Lowering the Impact of Intrusion Detection on Resources in Wireless Sensor Networks using a Domain Specific Language and Code Generation Techniques

Christophe Van Ginneken, Jef Maerien, Christophe Huygens, Danny Hughes, Wouter Joosen
iMinds - DistriNet - KU Leuven
B-3001, Leuven, Belgium
Christophe.VanGinneken@student.kuleuven.be,{firstname.lastname}@cs.kuleuven.be

***Abstract*—Introducing intrusion detection in wireless sensor networks proves to be a battle for resources. Implementing and optimizing a collection of detection algorithms to match available resources might very well be an unacceptable economic burden. This paper introduces a domain specific language to formally describe such algorithms and proposes the use of code generation techniques to automate the production of detection software. This automated process allows for the optimization of resource usage. Specifically the optimization of the execution time of the algorithms and the use of the energy-costly wireless radio are prime candidates. A prototype code generator was realized to show that these techniques can be implemented effectively, combining different intrusion detection algorithms in such way that they impose less impact on the resources than the sum of the impact of each algorithm by itself.**

I. Introduction

A wireless sensor network (WSN) is constructed using a large number of autonomous embedded devices, also called nodes. Their autonomy is rather absolute, being most of the time battery-powered and deployed in open terrain. Examples of WSNs include monitoring systems in volcanic regions [1] or flood areas [2]. Preliminary successes have now even steered researchers towards the introduction of WSNs in cities and homes, thus creating smart cities [3] and smart homes [4].

Due to their involvement in more and more personal applications, securing these nodes should be a priority. When we entrust these autonomous systems with some of our most intimate information, for example regarding our health [5], we would of course like them to be able to protect this information.

Securing WSNs, and especially single nodes, is a daunting task [6]. Mostly due to their autonomous nature, nodes are most of the time physically accessible. With physical access comes a wide range of potential, physical attacks [7] that are hard to detect, let alone prevent. Even if the nodes aren't physically accessible, their ways of communicating with the outside world through various forms of wireless communication, offer a plethora of possibilities [8] to attack them, ranging from low-level meddling with the routing of packets in the network, up to the application level.

Preventing intrusions should be a primary objective. Often this turns out to be impossible and intrusions are sometimes only noticed post-mortem. When prevention is not guaranteed, a second line of defence consists in the detection of intrusions, allowing the introduction of reactive systems to prevent future problematic situations. These intrusion detection (ID) systems (IDS) are well-known in classic networked environments, but are not easily transferred to the world of WSNs [9] [10], mostly due to the wireless and ad-hoc nature of the networks and the lack of a central point where communication can be monitored.

But WSNs also bring another problem to the table: resources. Due to their vastly deployed numbers and policy to be replaced rather than salvaged, nodes typically need to be cheap. Their limited functional requirements allow for them to be built using simple components without redundancy or excess margins. This introduces an inherent problem for IDS. These systems not only typically require a lot of resources to store detection information and have to execute their algorithms over and over again, but they also want to inspect every single network packet that's passing by. This way they not only would like the node to be powered-on all the time, they also require constant access to the wireless radio, which turns out to be the number one energy-consumer of a node. Introducing ID in WSNs turns out to be a battle for resources.

Finally, besides the technical resources, there is also an economic resource that plays an important part here. Given the same limited functionality, adding a complex piece like an IDS, might raise the production cost well above acceptable levels. The fact that ID algorithms for WSN are currently in a mere state of research, would currently require a developer to read through many research papers, making a selection of algorithms and implement them from scratch. Even if implementations for each of these algorithms would be available, the chance that they are applicable to the target platform is a big *if*. Even simply integrating them can be a complex task.

Our contribution to ID in WSNs is the introduction of a domain specific language (DSL) to formally describe ID algorithms. The DSL is node-oriented with a strong focus on

inter-node communication and aims to allow for the optimisation of execution of multiple ID algorithms, thus lowering the sequential and iterative execution of algorithms and reducing the use of the wireless radio.

Introducing such a formal, platform-independent language to express ID algorithms and accompanying this language with a code generation framework, offers an end-to-end solution to many of the problems introduced above. Research output would be directly applicable in a development environment and a code generation framework allows developers to simply select algorithms and have platform-specific code without any effort. Due to the possibility to optimize the use of resources, the impact of introducing an IDS can be lowered allowing more algorithms to be added, thus augmenting the barriers, which can help to reduce the number of undetected intrusions.

The remainder of this paper proceeds as follows: section II introduces research regarding ID in WSNs. Section III describes the problem space in detail. Section IV introduces our proposed overall solution, further detailed in sections V and VI, which respectively present our domain specific language, FOO-lang, as well as a prototype implementation of a code generator framework to complement the language. Section VII evaluates the prototype implementation and determines if the theoretical merits of introducing FOO-lang, are actually viable. Our conclusions and proposed topics for future research are presented in section VIII.

II. Related Work

When securing any network one tries to prevent intrusions. This should be the primary concern, but not all intrusions can be prevented and we need to fall back on detecting intrusions. In this section we look at recent contributions in this field.

Intrusions of computer networks can take many forms. In the case of WSNs this classification needs to be extended even further. In [8] a good overview of both is presented, showing clearly that WSNs suffer from their wireless and broadcasting nature. This causes many attacks based on e.g. eavesdropping to be nearly undetectable while they are happening. Only when the actual intrusions, based on the covertly collected information, have happened, the intrusion could possibly be detected based on the after-effects. So we can't cover the entire spectrum and need to focus on what is feasible.

Research into ID in WSNs typically focuses on two major topics that complement the architecture of WSNs: the nodes as single entities and the network as a group of such nodes. Both are important because the group cannot make a decision without members that detect malicious behaviour and a group-based decision is often needed because nodes can easily miss out on certain events that could indicate intrusions, due to their wireless and not-always-on nature.

A. Detecting Intrusions

There are different ways to construct a taxonomy for intrusion detection, but common themes do appear. According to [11] and [12] there are three major categories: anomaly detection, signature or misuse detection, and specification-based detection. In [13] the authors mostly agree with this topology, but add the notion of hybrid intrusion detection systems and cross layer intrusion detection systems as recent advances in research. Although these additional categories offer very interesting prospects, the authors have to admit that the impact on the resource-constrained sensor nodes might still be too high.

The meshed network topology of WSNs and its routing protocols have produced many related attacks and each of these attacks has been the focus of many research papers presenting algorithms to detect them. In the following paragraphs we take a look at some popular ones.

In a wormhole attack, an artificial low-latency link is created between two distant nodes in the network. This causes the routing protocol to divert more traffic through this link, offering the attacker an abundance of packets to inspect. In [14] an algorithm is proposed to allow nodes to determine if a wormhole is actively present. In essence, the algorithm relies on the exchange of neighbour lists to allow nodes to determine if unexpected substructures are present in the graph representing the network.

Another attack that is related to the wormhole is the sinkhole. When launching a sinkhole attack, as described in [15], the attacker first needs to compromise an existing node. Being in control of the node, the attacker can make his captured node look more *attractive* to the surrounding nodes and thus draw more traffic to itself. The sinkhole attack is typically a foundation for other routing level attacks, such as selective forwarding, modification of packets or even dropping them altogether. In its simplest form, the sinkhole offers the attacker with lots of packets to analyze and use to gather information about the network and its functionality.

Detecting sinkholes is very hard and is often only possible based on other attacks that are supported by the sinkhole. When the sinkhole is used to implement selective forwarding, [16] proposes an algorithm that allows a base station, the aggregating master node to which all nodes typically send their information, to observe missing data from nodes in the same area in a statistical way.

More typical attacks such as flooding, the sybil attack, rushing ... are amongst others presented in [17] and [10].

B. Cooperative Decision Making

As in many other situations, a group is often stronger than the sum of its individual components. This is surely the case for WSNs. Because not all nodes are always actively participating in the network, some nodes might miss clues that would otherwise lead them to detect intrusions. It is hardly impossible for a single node to detect a complex attack by itself. Therefore a second important research topic consists of cooperative algorithms to combine information from single nodes into a group-based decision about alleged intrusions.

In [18], the authors first present a theoretical foundation to analyze cooperative algorithms. Given these foundations they continue to present an algorithm consisting of 5 phases.

It essentially implements a voting system based on the Guy Fawkes protocol [19] to identify an intruder. It enables the exchange of suspected intruder information and allows distributed authentication of those *votes*. Based on these authenticated votes, a distributed decision can be made about the commonly identified intruder.

A distributed, cooperative algorithm can sometimes be implemented locally. This is illustrated by the reputation-based detection algorithm introduced in [20]. Using this algorithm, nodes can exchange information about the reputation of other nodes. Combining this information allows them to decide about their trust in a given node, based on more than their own, often partial, observations.

C. Software Attestation

A research topic that exists in parallel to the detection and cooperation duality is software attestation. Software attestation offers algorithms for exchanging information about the software that is running on a node, and aims to identify nodes that can no longer prove that their content is unaltered. Examples of evolving algorithms to implement this functionality include SWATT [21], ICE/SCUBA [22] and SAKE [23].

Software attestation is hard and many details can cause an algorithm to fail. Interesting is the discussion surrounding this topic that was started by [24], in response to the previously mentioned papers. In [25] the authors of the original papers counter many of the objections made to their work, but the general feeling is that even in the best conditions, there is always a way to circumvent even the most ingenious algorithm to perform software-based attestation.

III. Problem Analysis

The problem of introducing ID in WSNs is much broader than simply the implementation of a software component. It starts at the very foundations of WSNs. In contrast with for example SNORT [26], the de facto standard with respect to IDS in classic networks, there is currently no community based collection of algorithms that can be implemented. The reason is mostly due to the lack of a central/external location where all network traffic can be analyzed out-of-band. It's not possible to create a single entity that will monitor the entire WSN as is the case in a classic wired network. Therefore, all algorithms typically evolve independently, without any cohesion.

Even more, all of these algorithms lack a common, formal notation. In the case of SNORT, a detection language is provided and new signatures are added on a regular basis, creating an ever growing rule base, capable of detecting even the newest attacks out there.

If implementers of a WSN want to add ID to their network, and therefore to the nodes in the network, they are facing no small task: they need to gather research papers, from which they need to extract the proposed algorithms. Even if the papers would come with an implementation of their algorithm, the chance that the implementation matches the target platform of the newly created WSN is slim.

Simply implementing the different algorithms in sequence, or reusing existing implementations if they would be available, is also no valid option. All of these algorithms typically perform the same actions: first they analyze each incoming packet and collect information about nodes. Secondly, at given intervals, they iterate all known nodes, checking the aggregated information about them and making decisions about the trust to put in them. The algorithms typically also exchange information with other nodes to complement their own findings.

If such algorithms would simply be combined in a sequential way, the resulting code would be far from optimal and would consume many of the resources of the node it runs on: the repetitive parsing of the received packets would result in much longer processing times than needed, while the chattiness of the inter-node communication would cause the wireless radio to be on more often than wanted.

The cost to implement multiple algorithms over and over again, due to the need to optimize the code's organisation, would soon outweigh the available budget of any WSN implementation project. If not, the risk to make mistakes due to misinterpreting the often complex algorithms or the lack of flexibility to change sets of detection algorithms on a regular basis, or simply add a new algorithm in an easy way, are all very valid reasons to not go down this road. Then, how can we add some form of an IDS to WSNs?

IV. Proposed Solution

Different approaches are possible. A major contribution would consist in the creation of a software framework that accommodates some of the typical usage patterns: first, a generic payload parser could be envisaged, allowing algorithms to hook in using callbacks, thus making sure that the parsing is only done once and not repeated for each algorithm. Secondly, the framework could collect all outgoing messages and combine them in a single outgoing packet at the end of a cycle of the node's event loop.

Such a framework comes with guidelines on how to use it, but still relies on good behaviour of its users. Also, the integration of the algorithms with this framework would still be manual work and reconfiguration would typically impact this part of the implementation. Finally, although such a framework does support the implementation, it still requires thorough analysis of research material and extraction of the relevant algorithms.

In the case of SNORT [26], or any other classic network IDS for that matter, the central detection engine accepts formal descriptions of signatures of attack patterns. Because it's not possible to create such a central component once, there has been no urge to describe detection algorithms in a formal way. By introducing code generation techniques to convert a formal, platform-independent description of detection algorithms, and generating this central component, we can bridge the remaining gap.

Code generation allows for the creation of code that implements the actual algorithm. The algorithm can be described in a platform-independent way, therefore allowing reuse of

the algorithm on multiple platforms. The generated code can further access a minimal software framework that covers common tasks and/or platform specific functionality.

Given formal descriptions of the algorithms, an implementer of a new WSN could simply select a set of algorithms, feed their formal description to the code generator and obtain platform-specific code, optimized for execution and communication.

A formal description of ID algorithms would not only be beneficiary to implementers, but also to researchers. Formal descriptions of the algorithms allow for automated generation of code, but the same descriptions can also be loaded into simulators and be investigated in combination with other algorithms [27]. Another interesting option would be that of formal analysis of the algorithms. Further, describing the algorithms in a platform-independent way, would increase their usefulness and allow for more complex algorithms to be abstracted.

V. Introducing FOO-lang

The central component of our proposed solution is a formal description of intrusion detection algorithms. This formal description can be realized using a domain specific language (DSL). Before actually introducing a new language, it is important to ensure the reasons to do so are justified. Too often languages are implemented too quickly, where coding conventions or frameworks with well thought-out APIs fit the bill quite nicely.

The line can not be drawn in a black and white fashion and DSLs do have benefits, but don't come for free [27]. Many good reasons why to implement a DSL, or not, can be found in [27], [28] and [29]. Based on these, we try to justify our choice to propose a DSL for this solution.

A. Justification of the Use of a DSL

The primary goal for the proposed solution is to avoid the creation of inefficient code that drains the limited resources of a node. Two basic examples are presented to illustrate this unwanted behaviour: repetitive execution of the same tasks, such as parsing or iterating a set of known nodes and the use of the wireless radio.

The latter can be covered by a framework, as illustrated before. It comes with an obligation to actually use the framework. The former is a bit trickier. Imagine using the C language as a lingua franca to complement the framework, that hides the platform specific parts. This could be valid, but offering a complete language would soon result in loops that break with the guidelines and would undermine the conceptual idea of optimizing the organisation of the functionality.

Restricting the formal description to a subset that doesn't allow for constructions that violate the goals, is in this case a valid reason to tend towards a DSL. Introducing a DSL further also allows for more functional ways of handling the concepts that are part of the domain. In this case the domain clearly consists of interactions between nodes and local processing of algorithms that typically deal with sets of nodes. Although that C comes close to a lingua franca amongst both researchers and implementers, it is still a low-level language with very little support for concise operations such as pattern matching, event-driven-ness ....

If C doesn't fit the profile, maybe another language does. During the early days of our research, we looked at a language that seemed to fit the required functionality like a glove: Erlang [30]. It comes with strong communication paradigms, has pattern matching, is concurrent and event-driven by nature ...and according to [31] it can even be translated to C. This road looked very promising, but as with any other general purpose language, it proved to be too expressive and would allow for constructions that would thwart the envisaged optimizations.

Our final conclusion is that a real domain-functionality inspired DSL, that permits researchers to express the detection algorithms without overhead and without risk of introducing ill behaviour, is the way to move forward.

B. Design Principles for FOO-lang

Designing a language is an exciting task, but at the same time comes with great responsibility. To paraphrase Albert Einstein: it is important to make the language "as simple as possible, but not simpler". The language should come with enough support to write concise code, but not become obfuscated. The language constructs it offers should be applicable in generic combinations with each other and be especially transparent in use. On the other hand, it is also important that the language restricts the user without limiting his expressiveness or making certain expressions unintuitive.

Starting from this last requirement, one of the most important things that should not be available are *loops*. If we want to control the way lists of data are handled, we need control over the loops targeting that data. In case of this specific domain, data is centralized around *nodes*, so loops are typically targeting *nodes*. To meet this restriction but still offer the user a way to functionally handle nodes, scheduling and events are introduced. Instead of explicitly constructing a loop, functionality can be scheduled or attached to certain events that happen on the collection of nodes. This way, the intended loop is actually abstracted to its functional meaning and can be handled and combined with other execution strategies.

This is the actual core of the language and it is also where its name derived from: Function Organization Optimization.

A second important feature of the language is that it should feel natural. Luckily, dealing with embedded systems and WSNs, both researchers and implementers share a common language. Most development on these systems is done using C, and both parties know this language well. We therefore chose to mimic C for a lot of the general syntax.

FOO-lang also borrows basic syntax from object-oriented (OO) languages, in that respect that it bears the concept of methods that can be called upon *objects*. The language doesn't provide any means to define or instantiate objects though. The *nodes* domain controls the availability of these objects,

creating them and providing them to the functions that are defined.

To avoid typical constructions that require loops, the concept of pattern matching is introduced. A pattern of non-variables and variables can be provided to functions that support them. If such a function can match the non-variable part of the pattern, the variables in the pattern take on the corresponding values of the data against which the match was performed. It is obvious that this will typically be used to analyze incoming data when communicating between nodes. List literals are often used in conjunction with this functionality and allow to combine different (non-)variables; these are therefore also available in FOO-lang.

To allow complex code to be introduced, FOO-lang provides a way to import external functionality. This feature is provided through a statement that allows to define the prototype of the imported function, which can then be used as any other function.

One final important feature is type inference. FOO-lang tries to limit the need for typing information. Based on declaration and usage, most types can be inferred. In doing so, the language tries to focus on the functionality and not on the technical implementation. This mainly targets the researchers who now can focus on the bare essence of their algorithms. Also, typing can be platform-dependent and we want the descriptions to be platform-independent.

C. The Actual Language

In the previous paragraphs the design principles of FOO-lang were introduced. In this section we briefly introduce some of the syntax of FOO-lang using an example implementation.

Based on the algorithms that were introduced in many of the consulted papers, a common structure for these algorithms can be detected: (1) when new data is received the algorithm wants to process it, (2) at regular intervals the algorithm wants to validate the aggregated information on nodes and (3) the algorithm might want to communicate with other nodes to exchange information. Given this structure, we now introduce the *hello world* example for FOO-lang: *heartbeat*.

It can be considered the most elementary ID algorithm possible, still containing all aspects of a generic algorithm. It deals with availability: as long as we receive regular updates from a node, we trust it. When it fails to produce a *heartbeat*, we no longer trust it. The resulting FOO-lang code is presented in listing 1.

Following the design principles, most of this example code should be easy to understand. FOO-lang tries to be “readable”, especially to users with a C background and common knowledge about other languages and programming paradigms. Most language constructs are borrowed from other languages, like annotations using the ‘@’ symbol or atoms with the ‘#’ prefix.

Defining a language is one thing. It should of course also be possible to generate code from it that actually addresses the problems we identified before. In case of a language, the real proof is in building an actual code generator.

```
// each algorithm is contained in a single module
module heartbeat

// basic constants, types are inferred
const heartbeat_interval     = 3000    // 3 sec
const validation_interval    = 5000
const max_fail_count         =    3

// imports external functionality
// requires type information
from crypto import sha1(byte*)             : byte[20]
from crypto import sha1_cmp(byte*, byte*)  : boolean
from time   import now()                   : timestamp

// nodes can be extended with module specific data
extend nodes with {
  sequence   : byte      = 0
  last_seen  : timestamp = now()
  fail_count : byte      = 0
  trust      : boolean   = true
}

// event handler when receiving data
after nodes receive
do function(me, sender, from, hop, to, payload) {
  case payload {   // payload is a list of bytes
    // an atom and three variables following it
    contains( [ #heartbeat, time:timestamp, sequence,
                signature:byte[20] ] )
    {
      // skip our own or from untrusted
      if(from == me or !from.trust) { return }

      // validate signature
      if(sha1_cmp(signature, [sequence, time])) {
        from.last_seen = time
        from.sequence  = sequence
      } else {
        from.fail_count++
      }
    }
  }
}

// validate nodes using function at a given interval
@every(validation_interval)
with nodes do function(node) {
  // check time that passed since the last heartbeat
  if(now() - node.last_seen > max_last_seen_interval) {
    // the heartbeat is late, track this incident
    node.fail_count++
  }

  // check number of failures doesn't exceed our limit
  if( node.fail_count > max_fail_count ) {
    node.trust = false
  }
}

// function to construct and broadcast a heartbeat
function broadcast_heartbeat(node) {
  time = now()
  nodes.broadcast([ #heartbeat, node.sequence, time,
                    sha1([node.sequence,time]) ])
  node.sequence++
}

// broadcast a heartbeat at every heartbeat interval
@every(heartbeat_interval)
with nodes.self do broadcast_heartbeat
```

Listing 1: Example implementation of a *heartbeat*.

VI. Prototype Code Generator

We have built a prototype code generator for FOO-lang using Python and ANTLR [32]. The implementation tries to be as generic as possible and is based on transformations using the visitor pattern of abstract syntax trees (AST) and a semantic model [29].

The FOO-lang code is parsed using a parser generated by the ANTLR tool. This produces an AST, which in its turn is transformed into a semantic model using a visitor. The semantic model represents the actual meaning of the implementation. From a project point of view, the DSL is merely a textual representation of the semantic model and the parser and first visitor allow us to easily populate the model.

The initial model is an exact representation of the FOO-lang code that was parsed. This means for example that not all types are well defined. A second visitor, now targeting the semantic model, is used to check all types and tries to infer those that are still marked *unknown*.

With all types known, the model is semantically complete and we can move to the next phase: construction of a code model. A code model can actually be considered an AST that can be persisted as code again. The initial code model that is constructed from the semantic model typically contains all the language features available in FOO-lang. The construction is done partly by the generic generator and partly by the *nodes* domain.

The code generator aims to be language and platform independent. Both language and platform are exchangeable plug-ins to the generator. With the initial code model based on the full set of language constructs of FOO-lang, it is clear that the transformations that need to be performed by these plug-ins are the migration of features that are not literally available in the target language or platform.

The resulting code model is an AST that can be emitted in the desired language and conforming to the target platform.

VII. Evaluation

To evaluate the prototype we chose to use a system based on the Atmel ATMEGA1284p micro-controller [33] and the Digi XBee S2 Zigbee module [34]. To drive the hardware a minimalistic library was used, wrapping the technical calls to the components in more easy to use functions.

The reason for this at first sight unusual setup is simple: relying on basic hardware and software allows us to validate that the generator is capable of generating code even for the most basic environment available. Any step up, both in hardware or software, would offer better and higher abstractions that would make it easier to generate code for that situation. It shows that the requirements of the generator towards the platform are minimal, don't rely on advanced frameworks or operating systems and can therefore be applied in any environment, targeting any platform.

For the evaluation we constructed a small WSN consisting of three nodes: an end-device, a router and a coordinator. The end-device and the router were completely autonomous nodes, while the coordinator consisted of an XBee module, directly connected to a computer via a USB break-out board, allowing serial access via a terminal.

We started from a basic application that measures light intensity and reports it to the coordinator. The application was written both manually and generated. On top of this baseline we added two intrusion detection related algorithms: a heartbeat, that allows nodes to validate each other's continuous presence, and a reputation-building algorithm that checks if a parent node is cooperative and actually forwards messages with a destination further down the network[20].

Three criteria were evaluated: the image size of the resulting compiled code, the network usage in number of frames and bytes and the time required to perform once cycle of the event loop. These metrics were collected in four situations: without ID algorithms, with a heartbeat, with reputation-tracking and with both algorithms implemented.

In case of the manual implementation, both algorithms were constructed as standalone modules and sequentially called from the base-application's event loop. The code generator was provided with FOO-lang descriptions of the algorithms and generated all four cases.

Before reviewing the results, it is very important to put these results in a proper perspective. Comparing manually written code to generated code is no exact science. One can always argue that the quality of the two code bases is not comparable and that the manual code always can be improved.

Still it might be interesting to look at the numbers, given that both code bases are constructed with the same intentions and practices. We believe we have achieved this and have taken honest decisions while constructing the implementations, making them comparable. Numerical analysis of the implementation offers us a preliminary estimation of the effect of our proposed solution.

corollaryTables I and II show the collected data respectively for the manual and the generated implementation. The base case is presented in absolute values, while the implementations with the added algorithms are presented as relative values.

	base	heartbeat	reputation	both
size (bytes)	10500	148%	127%	175%
frames	20	255%	160%	315%
bytes	476	406%	181%	487%
time (μs)	48	196%	183%	310%

TABLE I: Results for the manual implementation.

	base	heartbeat	reputation	both
size (bytes)	10496	175%	156%	200%
frames	20	245%	160%	275%
bytes	476	399%	186%	454%
time (μs)	48	252%	252%	288%

TABLE II: Results for the generated implementation.

From these raw results, we learn that adding algorithms manually to the base-application, results in a cumulative increase. The impact of both algorithms is simply the sum of the impact of each algorithm by itself. In the case of the time required

to execute one cycle of the event loop, the total time is even a bit higher than simply the sum.
In the case of the generated implementation, this no longer holds: although that even the individual increases due to adding a single algorithm are higher than in the manual case, the result for the implementation of both algorithms is less than the sum of both. Here we clearly see the effect of reusing the framework that comes with FOO-lang. The same goes for all other metrics. Maybe most remarkable is the effect on the time of one event loop cycle: both algorithms by itself add 150% with respect to the base case, but when combined, the impact hardly increases. Here we learn that the introduced framework is responsible for the initial overhead, but it clearly pays for itself when adding more algorithms.
Table III compares the two situations by subtracting the manual case from the generated case, showing the impact of the code generation. A relative value for the entire implementation with both algorithms is also presented.

	base	heartbeat	reputation	both	result
size (bytes)	-4	2822	3070	2664	115%
frames	0	-2	0	-8	87%
bytes	0	-36	24	-156	93%
time (μs)	0	27	33	-11	93%

TABLE III: Comparison of both implementations.

When comparing the results of both implementations we first can conclude that the generator produces exactly the same code for the base case. We couldn't measure any real differences between both implementations.
Secondly, we see that the framework that comes with the generated code adds to the size of the resulting image. But the cost is almost constant and is relatively lower when algorithms are combined. With roughly 3KB of additional overhead, or 15% in this simple case, this impact is affordable.
From a functional point of view, the generated code lives up to the expectactions: both the number of frames as the bytes sent benefit from well organized code and reuse of a common framework.
Finally we see that both algorithms by itself add an equal amount of time to the processing of a single event loop cycle, but when combined, the event loop is about 7% faster compared to the manual case.
Without any effort to optimize the framework code, nor leveraging knowledge about the specific algorithms, this very basic generated code shows that the proposed principles are not only theoretically feasible but actually result in interesting improvements.

VIII. Conclusions and Future Work

WSN nodes typically respond to events in their environment or perform their tasks at clearly scheduled intervals. Applying a language that has these concepts at its core, allows focus on the bare essence and reuse of best practices at the generation level. The rather limited functional domain of ID in WSNs proofs to be a wonderful candidate for applying code generation techniques.
The ideas proposed in this paper and the prototype are foundations. They show that code generation techniques can be used to combine different, independent algorithms in an automated way, while offering better resource usage than simply sequentially combining the algorithms.
These preliminary steps open up a broad range of possible tracks to explore. On one hand our goal is to collect as many relevant research papers as possible and extract the algorithms from them, implementing them using FOO-lang. This will undoubtedly lead to extensions to the language and its core libraries. On the code generation front itself, the generator has to be lifted from the prototype level up to production quality. Next, many optimizations can be added to it: functional code inlining, continuation-passing style code structuring ...
On a higher level, other challenges are also prominently present: having formal, platform-independent descriptions of algorithms, allows for simulation and thorough analysis of their behaviour, both by themselves and in combinations. Finding ways to combine as many algorithms as possible in a single generated solution will enable implementers of WSNs to add a decent level of intrusion detection to their solutions. Another interesting track is that of policy-based generation and deployment. Different nodes in the network might benefit from different configurations of intrusion algorithms. When trying to detect a combination of selective forwarding and a sinkhole attack [16], this algorithm should only be deployed on coordinators or base stations, not on end-nodes.
Finally we hope that this paper might act as a call-for-participation by all ID in WSNs researchers and will lead to an ever growing collection of formal descriptions available to those that need them.

Acknowledgements

We would like to thank the reviewers for their thoughtful and helpful comments that enhanced the readability of this paper. Our gratitude and respect also goes out to all members of the WSN work group at KU Leuven for creating the nurturing environment where these ideas could grow.

References

[1] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *Internet Computing, IEEE*, vol. 10, no. 2, pp. 18–25, 2006.
[2] D. Hughes, P. Greenwood, G. Coulson, and G. Blair, "Gridstix: Supporting flood prediction using embedded hardware and next generation grid middleware," in *Proceedings of the 2006 international Symposium on on World of Wireless, Mobile and Multimedia Networks*. IEEE Computer Society, 2006, pp. 621–626.
[3] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: towards cooperation frameworks for open innovation," in *The future internet*. Springer, 2011, pp. 431–446.
[4] M. Chan, D. Estève, C. Escriba, and E. Campo, "A review of smart homes—present state and future challenges," *Computer methods and programs in biomedicine*, vol. 91, no. 1, pp. 55–81, 2008.
[5] J. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, and A. Wood, "Wireless sensor networks for in-home healthcare: Potential and challenges," in *High confidence medical device software and systems (HCMDSS) workshop*, 2005, pp. 2–3.
[6] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[7] A. Becher, Z. Benenson, and M. Dornseif, *Tampering with motes: Real-world physical attacks on wireless sensor networks*. Springer, 2006.

[8] D. G. Padmavathi, M. Shanmugapriya *et al.*, "A survey of attacks, security mechanisms and challenges in wireless sensor networks," *arXiv preprint arXiv:0909.0576*, 2009.

[9] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM, 2000, pp. 275–283.

[10] D. Djenouri, L. Khelladi, and N. Badache, "A survey of security issues in mobile ad hoc networks," *IEEE communications surveys*, vol. 7, no. 4, 2005.

[11] A. Mishra, K. Nadkarni, and A. Patcha, "Intrusion detection in wireless ad hoc networks," *Wireless Communications, IEEE*, vol. 11, no. 1, pp. 48–60, 2004.

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

[13] N. A. Alrajeh, S. Khan, and B. Shams, "Intrusion detection systems in wireless sensor networks: A review," *International Journal of Distributed Sensor Networks*, vol. 2013, 2013.

[14] R. Maheshwari, J. Gao, and S. R. Das, "Detecting wormhole attacks in wireless networks using connectivity information," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 107–115.

[15] I. Krontiris, T. Giannetsos, and T. Dimitriou, "Launching a sinkhole attack in wireless sensor networks; the intruder side," in *Networking and Communications, 2008. WIMOB'08. IEEE International Conference on Wireless and Mobile Computing,*. IEEE, 2008, pp. 526–531.

[16] E. C. Ngai, J. Liu, and M. R. Lyu, "On the intruder detection for sinkhole attack in wireless sensor networks," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 8. IEEE, 2006, pp. 3383–3389.

[17] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, 2002.

[18] I. Krontiris, Z. Benenson, T. Giannetsos, F. C. Freiling, and T. Dimitriou, "Cooperative intrusion detection in wireless sensor networks," in *Wireless Sensor Networks*. Springer, 2009, pp. 263–278.

[19] R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham, "A new family of authentication protocols," *ACM SIGOPS Operating Systems Review*, vol. 32, no. 4, pp. 9–20, 1998.

[20] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 3, p. 15, 2008.

[21] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 272–282.

[22] 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*. ACM, 2006, pp. 85–94.

[23] 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.

[24] 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*. ACM, 2009, pp. 400–409.

[25] A. Perrig and L. Van Doorn, "Refutation of on the difficulty of software-based attestation of embedded devices," *Based on the Paper Castelluccia C, Francillon A*, 2010.

[26] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, 1999, pp. 229–238.

[27] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.

[28] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography." *Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[29] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[30] J. Armstrong, R. Virding, C. Wikström, and M. Williams, "Concurrent programming in erlang," 1993.

[31] G. Wong, "Compiling erlang via c," *Software Engineering Research Centre Technical Information, Melbourne, Australia*, 1998.

[32] Terrence Parr, "Antlr3 website," http://www.antlr3.org.

[33] Atmel Corporation, "Datasheet for ATMEGA1284p," http://www.atmel.com/images/doc8059.pdf, 2009.

[34] Digi International, "Manual for XBee," http://ftp1.digi.com/support/documentation/90000976_P.pdf, 2013.