alarm time, compares it to the current system time, and either signals the timer to fire immediately, or sets the timer to fire after the appropriate period. The maximum estimated interval

for an event is determined from the type and circumstances of the event: the node which sent the packet will have a shorter alarm time than those which overheard it, to allow the retransmission attempt cycle to take place without triggering reports, and transmissions which take place over links which have not been approved have a much longer alarm interval, as the period must allow for approval to be sought from the central authority.

When the timer fires, the node retrieves the earliest message from the queue, and takes action as illustrated in the following diagram.

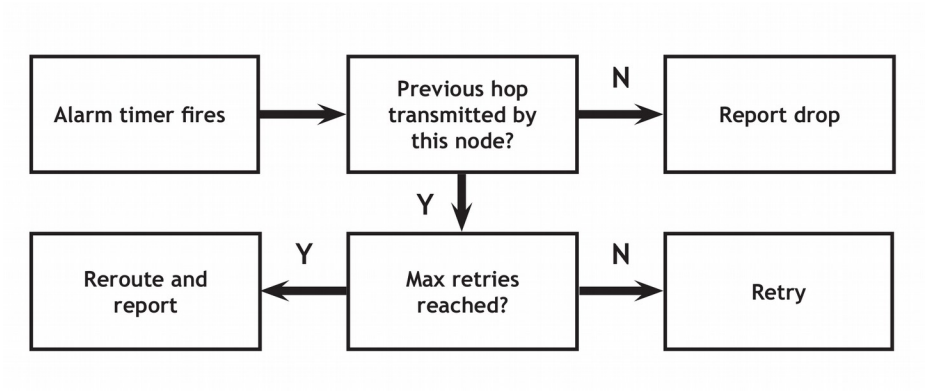


Figure 6: Timer.fired decision tree

The message queues are also searchable. When the function removeMsg is called on a queue with a pointer to a message object as an argument, the queue will return a message pointer which is either null, if the message was not in the queue, or a pointer to the buffered version of the message. This function intelligently determines either message identity or message continuity, and is sensitive to the type of message being compared, although it does require all of the messages within a queue to be of a single type. This allows, for example, the heard message task, which deals with messages which were overheard, but addressed to different nodes, to update the 'hop' state of a message: that is, which nodes the next routing stage of the message is supposed to be between. When a message reaches its final destination, an ack is broadcast containing the full contents of the message, to prove that the message was received intact and correctly.

Evaluation

In the field of wireless sensor network security, as in any security field, there are a number of critical and conflicting requirements. The primary concerns are these: security, computational tractability, energy consumption, and cost. As an example of the seeming incompatibility of these goals, take two of the more ground-breakingly secure advancements of recent years.

1. The implementation of public-private key cryptography on sensor nodes, implemented by Tan et al [4]. This is an extremely powerful cryptographic advancement, which allows transfer of information between nodes in a manner that is secure and satisfactory even in the presence of a computationally rich adversary who has physically captured one of the nodes. The technology requires a breakout board, the addition of which to an entire sensor network deployment would be extraordinarily costly, and the operations themselves expend a very large amount of energy.
2. The implementation of provably secure erasure and attestation by Perito [5]. This procedure overwrites all of the areas of memory in which even the most ingeniously implemented rootkit might be hiding, and proves incontrovertibly (contingent upon the assumption of adversarial silence, mentioned previously in this document) that the memory has been overwritten. The procedure, however, in its current implementation, involves almost 60 seconds of constant transmission, which is catastrophically expensive in terms of energy.

In other words, the advancement of the goal of security almost inevitably involves a certain amount of compromise with regard to one or more of the other goals. The development of this project is generally in line with the aims of Perito in the aforementioned attestation implementation, in that it is a provably secure mechanism which is evidently impractical for deployment in anything approaching an industrial or enterprise network, which is developed with the intention of later optimising the costs and scaling back the compromises on the other critical fronts.

Excited to have built a homogenous monitoring system. Talk about the panopticon – for high value information in a high risk environment, ubiquitous surveillance has the potential to be a very valuable tool.

Location signatures are able to, in a laboratory environment, detect node movement of as little as 4 metres (approximately 60% of the time, with this number increasing to 75% when the node moves on the other side of a solid object. Did not perform field study, but know that this level of accuracy will become much lower in a noisy environment. Some use, but relatively limited.

The system, as it stands, functions. It is more of an experimental object than a functional tool, but it presents an interesting foundation for further work.

Low power mode works. While packets are not received the first transmission from low power mode by watchdogs, they are not received first go by the recipient either., If the network is in low power mode, it will result in a dual transmission, which gives the surrounding nodes a much better chance to overhear, buffer, and analyse the transmission.

Some new work, or new combinations of existing work, but to be frank very limited work of an actually groundbreaking nature

Difficulties with report flooding, ghost reports

Conclusions and future work

Achievements

In this project, a system has been implemented which performs the following functions.

1. Detects predetermined modes of misbehaviour in a distributed way

   In this system, every node which acts as a collector also acts as a monitor. The motivation behind this goal was to eliminate the 'single weak point' characteristic of systems which are monitored by a cluster head, or by elected watchdog nodes. While the attacker may easily outclass the network hardware in terms of computational ability, the network hardware is relatively certain to outclass the attacker in terms of density and population. By focusing on the maximum possible utilisation of node density and profusion, this system represents an experiment in unilateral peer monitoring methodologies.

2. Functions in a way which is platform agnostic

   Due to the position of the security layer represented by this project between the existing TinyOS active message protocol and what might be loosely defined as user space, this system can be trivially ported to any other platform which is capable of running TinyOS. The only modifications that would need to be made would be the precise type names used in the definitions of the header and metadata pointers, and (for non TI radio chipsets) the metric used for link quality indication.

3. Allows for asymmetrical link restriction and approval with an extremely fine-grained level of control

   The link control system not only automatically allows links which were possible during the starting topology, but allows for links which only allow traffic in one direction, and allows for the modification of link permissions in real time, at the granularity of a single unique node-node link.

4. Compatible with a large number of multihop unicast routing protocols

   The top-level design of this project is such that the routing protocol is contained within a separate module. Due to this, it is only necessary to implement the routefinder interface within the routing protocol to ensure compatibility. This is often a case of simply causing the route finding algorithm to be triggered by a RouteFinder.getNextHop command rather than an AMSend.send command, and causing the discovery of a route to trigger a RouteFinder.hopFound event rather than a SubSend.send command.

5. Detects localisation anomalies

   The logging of localisation signatures allows certain subsets of impersonation attack to be detected and defended against.

6. Allows certain types of system traffic (link validation exchanges) to be routed through the

system which they support

   All link validation requests and confirmations are routed through the CAM stack, meaning that they can be both authenticated and guaranteed delivery. This allows for a degree of trust in the link control mechanism which would be impossible otherwise.

7. Delivers extremely fine-grained and configurable diagnostic information to the base station, alongside the data which the system is configured to collect

   All of the events and anomalies which are detected by this system trigger the dispatch of a report which gives a high level of detail about the event and the circumstances surrounding it. These reports may be collated at the central authority, allowing the development and support of analytical systems which are very closely targeted to the needs of the individual deployment.

8. Provides reliable delivery of packets

   By the use of continuous monitoring and buffering of packets, as well as a reliable routing algorithm, the system is able to provide guarantees regarding the delivery of packets, as long as a route to the destination exists.

9. Self-heals routes in the event of network disruptions

   As the system detects the failure of links, and is able to communicate them back to the routing engine as and when they occur, and updated routing information may be propogated throughout the system.

Talk about minimum density and number of uncorrupted nodes. Diagrams.

Limitations

First and foremost, this constitutes an experimental implementation of a novel topographical device, and as such it is not, and is not intended to be, a device suitable for immediate deployment. With that in mind, however, the following limitations are unavoidably evident.

1. Anomaly and event reports are not currently protected by the security layer
2. No cryptographic solution is currently implemented
3. The routing gradient comprehension system for AODV does not currently work (suggest fix)
4. The location signature system is only useful in cases of very large geographical discrepancy
5. Little attention has been paid to denial of service attacks within the network
6. The network headers currently contain redundant information
7. The thresholds and settings are not currently centrally adjustable
8. Very heavyweight, costly in energy

9. Although intended to be configurable, system is currently not
10. Node blacklisting not yet been implemented

Suggestions for future work

Possibilities for future work on this system are myriad, but suggest the following few as higher priority

1. Work on report flooding
2. node blacklisting – give example implementation
3. Easy configuration method for sysadmins
4. Implement a version which allows, say, minisec to be utilised over a multihop route. (build up keys)
5. Add range limit comprehension, works with larger networks. Give suggested implementation.

Perform a more thorough formal threat analysis of the system as it stands in light of these changes.

Bibliography

[1] Q. Huan, I. Avramopoulos, H. Kobayashi, and B. Liu, “Secure data forwarding in wireless ad hoc networks,” in *2005 IEEE International Conference on Communications, 2005. ICC 2005*, 2005, vol. 5, pp. 3525–3531 Vol. 5.

[2] K. Ioannis, T. Dimitriou, and F. C. Freiling, “Towards intrusion detection in wireless sensor networks,” in *Proc. of the 13th European Wireless Conference*, 2007.

[3] T. Park and K. G. Shin, “Soft tamper-proofing via program integrity verification in wireless sensor networks,” *Mob. Comput. IEEE Trans. On*, vol. 4, no. 3, pp. 297–309, 2005.

[4] H. Tan, W. Hu, and S. Jha, “A hardware-based remote attestation protocol in wireless sensor networks,” in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2010, pp. 378–379.

[5] D. Perito, “Code Execution on Embedded Devices,” 2011.

[6] Q. Wang, K. Ren, S. Yu, and W. Lou, “Dependable and Secure Sensor Data Storage with Dynamic Integrity Assurance,” *ACM Trans. Sens. Netw.*, vol. 8, no. 1, pp. 1–24, Aug. 2011.

[7] D. Perito, “Exécution sécurisée de code sur systèmes embarqués,” Université de Grenoble, 2011.

[8] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, “On the difficulty of software-based attestation of embedded devices,” in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 400–409.

[9] A. Perrig and L. V. Doorn, *Refutation of “on the difficulty of softwarebased attestation of embedded devices*. .

[10] D. Perito and G. Tsudik, “Secure Code Update for Embedded Devices via Proofs of Secure Erasure,” in *Computer Security – ESORICS 2010*, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds. Springer Berlin Heidelberg, 2010, pp. 643–662.

[11] W. Hu, H. Tan, P. Corke, W. C. Shih, and S. Jha, “Toward trusted wireless sensor networks,” *ACM Trans. Sens. Netw.*, vol. 7, no. 1, pp. 1–25, Aug. 2010.

[12] R. Román, C. Fernandez-Gago, J. López, and H. H. Chen, “Trust and reputation systems for wireless sensor networks,” *Secur. Priv. Mob. Wirel. Netw.*, pp. 105–128, 2009.

[13] D. Martins and H. Guyennet, “Wireless Sensor Network Attacks and Security Mechanisms: A Short Survey,” in *2010 13th International Conference on Network-Based Information Systems (NBiS)*, 2010, pp. 313–320.

[14] C. Lynch and F. O’Reilly, “Processor choice for wireless sensor networks,” in *Proc. ACM Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005, pp. 52–68.

[15] T. Giannetsos, T. Dimitriou, I. Krontiris, and N. R. Prasad, “Arbitrary Code Injection through Self-propagating Worms in Von Neumann Architecture Devices,” *Comput. J.*, vol. 53, no. 10, pp. 1576–1593, Dec. 2010.

[16] A. K. Pathan, H.-W. Lee, and C. Hong, “Security in wireless sensor networks: issues and challenges,” in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, 2006, vol. 2, p. 6 pp.–1048.

[17] A. Perrig, J. Stankovic, and D. Wagner, “Security in wireless sensor networks,” *Commun. ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[18] L. Buttyán, D. Gessner, A. Hessler, and P. Langendoerfer, “Application of wireless sensor networks in critical infrastructure protection: Challenges and design options [Security and Privacy in Emerging Wireless Networks],” *Wirel. Commun. IEEE*, vol. 17, no. 5, pp. 44–49, 2010.

[19] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, Mar. 2002.

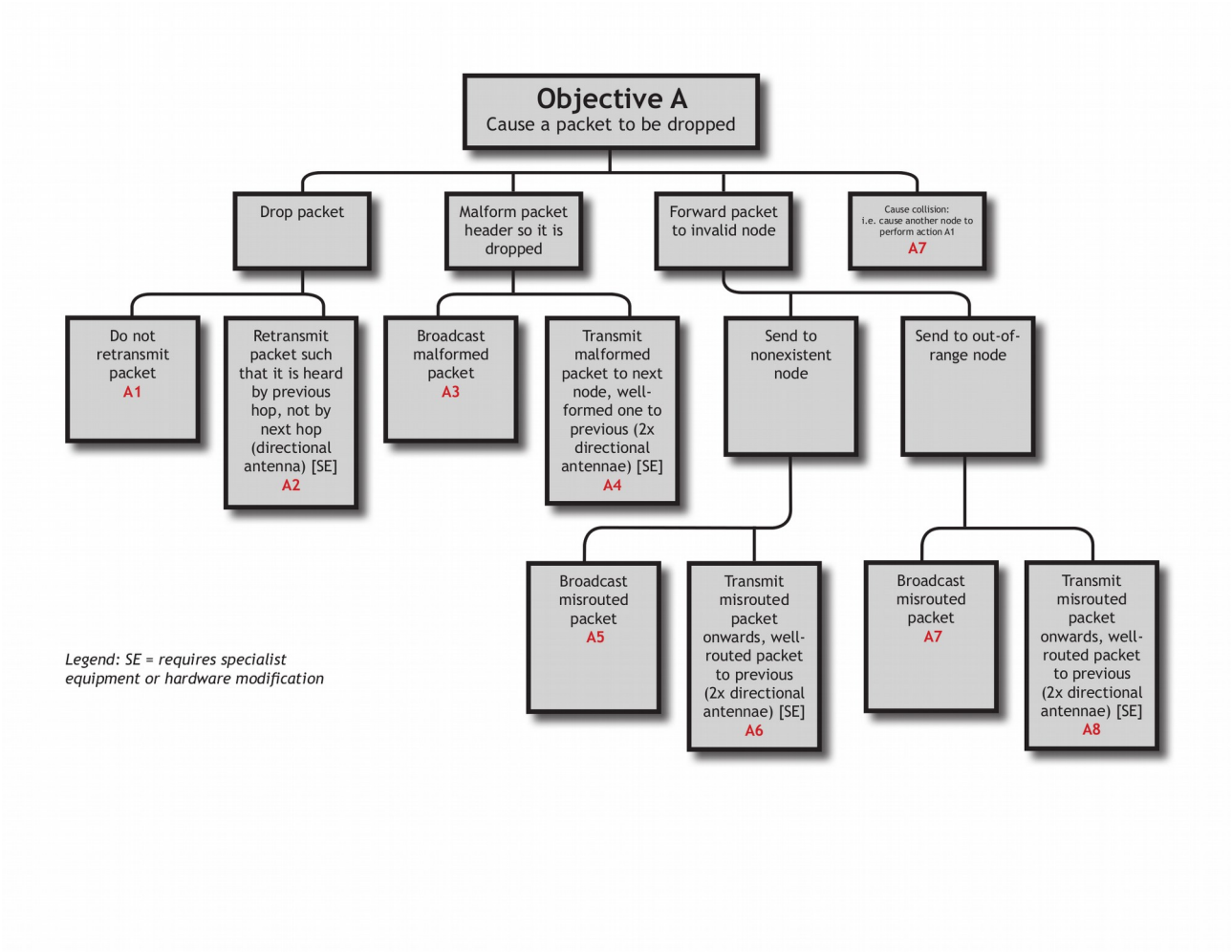
[20] M. Hatler, D. Gurganious, and C. Chi, “802.15.4 & ZigBee: Expanding markets, growing threats.” A Market Dynamics report (9th edition), 2012.

[21] M. Saxena, “Security in wireless sensor networks-a layer based classification,” *Dep. Comput.*

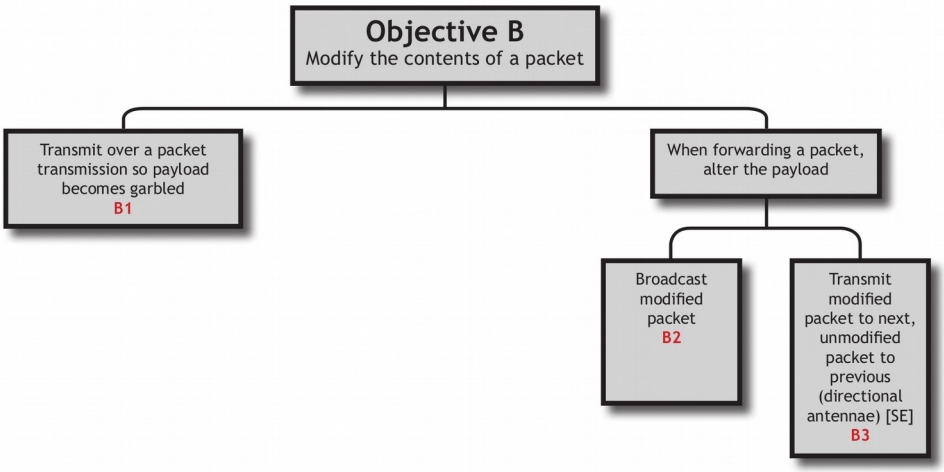
Sci. Purdue Univ., 2007.
[22] D. J. Malan, M. Welsh, and M. D. Smith, “A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography,” in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004*, 2004, pp. 71–80.
[23] S. Ravi, A. Raghunathan, and S. Chakradhar, “Tamper resistance mechanisms for secure embedded systems,” in *17th International Conference on VLSI Design, 2004. Proceedings*, 2004, pp. 605–611.
[24] A. Seshadri, M. Luk, and A. Perrig, “SAKE: Software attestation for key establishment in sensor networks,” in *Distributed computing in sensor systems*, Springer, 2008, pp. 372–385.
[25] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, “SCUBA: Secure code update by attestation in sensor networks,” in *Proceedings of the 5th ACM workshop on Wireless security*, 2006, pp. 85–94.
[26] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, “SWATT: softWare-based attestation for embedded devices,” in *2004 IEEE Symposium on Security and Privacy, 2004. Proceedings*, 2004, pp. 272–282.
[27] A. Francillon, C. Castelluccia, D. Perito, and C. Soriente, *Comments on Refutation of On the difficulty of software-based attestation of embedded devices*. Technical Report, INRIA, 2010.
[28] “Timeline of computer security hacker history,” *Wikipedia, the free encyclopedia*. 06-Jun-2014.
[29] J. Brandon, “A Short History of Hacking,” *mental floss*, Jun-2006.
[30] D. Zhang and L. Donggang, “DataGuard: Dynamic Data Attestation in Wireless Sensor Networks,” *2010 IEEEIFIP Int. Conf. Dependable Syst. Netw.*, 2010.
[31] P. Wagle and C. Cowan, “Stackguard: Simple stack smash protection for gcc,” in *Proceedings of the GCC Developers Summit*, 2003, pp. 243–255.
[32] S. Axelsson, “Intrusion detection systems: A survey and taxonomy,” Technical report, 2000.
[33] I. Butun, S. D. Morgera, and R. Sankar, “A Survey of Intrusion Detection Systems in Wireless Sensor Networks,” *IEEE Commun. Surv. Tutor.*, vol. 16, no. 1, pp. 266–282, First 2014.
[34] R. Roman, J. Zhou, and J. Lopez, “Applying intrusion detection systems to wireless sensor networks,” in *Consumer Communications and Networking Conference*, 2006, vol. 1, pp. 640–644.
[35] S. Shin, T. Kwon, G.-Y. Jo, Y. Park, and H. Rhy, “An Experimental Study of Hierarchical Intrusion Detection for Wireless Industrial Sensor Networks,” *IEEE Trans. Ind. Inform.*, vol. 6, no. 4, pp. 744–757, Nov. 2010.
[36] W. Xin-sheng, Z. Yong-zhao, X. Shu-ming, and W. Liang-min, “Lightweight defense scheme against selective forwarding attacks in wireless sensor networks,” in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. CyberC ’09*, 2009, pp. 226–232.
[37] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, “SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust.,” in *NDSS*, 2012.
[38] J. young Kim, R. D. Caytiles, and K. J. Kim, “A Review of the Vulnerabilities and Attacks for Wireless Sensor Networks.,” *J. Secur. Eng.*, vol. 9, no. 3, 2012.
[39] J. Kim, “AODV implementation on TinyOS 2.x.” University of Arizona, 28-Apr-2011.

Appendix 1: Attack Trees

Objective A: Cause a packet to be dropped

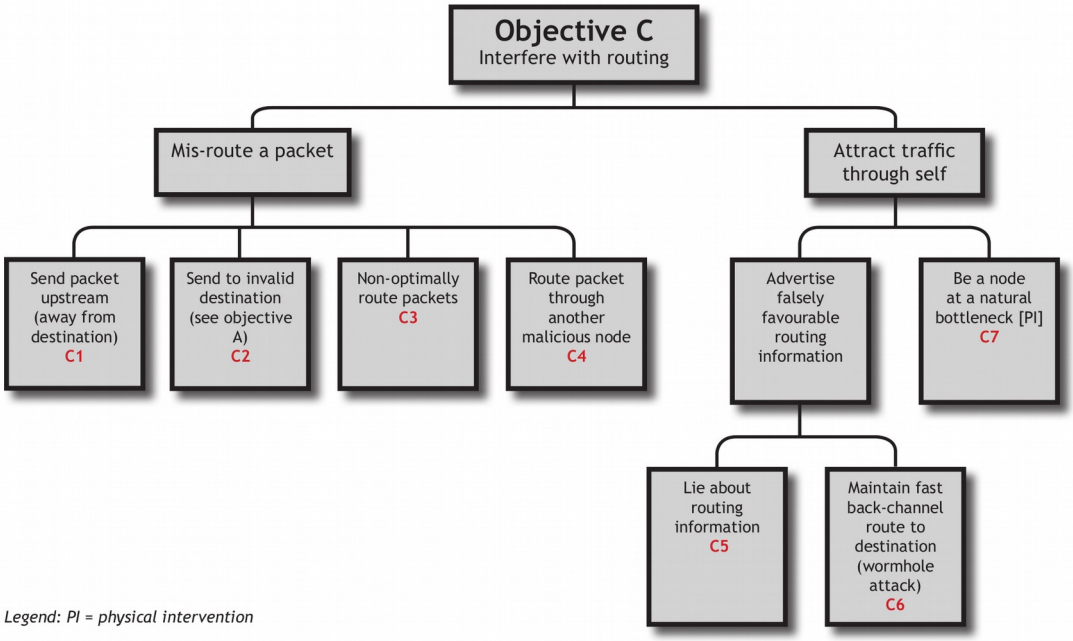


Ojective B: Modify the contents of a packet



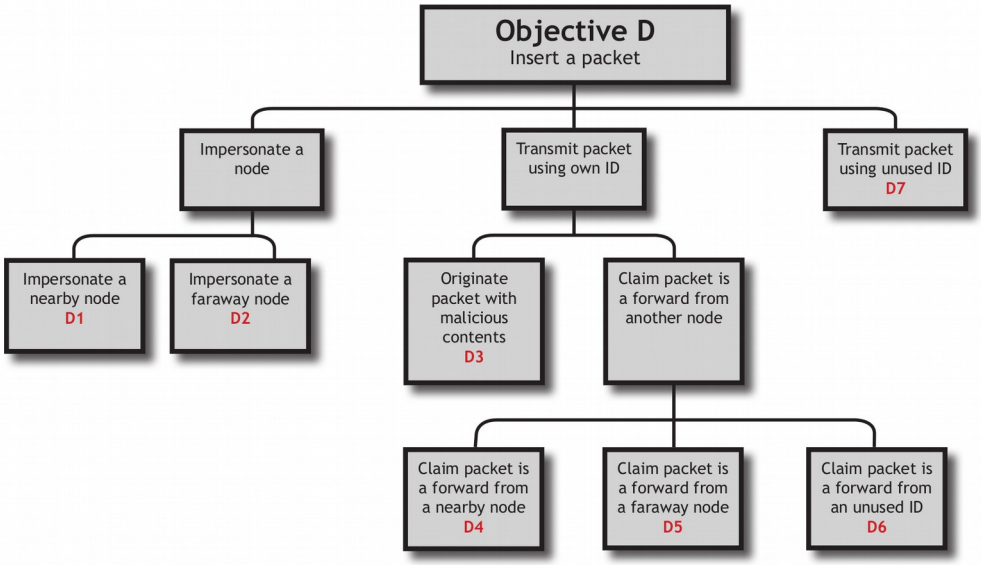
Legend: SE = requires specialist equipment or hardware modification

Objective C: Interfere with routing



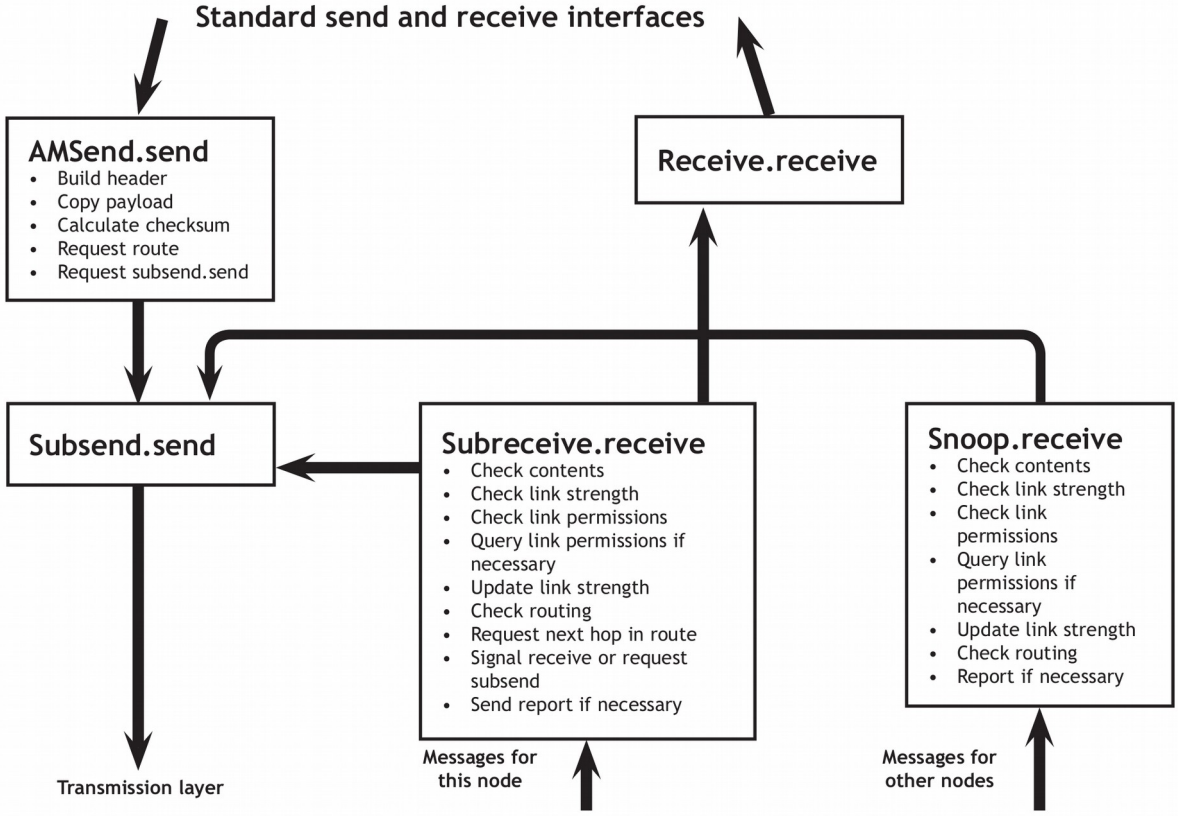
Legend: PI = physical intervention

Objective D: Inject a packet



Appendix 2: Implementation version schematics

Naïve implementation



Idiomatic implementation