In-Network-Aggregation as Case Study for a Support Tool Reducing the Complexity of Designing Secure Wireless Sensor Networks

Steffen Peter , Krzysztof Piotrowski and Peter Langendörfer
IHP GmbH, Im Technologiepark 25, 15236 Frankfurt(Oder), Germany
Email: {peter,piotrowski,langendoerfer}@ihp-microelectronics.com

***Abstract*—This paper shows how complex security-related design decisions in wireless sensor networks can be made less difficult with a proposed supporting tool. As case study in this paper we focus on in-network-aggregation which is a promising option to reduce network effort. We introduce several algorithms for concealed data aggregation, each with its own benefits and drawbacks concerning security issues but also with respect to code size, processing overhead etc. Selecting the optimal combination requires in-depth knowledge of programming resource constrained devices, protocols for those devices, and last but not least significant background in security.**

Our proposed configuration tool – named configKIT – has been designed to cope with such complexity. This paper presents how configKIT works, how it will be set up, and how it can be applied in practice. The pre-compiler assessment process considers memory, energy and security parameters and provides reliable application-dependent configurations to the developer before a single line of code is written. The approach can reduce development time significantly and enables even complex and sophisticated security algorithms to a broader public.

Keywords: **Configuration, Wireless Sensor Networks, Security, In-Network Aggregation**

I. Introduction/Motivation

Configuration and development of wireless sensor networks (WSN) in practice is a challenging, time-consuming, and error prone task. Programming such severely constraint devices requires to respect many different aspects, for example code size, processing overhead and energy consumption. If additionally security aspects and effects of module interactions have to be considered, the task becomes so complex even experts make mistakes. In best case such errors merely increase the development time, for example if after compilation the program is too large for the applied hardware. In worst case the development error will compromise security and other properties unnoticed over the lifetime of the network.

For software development for personal computers several tools are available that reduce development time while respecting complex dependencies like security and interoperability. For example model driven [1] tools like VEST [2] support the development with effective composition, configuration, and the associated dependency analysis, even checking security properties. However such tools do not respect the constraint properties of WSNs. Dedicated tools for WSNs [3], [4] usually propose own programming methologies. They help to improve the implementation process of WSNs but do not directly allow reusing existing solutions, what clearly reduces the design space but also disables many specialized state-of-the-art solutions.

A promising approach that enables flexibility and high performance is the usage of small specialized modules and to connect them directly without overhead structure or inter-module communication. The idea has been realized in nesC/TinyOS [5], which we are also using in our project. It merges all required components to one block what promises to be very memory-efficient and results in reduced computational overhead. Beside the problem of keeping an overview of a large set of small modules the major issues of such programs are development and maintenance costs. The substitution of one module may affect many other modules. It is particularly the case with respect to security properties. An alternation of a security profile that eventually exchanges several security protocols and modules would correspond to a completely new development of the system.

We have developed a tool named configKIT that promises to cope with the complexity of implementing WSNs. The tool proposes suitable configuration, i.e. units of functional modules, based on brief inputs of application requirements. Therefore it uses a repository that contains the hardware information of the nodes and a database of software modules and properties.

This paper will study how configKIT can help mastering the complexity of the particular case in-network aggregation (INA). Therefore we briefly introduce INA and applicable INA-algorithms in Section II and present implementation results in Section III. Section IV introduces the configuration tool and Section V demonstrates how the database will be set up with the INA-implementation results. Finally Section VI demonstrates how other developers can use configKIT and the embedded data to reduce their development time while easily coping with the complexity of the different algorithms.

II. In-Network Aggregation

A major application scenario for WSNs is monitoring environmental data that is transmitted to a central point (sink). An obvious solution is to send all packets from the sensors to the sink. Since sensors can be outside the direct radio range intermediate nodes forward the messages through the network. This straightforward solution can be easily implemented but

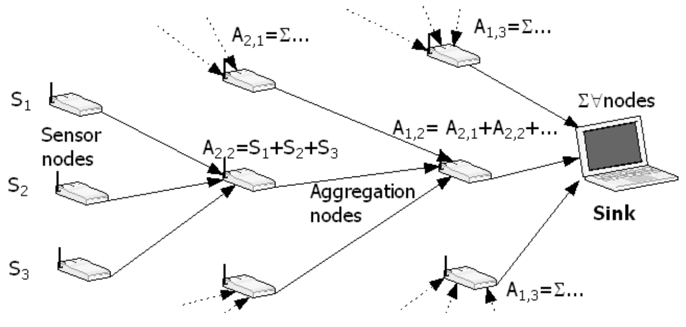


Fig. 1. Principle of in-network aggregation: Sensors send their values to aggregator nodes that combine the data to one data packet

would need a lot of energy for the transmission and forwarding of the packets. Since sending one bit consumes the same amount of energy as executing 50 to 150 instructions on sensor nodes [6], omitting as much network traffic as possible is a substantial task in the area of designing WSN applications.

A well known approach that reduces the energy required for packet transmission in the sensor-to-sink scenario is the in-network aggregation (INA). In many scenarios the sink does not need the exact values for all sensors but a derivative such as sum, average, or deviation. The idea of the INA is to aggregate the data required for the determination of the derivatives as close to the source as possible, instead of transmitting all sensed values through the entire network. Figure 1 shows the data gathering sensor nodes S_1,S_2,S_3 sending their data to the intermediate aggregator node $A_{2,2}$ which aggregates the three values and sends one instead of forwarding three packets. The next aggregator node ($A_{1,2}$) combines further values that are finally sent to the sink.

A serious issue connected with the INA is the security of the data. If the data is encrypted on the nodes with classic methods, it can only decrypted by the sink, what would disable the feature of INA. Waiving end-to-end (ETE) secrecy the data could be encrypted hop-by-hop (HBH). An approach that promises the combination of ETE-security and INA is the concealed data aggregation (CDA). CDA allows aggregating encrypted values on the intermediate nodes without decryption. The following subsections present applicable approaches of secure INA for WSNs.

A. Hop-by-Hop Encryption

Hop-by-hop encrypted aggregation encrypts the data sent over the wireless link so that potential eavesdroppers cannot extract the actual data. Each intermediate node decrypts the data so that it can be processed, i.e. aggregated. Finally the node encrypts the result and sends the encrypted data to the next hop. The HBH approach has the drawback that the data must be decrypted and re-encrypted on every aggregation node which is clearly unfavorable from the energy consumption point of view. Another critical disadvantage is the missing feature of end-to-end protection. It poses a risk if intermediate nodes can be accessed or compromised, which is a sensible attack scenario for WSNs.

However, using standard cryptographic libraries such as tinySec [7] the HBH-INA approach easily enables secrecy over the wireless link while reducing the transmitted number of packets.

B. Concealed Data Aggregation

Concealed Data Aggregation (CDA) is an improved version of the INA, which, in contrast to HBH, ensures the ETE-privacy, i.e. the encrypted values do not need to be decrypted for the aggregation. Instead, the aggregation is performed on encrypted values and solely the sink can decrypt the result.

The fundamental idea of CDA are privacy homomorphism functions that have the property: $enc(a+b) = enc(a) \oplus enc(b)$, i.e. an operation on two encrypted values has the same result as the encryption of the sum of the two unencrypted values.

Though it is not the purpose of this paper to extensively explain and discuss the algorithms of CDA approaches, the next few subsections introduce three schemes for WSNs, since it will help to follow the property assessments later in the paper. A more extensive discussion can be found in [8].

1) CaMyTs: A key stream based CDA approach was proposed in [9] by Castelluccia, Mykletun, and Tsudik. The idea is to perform a modular addition of a classic stream cipher and with the sensed data. Each sensor uses a different pseudo random stream cipher. For encryption, the plaintext is simply added to the current key of the stream modulo the length of the key space. The sink has to subtract the corresponding key stream to obtain the plaintext again. CaMyTs is provable secure and well suited for the application on WSNs since only one modular addition is required for encryption and aggregation. A problem is the decryption, which requires exactly the same key streams of each individual sensor node. Processing all the streams is not only a computational problem, but the sink that decrypts the aggregated values must also stay synchronized with each key stream and it has to know which sensor data is part of the aggregate.

2) Domingo-Ferrer: In [10] Domingo-Ferrer introduced a symmetric PH (DF) scheme that also has been proposed as efficient CDA system for WSNs in [11]. It is a symmetric algorithm that requires the same secret key for encryption and decryption. The aggregation is performed with a key that can be publicly known. It is required that the same secret key is applied on every node in the network. Though it is known that the algorithm has security flaws, it can be considered as applicable for most scenarios in sensor networks as long as long term secrecy is not important. Since the plaintext will be split in several segments the ciphertext is slightly larger than the plaintext.

3) Hybrid: In [12] a hybrid (hCDA) approach has been proposed that combines two existing CDA schemes to a combined one. It executes the both embedded schemes successive while obtaining the additive privacy homomorphic property. The idea was motivated by the observation that all schemes have security flaws, that could be overcome with a combination of two schemes. In particular the combination of CaMyTs and

TABLE I
OVERVIEW OF THE FOOTPRINT OF THE IMPLEMENTED AGGREGATION APPROACHES IN THE 85-NODES-NETWORK

Alg.	Node	Size		Traffic		Duty Cycles
		ROM [byte]	RAM [byte]	Packets [number]	Total Size [byte]	per epoch per Node
No INA No Crypt	Sensor	12468	271	214	4922	109980
	Aggregator	11586	321			230584
	Base	11632	273			319460
INA H2H Encr.	Sensor	15120	468	106	2438	95980
	Aggregator	15694	548			78325
	Base	14548	500			62778
INA without encrytion	Sensor	12490	271	106	2438	94914
	Aggregator	11834	321			74016
	Base	11632	273			57993
CaMyTs	Sensor	15460	347	106	2438	115717
	Aggregator	14608	373			71934
	Base	14794	1311			1124207
DF	Sensor	13749	332	106	2862	95443
	Aggregator	12942	374			84287
	Base	12972	354			71108
Hybrid CaMyTs and DF	Sensor	17070	464	106	2862	115906
	Aggregator	15994	478			84270
	Base	16274	1428			1135265

DF seems to be promising. With the improved protection also the effort for en/decryption clearly increases because the two embedded schemes result in two encryptions that have to be performed on the nodes.

III. CDA Implementation Results

This section describes the steps that are required to implement the INA schemes applying standard methods. The implementation results will deliver the input required for setting up the configKIT database in Section V.

For practical comparison we wrote a small application that does nothing but sending stored value from the sensor nodes to a sink. We simulated a network with one sink, four aggregator notes as first level aggregators, 16 aggregator nodes as second level, and 64 data-delivering sensor nodes. So the initial situation can be stated: We have network and task and we are looking for the best algorithm that forwards the aggregated values from the sensor nodes to the sink. We are not interested in the precise sensor readings but in an average value, i.e. the sum of the single readings and the number of values.

We implemented the application in TinyOS. For the network we used the standard TinyOS ActiveMessage modules, and additionally the Timer and the LEDs of the simulated sensor node. The application is similar for all tested cases. The only difference is the applied INA-algorithm.

The following INA approaches have been investigated:

- No INA (our code)
- Hop-by-Hop (StreamCipher taken from [7])
- INA without encryption (our code)
- CaMyTs (code from original authors [9])
- DF (code from authors of [11])
- Hybrid CDA (combining code written by us)

Since the source codes are from different authors we already had the first problem harmonizing the interfaces. For all but CaMyTs it was sufficient to slightly adapt the interfaces. For CaMyTs we wrote a wrapper.

The code was compiled for micaz [13] nodes and simulated with Avrora version 1.7.105 [14]. Avrora is also the source the estimated packet count and sizes and the CPU duty cycles.

Table I shows the results for each role and algorithm. The first two rows show the memory consumption (ROM and RAM). It is not surprising that the no in-network-aggregation is the smallest and the hybrid has the largest code size. It was also expected that the base-node of CaMyTs and hCDA (that contains CaMyTs) requires the most dynamic memory, because it has to store each sensor's private key.

It is a bit surprising that CaMyTs is larger than DF, despite the algorithm should not be significantly more complicated. It is apparently caused by the more flexible CMT implementation and the currently still required wrapper. However, the actual reason – and here we initially made a wrong assumption – is that CaMyTs needs a StreamCipher that requires significant memory and energy.

The number of packets in our simulation is as expected. These results could have been computed without actual implementation. Though the network is not severely large, the packet count for the no-INA is already the double of all INA approaches. Packets of DF and hybrid are slightly larger than packets for the other three CDA algorithms because DF (that is also part of hCDA) has a larger ciphertext. This table (as well as the current implementation) does not consider potential packet loss or not-responding nodes. It can be expected that it would increase the total packet size for the CaMyTs based simulations.

The probably most interesting numbers are the computation cycles per epoch. The micaz has roughly 8 million cycles per second. Most approaches have a utilization of about 1 percent. The duty time includes everything the CPU has to perform, i.e. beside the cryptographic operations, also receiving and sending of packets. The actual energy consumption required for radio, board, memory are not considered here. The No-INA approach needs much more computation efforts than expected.

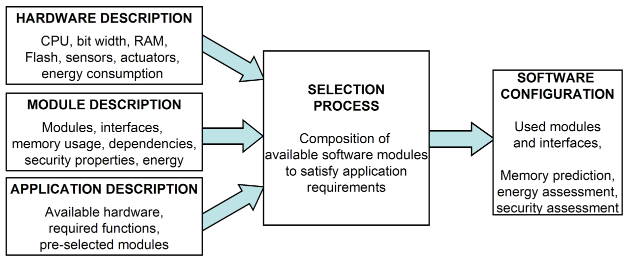

HARDWARE DESCRIPTION
CPU, bit width, RAM, Flash, sensors, actuators, energy consumption
MODULE DESCRIPTION
Modules, interfaces, memory usage, dependencies, security properties, energy
APPLICATION DESCRIPTION
Available hardware, required functions, pre-selected modules
SELECTION PROCESS
Composition of available software modules to satisfy application requirements
SOFTWARE CONFIGURATION
Used modules and interfaces,
Memory prediction, energy assessment, security assessment

Fig. 2. Structure of configKIT

The CPU is very busy treating packets in the network stack. Though for the sensor nodes the most packets do not reach the application layer receiving and discarding them needs a lot of additional energy. While the additional energy for aggregator and base node was expected because they have to handle the additional packets the extend has been surprising.

The base nodes handling the CaMyTs algorithm have the highest load. It is caused by the required forward key computation for each connected sensor node. This symmetric key computation is also the reason why the sensors handling CaMyTs need more cycles than the other algorithms.

The results show that all implementations have reasonable memory- and energy consumption. The simulation has also shown that the implementations are correctly working. Thus, the decision of the most suitable algorithm for a particular application can be primarily made based on the security attributes discussed as discussed in [8].

Now that we have the table of implementation results and the knowledge of the security properties we easily can assemble the INA in practice. In Section V we show how the data can be entered into the database of configKIT so that other developers have the chance to exploit the results conveniently.

IV. Configuration Toolkit

The introduction of the different CDA algorithms and their different properties clearly shows the complexity of applying sophisticated functions in WSN. In a complete application the selection of the INA-scheme would merely be one step. Selecting the proper key-distribution-scheme and a transport/network protocol are other apparent decisions that have to be made. An actually secure CDA approach can turn worthless if the keys are distributed without secrecy, or if the slightest fluctuations in the network path can destroy all results.

In particular for tinyOS an abundance of great tools and implementations is available. However, even for experts it is hardly possible to keep an overview over all modules, their properties, requirement and side-effects. This is why we have developed the configuration supporting tool 'configKIT' that on one hand offers the enormous set of state-of-the-art approaches and implementations while on the other hand provides good accessibility that eventually concludes in an accelerated development process.

A. Architecture

The general notion of configKIT is shown as Figure 2. Based on the three inputs hardware description, module description, and application description a module selection algorithm computes configurations that:

- can be combined to one software image running on the intended node (compatibility of interfaces)
- are compatible to the available hardware (compatibility to hardware)
- fulfill the requirements semantical and functional requirements of the application
- fulfill the security requirements

The software configuration contains suitable modules and a prediction of memory and energy consumptions, and an assessment of the security properties. The result will not be one prototypical application framework that is slightly tailored for each purpose, but a newly computed composition of modules with each configuration process.

B. Inputs to the Selection Process

The decisions of the security compiler are based on the following data:

1) Hardware Description: The Device Information incorporates data of hardware and operating system of the sensor nodes. Processor type, bit width, memory size and indeed sensors which are available or supported. Extended properties are available battery power, life time but also quality or classification of sensor readings. All these data is stored in a device database. A user can pick the hardware that is available or define a new hardware and store it in the database. The footprint (i.e. memory, CPU) of the particular device are the constraints the software selection process has to respect.

2) Module Description: The Module Description is a second database in the configKIT repository. It incorporates all available software modules, their dependencies, requirements and properties. It also includes the modules' footprint, like required memory but also the security degree provided by the specific module. It is a network of interfaces and implementations. Modules are connected to modules, they use, by interfaces – just like in tinyOS. Interfaces can be provided by more than one module, what reflects the design options the developer has. The module description will be described in more detail in Section V.

3) Application Description: The Application Description is the third element of the security compiler. Virtually it points on the hardware, modules and requirements in the module description. It provides the initial conditions for the compiler. The description can be very precise or quite abstract. So a developer can chose a particular algorithm, but it is also possible to ask for a routing that has good security features.

The actual application module is also a module that usually uses the other modules. Consequently it is included in the Module Description.

C. Selection Process

The selection algorithm composes the proper configurations that satisfy the requirements of the application description using the software repository and hardware properties. The selection process is the 'brain' of configKIT. Briefly, it:

- builds a dependency graph based on available modules,
- selects the modules that are explicitly required by the application description,
- starts a backtrack algorithm that looks for all sets of modules that satisfy the application needs,
- filters the configurations that satisfy the security and energy requirements.

The algorithm traverses through the graph and 'tries' each possible implementation-alternative for each required interface until either the requirements are fulfilled or no modules can be found that match a dependency. Though the algorithm theoretically is NP-complete, optimizations and the natural structure of dependencies help to boil down the practical efforts. Currently our database contains 60 modules and the run time is less than 0.1 sec on a Pentium processor.

V. Setting up configKIT for INA

In this section we explain the steps required for setting up configKIT so that at the end it is aware of the properties of the different approaches. First we have to set up the interfaces, then add the modules, then determine and set the properties of the modules so that finally we can add an application that uses the available interfaces.

A. Adding the interfaces

The actual process of adding interfaces to the database is quite straightforward. We select the name of the new interface and set whether the interface is an actually implemented interface or a virtual interface. Virtual interfaces can be considered as an abstract class as kind of mother class for a set of specific interfaces. In case of INA, we define INA as the abstract interface. Later this abstract interface can be addressed by an application or another module that wants to use INA. It will be task of the selection algorithm to find and to assign suitable implementations.

Then we add the actual interfaces: Hop-by-Hop, CDA and hCDA (hybrid CDA). The dedicated specialized interfaces are needed if an application explicitly uses one interface. For example if the application developer wants the application to use Hop-by-Hop then he/she connects the application module with the HbH-interface. Then the other INA-approaches will never be considered as option for this module. The additional classification of interfaces also improves the understandability and maintenance and helps addressing (sub-)sets of interfaces. It implicitly allows a level of fuzziness because applications can pick either direct modules (e.g. HbH), small sets (e.g. CDA), or general classes (e.g. INA).

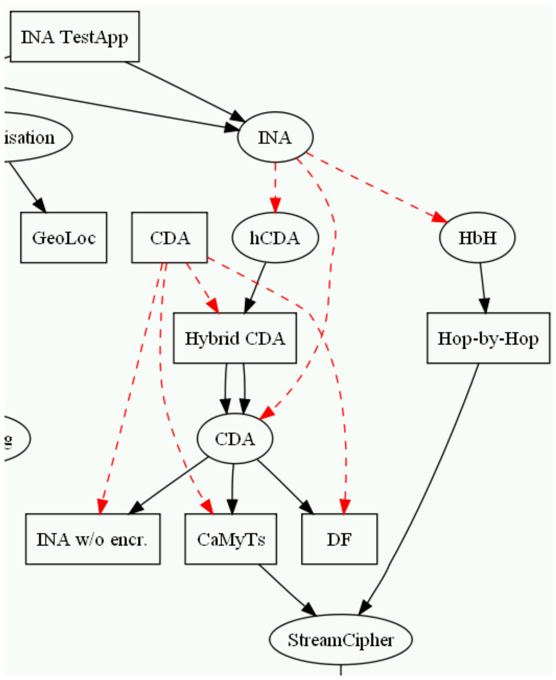


Fig. 3. Detail of a configKIT dependency graph: boxes are modules, ellipses are interfaces, black arrows are use/provide, red arrows are 'is-relations'.

B. Adding the modules

In the second step, the modules will be added to the repository. The modules either provide or use available interfaces. In our INA case we add:

- INA without encryption provides CDA (we consider no encryption as weakest form of encryption)
- Domingo Ferrer (DF) provides CDA and uses a random interface
- CaMyTs provides CDA and uses a StreamCipher
- HbH provides HbH and uses a StreamCipher
- hybrid CDA (hCDA) provides hCDA and uses two CDAs

We presume that the other interfaces (Random and Stream-Cipher) have already been defined.

As for the interfaces we can also define abstract classes that can combine a set of modules in a group. In our case we add the abstract module CDA and assign the specific modules (hybrid CDA, INA w/o encr., CaMyTs, DF). With the abstract module CDA we can later address the set of semantically similar CDA-modules. Finally we have to add the application module. An application is just another module that uses other interfaces. Our simple application that does nothing but sending aggregated values, requires a network interface and the INA-interface. All interfaces and modules mentioned so far can be seen in Figure 3. It is a detail of the screen shot of the configuration tool emphasizing the CDA part.

C. Setting the Properties of the Modules

When the modules and dependencies have been described the properties of the modules can be set. Currently configKIT supports the properties memory consumption, security properties (reliability, secrecy and integrity), and energy assessment.

1) Memory consumption: Estimation of memory consumption is probably the easiest of the properties. One could expect that it is sufficient to measure the size of the single modules and add them up at the end of the assessment process to get the size of the application. But there are several issues that have to be considered. First, memory has to be differentiated between RAM and ROM. In particular RAM is severely constraint on most WSN nodes, while usually sufficient ROM in the flash memory is available.

Another problem is that memory consumption differs with the applied hardware. A module compiled for the mica node will not have the same size as for an MSP430. Compiler optimizations or other optimization-effects caused by the connection of the modules can additionally affect the result. For example code that is not actively used in a specific configuration will be usually not included in the binary image. Another example is a storage module that stores a cryptographic key. If it stores the key for a 256 bit encryption it needs more space than storing a 128 bit key. Thus, the memory usage cannot be precisely determined before the complete image is assembled.

Since compiling every considerable configuration is not feasible we apply the following approach: In the evaluation phase we use the straightforward addition of module sizes. We know that the result is not perfectly precise but is a good estimation that helps to assess the memory consumption. Consequently the memory property is a set of two numbers, i.e. RAM and ROM, that describe the module size.

2) Security Properties: Security properties are not as easy to assess as memory size. The first problem is that there exist nothing like an obvious security metric that can be applied. In our first approach we assigned each module an subjective security degree on a scale from no security to very secure. Though it already helped to filter weak modules the results were not always satisfying. Indeed one number cannot describe the security properties of a complete toolbox. If the application needs high integrity but does not need concealment it is indeed not necessary to embed a high secure stream cipher. This is why it is useful to specify the particular security classes.

At this point of development the security of a module is defined by three numbers, rating the security strength of the properties robustness, secrecy and integrity. The notion of classification allows tackling the security degrees individually. If an applications needs security in terms of robustness and does not care for concealment the expected outcome will be different than vice versa. That classification can either be directly set, or – in case the module uses other security relevant modules – it can be derived from the modules a module uses. The derived assessment is a function that apply the algebraic functions (+,-,MIN,MAX). For example if we assume the security properties for the hybrid CDA are the best of its two sub-modules, then the three security properties of the hybrid CDA would be:

- integrity = MAX($1.integrity, $2.integrity);
- secrecy = MAX($1.secrecy, $2.secrecy);
- robustness = MAX($1.robustness, $2.robustness);

while $1 and $2 point to the first and second connected module

TABLE II
INITIAL INA ENERGY AND SECURITY ASSESSMENT

Alg.	Integrity	Secrecy	Robustness	Energy
No INA	1	0	1	0
H2H Encr.	2	1	1	1
INA w/o encr	1	0	0	3
CaMyTs	3	$1	1	1
DF	1	2	2	2
Hybrid	MAX($1,$2)	MAX($1,$2)	MAX($1,$2)	0

respectively.

CaMyTs directly forwards the secrecy parameter of the used StreamCipher (secrecy = $1.secrecy), so that the secrecy of CaMyTs is determined by the secrecy of the StreamCipher.

The assessment of the other INA modules is a straightforward classification on a subjective metric from 0 (not secure) to 3 (very secure). The classification is based on the results in [8]. Table II shows the assessments.

Please be aware that such a simplistic security assessment does not hold the expertise of an security expert, but it helps evaluating and filtering in the early phase of the development.

3) Energy Consumption: In the current version we treat the energy property similar to the security properties. Though objective metrics for energy are available, e.g. energy or cycles per operation, we still lack an approach to harmonize them so that they can be compared. The INA approaches for sure can be compared by cycles per epoch, as it is done in Table I. However the data could only be used in direct comparisons and would not have any significance in comparison with routing protocols for example.

With our proposed subjective metric, i.e. '0' for bad and '3' for very good, it is easier to combine different protocols, and we still keep the option to compare similar protocols directly. Clearly that approach is not perfectly satisfying, but as the properties for security and memory it can help to filter suitable or not suitable modules.

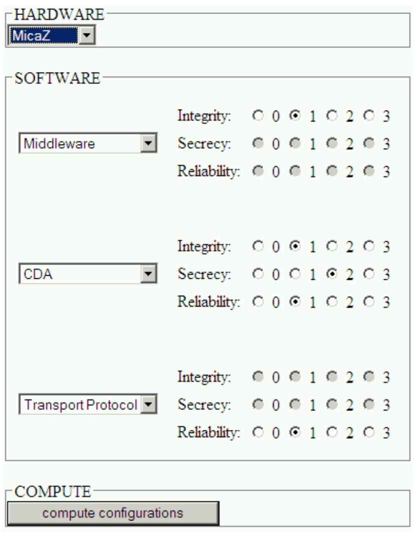
D. Incremental Extension of the database

Up to this point we showed how to set the initial set of modules. Since we added six modules and some interfaces the process of adding modules appears a bit complicated. In practice the complexity is not so high because most of the time just one module will be added per time. Considering we want to add a new CDA algorithm, all what has to be done is to add new interfaces if required and then add the new module with its properties. If the interface is connected to the INA the new module will automatically considered the next time configKIT looks for suitable configurations.

This way we hope other developers can be encouraged to add their modules to the database.

VI. CONFIGURING INA WITH CONFIGKIT

In the previous section we explained how to set up the repository of configKIT.In this section we show how developers – with different level of experience – can use confgKIT and this way can exploit the stored data to easily configure their WSN-application with INA.



HARDWARE

MicaZ

SOFTWARE

Middleware

Integrity: 0 [✓ 1] 2 3

Secrecy: 0 1 2 3

Reliability: 0 1 2 3

CDA

Integrity: 0 [✓ 1] 2 3

Secrecy: 0 1 [✓ 2] 3

Reliability: 0 [✓ 1] 2 3

Transport Protocol

Integrity: 0 1 2 3

Secrecy: 0 1 2 3

Reliability: 0 [✓ 1] 2 3

COMPUTE

compute configurations

Fig. 4. Selection dialog box sets the inputs for the application description

As introduced in Section IV-B3 the application description, which is the input of the selection process, is mainly a set of pointers to modules in the software repository with additional requirements. The selected modules can be both abstract modules or concrete modules. Additionally global requirements like available hardware has to be set.

An exemplary selection dialog can be seen as Figure 4. In that example the user chose the abstract modules Middleware, CDA and a Transport Protocol with medium security requirements. The hardware in the screen shot is a micaz node.

In our simpler CDA example, the user already has written the application top module (INA TestApp in Figure 2) and is looking for suitable modules that can be connected. Without additional requirements the request would be just: 'INA TestApp'. The result can be seen as Figure 5. The network function is ActiveMessage and the preferred INA approach is INA without aggregation. Since no security requirements were given, configKIT selected the smallest solution. The `sec: (0|0|0)` indicates that no security features are provided. The predicted memory consumption is 11 kBytes and the predicted energy consumption is low.

If the secrecy should be increased it is sufficient to change the secrecy radio button from '0' to '2'. Then DF would be the preferred INA approach. CaMyTs and hCDA are alternatives that are also listed, but they are larger and need more energy. The resulting security parameters are 1 for integrity, 2 for secrecy, and 0 for robustness. Robustness is 0 because simple ActiveMessage is not a reliable network protocol. Increasing the robustness requirement would add reliable network or transport protocols.

Increasing the required level of integrity to '2' disqualifies DF as proper solution because its integrity property is too weak. Then CaMyTs with an StreamCipher will be chosen instead. Figure 6 shows the output in case all three security parameters are set to '2', i.e. 'good'. Since CaMyTs does not provide sufficient robustness the only remaining choice is the

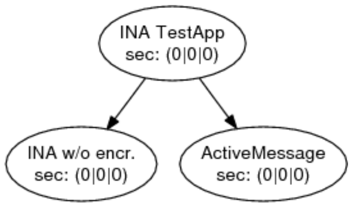


Fig. 5. Initial configuration of the INA-application: Simple network protocol and simple INA algorithm are small but insecure

hybrid CDA which applies both CaMyTs and DF. The need for a robust network protocol added the Transport and network protocols DTSN/DSDV [15]. It can be seen that all security requirements are at least 'good' ('2'). It can also be seen that the complexity of the system is already quite high. The total memory prediction is more than 25 kBytes (4kB RAM), so that it is still acceptable for the micaz node.

All tests so far could be described in one line as part of the application description. The missing configurations of Table I (hybrid without reliable transport protocol and Hop-by-Hop) are apparently no obvious choices. Actually this is reasonable because using HbH is questionable if CDA can be applied and to use a heavy weight CDA without reliable transport is also not sensible. However, if the developer – and we for comparison reason – is looking for the assessment of such configurations, additional inputs in the application description can be set. In both cases it is sufficient to remove the security requirements and to add the specific module to the application.

Table III summarizes the required inputs for each case introduced in Table I. It shows also the predicted total memory consumption and the assessment of energy and security. The selections and proposed configuration based on the inputs are reasonable, and the predicted memory consumption cor-

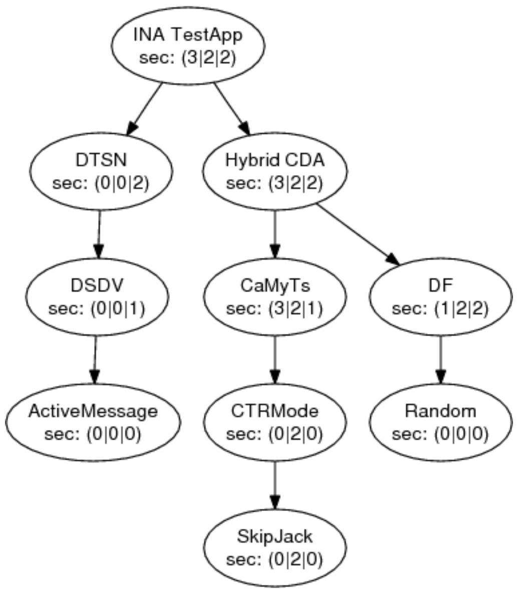


Fig. 6. Output of of the configuration process if all security properties have to be good or better

TABLE III
OVERVIEW OF EXAMPLARY INPUTS, THE RESULTING CONFIGURATIONS AND THEIR ESTIMATED PROPERTIES

Alg.	Inputs	ROM (kB)	Integrity	Secrecy	Robustness	Energy
No INA	no INA TestApp	10	0	1	0	0
H2H Encr.	INA TestApp + HbH	13	1	1	0	2
INA w/o encr	INA TestApp (no sec)	11	0	0	0	3
CaMyTs	INA TestApp (high secrecy)	15	3	3	0	1
DF	INA TestApp (good robustness)	13	1	2	0	2
Hybrid (secure network)	INA TestApp (good robustness and secrecy)	25	3	3	2	0
Hybrid (no secure network)	INA TestApp + hCDA	16	3	3	0	0

responds to the actual memory consumption determined in I. The security assessments are correct considering our inputs, though it can be discussed whether robustness has to be zero whenever a reliable network protocol is missing. The energy evaluation is also correct and it allows direct comparisons, but the significance of that assessment is at least questionable.

VII. CONCLUSIONS

We chose INA in this case study demonstrating our WSN configuration tool configKIT. This scenario combines many aspects a developer of WSNs application has to face, i.e. energy efficiency, memory consumption and levels of security. We demonstrated how configKIT has to be set up in order to help developers getting an overview of the different approaches, and we showed how the data can be applied in practice.

The presented structure of the configKIT repository allows successively extending the database so that eventually a comprehensive consideration of an abundance of modules and approaches is not only feasible but intended.

Still the results do not satisfy in each situation. The estimations of memory usage, energy consumption and security properties are early approaches that are subjects for refinements. However, even the presented methods already help assessing potential configuration very early in the development process. The results are sensible as demonstrated in Section VI. configKIT highlights aspects a developer could easily forget, e.g. that an actually small module depends on a large other module, or that a robust security means does not provide robustness unless a reliable network protocol is added. Experts in the particular area will surely be able to find better solutions, while average developers will get a lot of new possibilities and understanding of complex relations.

Even without a feature like automatic implementation configKIT can reduce development time of application for WSNs. A developer needs few mouse-clicks to define the requirements and less than a second later a configuration is provided that will run on the specified hardware and satisfy all the other needs that have been defined. With the knowledge of the right modules and their interfaces we assume that the required adaptation of the actual implementation can be accomplished much faster and with less errors.

ACKNOWLEDGMENT

The work presented in this article was supported by the European Commission within the STReP UbiSec&Sens of the EU Framework Program 6 for Research and Development (IST-2004-2.4.3). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the UbiSec&Sens project (http://www.ist-ubisecsens.org) or the European Commission.

REFERENCES

[1] A. Brown, J. Conallen, and D. Tropeano, *Models, Modeling, and Model-Driven Architecture (MDA)*. Springer Verlag, 2005, ch. Introduction:.

[2] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis, “Vest: An aspect-based composition tool for real-time systems,” in *Proceedings of the IEEE Real-time Applications Symposium*, 2003.

[3] L. Luo, T. F. Abdelzaher, T. He, and J. A. Stankovic, “Envirosuite: An environmentally immersive programming framework for sensor networks,” *Trans. on Embedded Computing Sys.*, vol. 5, no. 3, 2006.

[4] M. J. Ocean, A. Bestavros, and A. J. Kfoury, “snbench: programming and virtualization framework for distributed multitasking sensor networks,” in *VEE ’06: Proceedings of the 2nd international conference on Virtual execution environments*, 2006, pp. 89–99.

[5] J. Hill, P. Levis, S. Madden, A. Woo, J. Polastre, C. Whitehouse, R. Szewczyk, C. Sharp, D. Gay, M. Welsh, D. Culler, and E. Brewer, “TinyOS: http://www.tinyos.net,” December 2005.

[6] K. Piotrowski, P. Langendoerfer, and S. Peter, “How public key cryptography influences wireless sensor node lifetime,” in *Proceedings of the 4rd ACM Workshop on Security of ad hoc and Sensor Networks, SASN*, 2006.

[7] C. Karlof, N. Sastry, and D. Wagner, “Tinysec: A link layer security architecture for wireless sensor networks,” in *Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[8] S. Peter, D. Westhoff, and C. Castelluccia, “A survey on the encryption of convergecast-traffic with in-network processing,” *IEEE Transactions on Dependable and Secure Computing*, 2008, preprint available at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.23.

[9] C. Castelluccia, E. Mykletun, and G. Tsudik, “Effcient aggregation of encrypted data in wireless sensor networks.” in *MobiQuitous*. IEEE Computer Society, 2005, pp. 109–117.

[10] J. Domingo-Ferrer, “A provably secure additive and multiplicative privacy homomorphism,” in *ISC ’02: Proceedings of the 5th International Conference on Information Security*, 2002.

[11] J. Girao, D. Westhoff, and M. Schneider, “Cda: Concealed data aggregation for reverse multicast traffic in wireless sensor networks,” in *IEEE International Conference on Communications*, 2005.

[12] S. Peter, P. Langendoerfer, and K. Piotrowski, “On concealed data aggregation for wireless sensor networks,” in *CCNC ’07: Proceedings of Fourth IEEE Consumer Communications and Networking Conference*, 2007.

[13] C. T. Inc., *MPR / MIB User’s Manual*, 2005, available at: http://www.xbow.com /Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf.

[14] B. L. Titzer, D. K. Lee, and J. Palsberg, “Avrora: scalable sensor network simulation with precise timing,” in *IPSN ’05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.

[15] B. Marchi, A. Grilo, and M. S. Nunes, “Dtsn: Distributed transport for sensor networks.” in *Symposium on Computers and Communications*, 2007.