


Article

Simulation-Based Performance Validation of Homomorphic Encryption Algorithms in the Internet of Things

Johannes Kölsch *, Christopher Heinz, Axel Ratzke and Christoph Grimm 

Design of Cyber-Physical Systems, TU Kaiserslautern, 67663 Kaiserslautern, Germany; heinz@cs.uni-kl.de (C.H.); ratzke@cs.uni-kl.de (A.R.); grimm@cs.uni-kl.de (C.G.)

* Correspondence: koelsch@cs.uni-kl.de

Received: 30 August 2019; Accepted: 18 October 2019; Published: 22 October 2019



Abstract: IoT systems consist of Hardware/Software systems (e.g., sensors) that are embedded in a physical world, networked and that interact with complex software platforms. The validation of such systems is a challenge and currently mostly done by prototypes. This paper presents the virtual environment for simulation, emulation and validation of an IoT platform and its semantic model in real life scenarios. It is based on a decentralized, bottom up approach that offers interoperability of IoT devices and the value-added services they want to use across different domains. The framework is demonstrated by a comprehensive case study. The example consists of the complete IoT “Smart Energy” use case with focus on data privacy by homomorphic encryption. The performance of the network is compared while using partially homomorphic encryption, fully homomorphic encryption and no encryption at all. As a major result, we found that our framework is capable of simulating big IoT networks and the overhead introduced by homomorphic encryption is feasible for VICINITY.

Keywords: discrete event simulation; IoT; homomorphic encryption; smart grid; validation

1. Introduction

Nowadays, the number of applications in which IoT networks are deployed grows rapidly. These infrastructures operate in different domains such as smart cities, energy, eHealth, and transportation and behave as isolated islands in the global IoT ecosystem [1]. Their secure interconnection is a big challenge that still needs to be resolved. One promising approach towards interoperability of IoT networks across different domains is proposed by the VICINITY project. The project is supported by European Union Horizon 2020 program with the duration of four years (January 2016–December 2019) and the consortium of 15 partners from seven different countries. In this paper, we present a simulation framework to validate the security of a smart energy use case, which is enabled by using homomorphic encryption. The performance of the overall IoT ecosystem is evaluated with and without using the additional homomorphic encryption layer. The real micro-service for the encryption is integrated into the simulation as hardware in the loop with the approach presented in [2].

1.1. The VICINITY Project

The goal of the VICINITY project is to develop a platform that connects isolated IoT infrastructures into one global ecosystem called virtual neighborhood where users can select to which other systems their smart objects should be connected in a peer-to-peer network. The platform automatically supports interoperability from technical up to semantic level. The semantic interoperability of smart objects coming from different operators and using different standards is enabled with the semantic model integrated into the VICINITY platform [3].

The VICINITY architecture offering interoperability “as a service” is shown in Figure 1. It is based on a decentralized, bottom-up and cross-domain approach that resembles a social network, where users can configure their setups, integrate standards according to the services they want to use and fully control their desired level of privacy in a P2P (peer-to-peer) network.

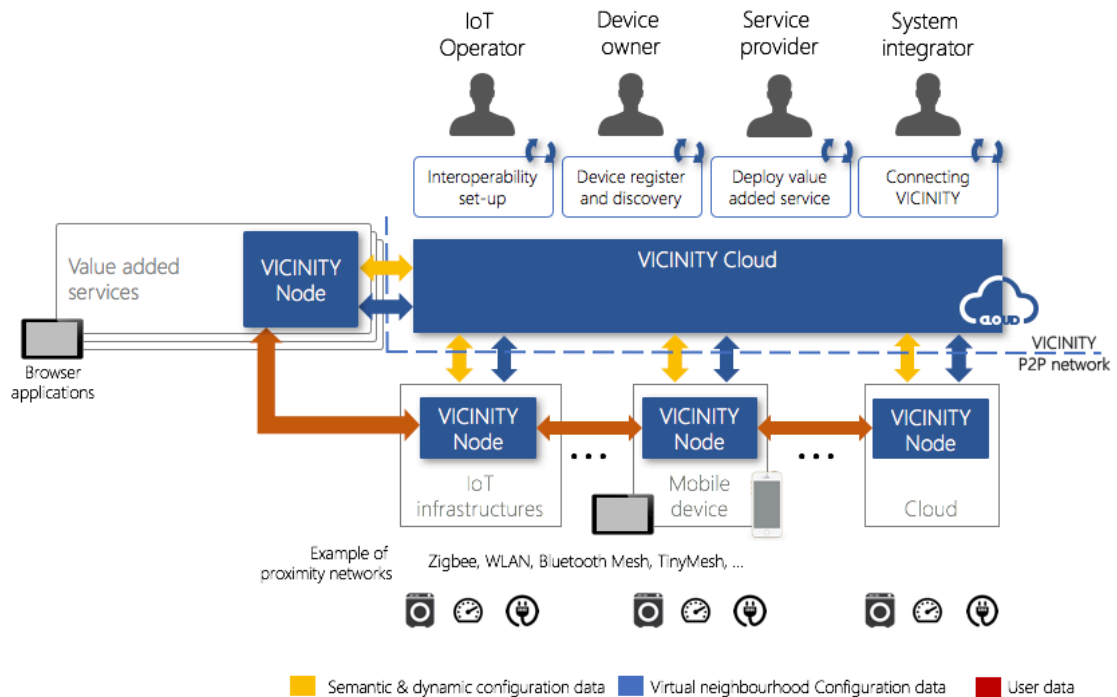


Figure 1. High-level VICINITY architecture [4].

At the end of the project, VICINITY’s approach will be demonstrated on four pilot sites coming from the following different domains: energy, building automation, health, and transport. VICINITY’s potential to create new, cross-domain services will be demonstrated by value added services, such as micro-trading of demand-side management capabilities, AI-driven optimization of smart urban districts and business intelligence over IoT.

The focus of the work presented in this paper is to build a virtual environment for simulation and validation of IoT infrastructures and their use cases in real life scenarios before their real deployment. Here, a smart energy use case at one of the VICINITY pilot sites, located in Tromsø, Norway, is used for demonstration.

1.2. Homomorphic Encryption

VICINITY’s architecture offers privacy built-in by design, as only metadata on the connected devices is stored in a central cloud. Sensitive information, e.g., sensor readings, are transmitted peer-to-peer only from the data producer to its intended consumer. Additionally, this needs to be approved by the data owner in advance and on an individual basis. However, once the data owner has given his consent, his data are given out and will be available to potentially malicious third parties, which may seem trustworthy at first glance. As Value-Added Services (VAS) are the key to making the whole IoT “smart”, a user may be tempted to share his data with such a service, even though he would rather not give out his information. An even better approach would be to never give away any sensitive data (in clear text), yet still allowing the VAS to work. What may sound contradicting is exactly what is possible with the use of homomorphic encryption (HE) Schemes. Homomorphic encryption enables certain calculations, or, in the case of fully homomorphic encryption, any arbitrary function to be executed on encrypted ciphertexts without the need to decrypt this data first. Partially homomorphic encryption has been subject to research for quite some time, only enabling a limited

number of operations to be executed. With the introduction of a fully homomorphic encryption Scheme by Craig Gentry in 2009 [5,6], any arbitrary computation is now possible on ciphertexts with no need to decrypt and giving out any cleartext information at all. However, fully homomorphic encryption is more expensive in terms of computational demand, so people need to decide on a trade-off between performance and versatility. As we demonstrate in Section 5.5, using our Framework enables us to evaluate different options in a realistic but controlled environment.

2. State of the Art

2.1. Overview of IoT Simulators

The size of the IoT market today is growing at enormous speed and will continue to do so. The number of connected devices has already exceeded the worlds human population [7]. In such complex IoT networks, simulation plays an important role for the early-phase validation before their deployments in the real-life world.

2.1.1. S^3 and OMNeT++

In [8], Feretti and D'Angelo proposed the concept of Smart Shires and stated the importance of simulation during development, in particular as means to validate the architecture before the deployment of a prototype. As requirements for their simulation tool of choice, they determined scalability and (almost) real-time capabilities. Furthermore, they stated that a multi-level simulation is required in most cases, since running the whole model on highest degree of detail would prove itself infeasible. Instead, they favored the approach of combining different simulators together, each one specializing on the domain to be simulated. More specifically, for the concrete case of simulating their smart shire concept, they aimed to combine a discrete event simulation engine with an agent based model. Subsequently, Feretti and D'Angelo developed the Smart Shire Simulator (S^3), based on the GAIA/ARTIS middleware. ARTIS is a framework, designed to enable seamless sequential, parallel, and distributed execution of large-scale simulations with a collection of different communication algorithms, such as TCP/IP, MPI or shared memory. It provides synchronization algorithms for pessimistic or optimistic approaches. The GAIA (Generic Adaptive Interaction Architecture) (<http://pads.cs.unibo.it/doku.php?id=pads:gaia>) framework is used to simplify the simulation of scenarios in a parallel and distributed fashion by a high-level application programming interface, thereby reducing simulation time through strategies based on adaptive partitioning of the model.

Subsequently, they subjected S^3 to a performance evaluation in order to verify their aforementioned claims. The exact hardware setup for the tests can be inspected in [8]. The tests took place in a bidimensional toroidal space without any obstacles. The simulation space contains a given number of simulated entities, of which some follow the random way point mobility model, while others remain static. For their experiments, the number of simulated entities was in the range of 1000–32,000, of which 50% were mobile nodes. The speed of the random way point was uniform in the range of 1–14 space units/time step and the sleep time was set to zero. The interaction range between the simulated entities was set to 250 space units and the forwarding range to a value greater than 200 space units, while the density of nodes was set to one node every 10,000 space units². The simulations were run for 900 time units, where one time unit equals one time step within the simulation. The time to live (TTL) of messages was set to four hops and the dissemination probability to 0.6. In a first test with only one CPU core and the Wall clock time as measure, the approach based on a sequential simulator proved itself as not adequate due to scalability constraints. The next tests focused on a parallel setup. In the range of moderate load (1000–8000 SEs), no speedup could be gained with only two cores, due to communication overhead. Within the range of heavy load, more cores could increase the speedup, but the performance gain was not as high as expected by the authors. They assume the reason for that is the nature of the used model itself with only small amounts of computation per simulated entity, but a large number of communications between the simulated entities. The last

experiments were conducted to evaluate a possible performance gain in the parallel setup with enabled adaptive partitioning. It showed that there was always a speedup compared to the static partitioned model.

In [9], D'Angelo, Feretti and Ghini expanded their approach of simulating sole "Smart Shires" to the simulation of the Internet Of Things (i.e., the combination of one or several Smart Shires with the surrounding urban areas). For the purpose of simulating "Smart Shires" in conjunction with neighboring smart cities, they propose a more elaborated version of a multi-level simulation. Their general approach is a multi-level simulation in which a coarse detailed top-level simulation engine coordinates a set of domain specific simulators.

A special point of interest for them is the interoperability between these different simulators and, of course, the inter-model operations. Subsequently, such a simulation is built for the use case of simulating a smart shire: The implementation comprises two levels. The top level is a time stepped agent based simulator. This Level 0 simulator describes the whole smart territory at a basic level where different entities produce wares, publish or subscribe to a multitude of different services, and move freely through the environment. Some of those entities are static while others follow specific mobility models. To ensure adequate parallel and distributed simulation capabilities, the authors used the S^3 again. All the aforementioned entities are equipped with wireless devices. The interaction between them is based on the Priority-based Broadcast (PbB) Protocol. To decrease computational effort, the message forwarding is based on a minimum distance between the corresponding entities. Additionally, message caching is implemented. The level below is implemented using OMNeT++ in version 4.4.1 with the INET extension in version 2.3.0.

The OMNeT++ models are realized as grids of fixed nodes as market sellers in the context of the smart shire scenario. Additionally, there exist N mobile nodes, which are injected into the model by the Level 0 simulator. These mobile nodes represent pedestrians that are equipped with WiFi and move with walking speed. The two simulation levels communicate through a TCP/IP connection that is created during the time; the lower level simulation runs. Through this connection, the top level simulator can send its continue simulation and end simulation messages while the parameters and results for the respective simulation are exchanged through a directory system on a hard drive. Since the Level 0 simulator is time stepped and OMNeT++ is a discrete event simulator, communication between the levels can only occur at the end of a Level 0 time step. Generally, D'Angelo, Feretti and Ghini stated that, in the case of all simulators being time stepped, synchronization between the levels is eased when the top level time step is simply a multitude of lower level time steps. If none of the employed simulators were time stepped, special synchronization points would have to be introduced.

To evaluate the efficiency of their proposed approach, the authors conducted a series of experiments. First, they examined the two simulators individually in terms of scalability. The hardware setup for the experiments was the same as in [8]. In addition, the parameters for the experiments with S^3 remained the same as in [8], with the addition that message caching had been introduced. Generally, the test results also remained the same as in [8]. However, it turned out that the newly introduced message caching had no positive effect on performance gain, but the opposite. Additionally, D'Angelo, Feretti and Ghini took a look at the peak virtual memory usage and the peak resident set size. However, they state clearly, that S^3 was designed with focus on execution speed and not on effective memory usage. As a result, the authors found the memory usage to be acceptable. Finally, they conducted a series of experiments on the simulators dynamic partitioning capabilities and concluded that, with this technique, there was always a speedup to measure in comparison to static partitioning. However, this speedup was dependent on properly partitioning the model. Next, they evaluated the performance of OMNeT++. Each simulation run was the length of a single time step of the Level 0 simulation. The results indicated that increasing the number of simulated entities significantly increased the memory usage and the wall clock execution time of the simulator. From this, the authors concluded that OMNeT++ was adequate for small scale experiments, but not for large scale Internet of Things scenarios. Finally, they examined the multi-level approach. First, a sequential S^3 simulation run on

Level 0 was considered. The simulation contained 1000 simulated entities and comprised one logical process. From there, one single simulated entity was transferred to Level 1 and managed by OMNeT++ for the period of one Level 1 time step. This experiment showed that the wall clock time increased linearly with the number of lower level simulators created. However, the costs added by spawning new OMNeT++ instances lay within the range that was expected by the authors and according to them, the overhead caused by the coordination between simulators, had proven as negligible. On the other hand, it also showed that a higher number of simulated entities in the higher level reduced the wall clock time overhead, which occurred from creating lower level simulations. This is due to the fact that the upper level has to wait for the lower level to finish. With more simulated entities in the upper level, the upper level can simulate them while waiting for the lower level simulation to finish. As a consequence, the lower level simulators proved themselves as bottleneck for the multi-level simulation that the upper level had to wait for. Secondly, the authors conducted experiments with more than one logical process within the Level 0 simulation. These tests showed that, within the Level 0 simulation, the logical processes that spawned lower level simulations became the bottleneck of the system. However, the so generated coordination overhead is dependent on the number of simulated entities in such a logical process. Lastly, the authors stated that their experiments showed that multiple spawns of lower level simulators could consume huge amounts of memory; therefore, they suggested that a distributed architecture is advantageous.

In [10], D'Angelo, Feretti and Ghini extended their approach to focus a Parallel and Distributed Simulation (PADS) to achieve performance gain. To this end, they chose a hybrid simulation approach coupled with PADS since they found out conventional monolithic simulators are not scalable enough. This approach should circumvent the problems arising with the number of entities in a large scale IoT simulation which had proven itself as a bottleneck. It is of importance that, in this approach, the use of a hybrid simulator does not change the total number of simulated nodes in contrast to the approach presented in this paper. As stated in [9], the design of inter-model interactions and the interoperability between the simulators within a multi-level approach have to be done carefully. In addition, the transfer of objects and data between the involved simulators can turn out to be a hard task. Furthermore, the authors stated that it is of course in the realm of possibility that the hybrid simulation introduces additional approximation error in analysis. Before presenting their implementation of a case study, they specified the following general problems arising in parallel and distributed computing [10]:

- The need of methodologies and tools to define of conceptual models
- The availability of simulation paradigms that are easy to use
- Well-defined interfaces for the interoperability among simulations
- nNw approaches for the composability of simulators and the automatic mechanisms for the deployment and management of simulators on distributed infrastructure

The aforementioned case study is a so-called *Smart Market* scenario, again in the vicinity of a smart territory. The hybrid simulation approach to this is an extension of the multi-level simulation from [9]: The top level simulator that advances the simulation as a whole is again implemented by S^3 , this time with a more refined broadcasting approach, called geoPbB. Level 1 is realized by two simulators this time, which work in succession. The first of the two is an ADVISOR (ADvanced VehIcle SimulatOR) based simulator. ADVISOR in turn, is based on Matlab/Simulink and developed for the analysis of performance and fuel economy of vehicles [11]. The simulator uses a component based approach in which the components are modeled by equations and quasi steady approximations. It is utilized by the upper level in batch mode to simulate vehicles arriving at the neighborhood of the market place and the parking activity of these customers and reports the emissions back to it. The results of this simulation are then used to provide the second simulator on this level with the number of novel customers entering the market place. This second simulator on the level is again OMNeT++ in the same configuration as in [9]. The different simulators were employed on different operating systems: The top level simulator S^3 , as well as the lower level simulator OMNeT++ were executed in a

Linux environment while the ADVISOR based simulator was executed on Windows. Communication between the levels again was message based, implemented through TCP socket connections. Since both simulators on Level 1 run consecutively, both are encapsulated in a wrapper that executes them successively and handles the synchronization with Level 0.

After a series of experiments focused on the individual performance of the simulators S^3 and OMNeT++ on the same hardware setup as in [8] before, the authors conducted a series of tests with only S^3 and OMNeT++ on a single Linux host. It proved the assumption: if more than one logical process in Level 0 creates multiple Level 1 instances, the memory consumption can increase vastly. The authors recommended the partition of Level 1 in multiple interconnected hosts. Subsequently, D'Angelo, Feretti and Ghini continued testing with a parallel and distributed approach. While Level 0 (S^3) and Level 1b (OMNeT++) reside on a Linux host as described above, Level 1a (ADVISOR) is created on a windows host whose detailed specifications can be reviewed in [10]. The two hosts are interconnected by fast ethernet LAN. When the number of simulated entities, which were transferred to Level 1, was limited to one, experiments showed relevant costs arising due to the sequential execution of Level 1a and Level 1b. This could possibly be balanced through a parallel setup for Level 0. However, a possible speedup would be rather low, due to the nature of the used model, which lacks embarrassingly parallel problems. Finally, tests regarding the partition of simulated entities in more and more logical processes showed a linear increase of overhead and the addition of logical processes would only cause memory thrashing.

The three thus far presented works used multi-level simulation to model rough movements and services on the top-level by agent-based modeling and finer grained movement and communication on the lower levels by discrete event simulation and other simulation approaches. This stands in contrast to this work that only uses discrete event simulation techniques. In addition, the approach proposed in this work does not necessarily rely on single domain-specific simulators for specific levels, since in every level multiple simulators can manage their respective models.

2.1.2. ACOSO and OMNeT++

Another hybrid simulation approach was proposed by Fortino et al. [12–14]. The basic concept throughout all three works is to concentrate on the thing aspect of the Internet of Things, which in turn means focusing on the Smart Objects that constitute the Internet of Things. This focus on a smart object based IoT comprehension enabled the authors to apply principles of the agent-based computing paradigm to the problem of simulating complex IoT systems. By tightly coupling smart objects with agents, the IoT ecosystem can be treated as a multi agent system. Consequently, agents running in different cooperating smart objects form a decentralized multi agent system and this way maximize the interoperability among the heterogeneous subsystems and distributed resources [13]. This facilitates the system modeling and development, increases the scalability and robustness, and, at the same time, reduces the design time and time to market [13]. Exploiting this modeling technique, Fortino et al. used the agent based simulator ACOSO (Agent-Based COoperating Smart Object) to model the smart objects involved in a specific scenario. Subsequently, they used OMNeT++ in conjunction with INET to simulate the communication between these smart objects. In the following experiments, the authors focused on general observations of character of communication between smart objects in different sized deployment scenarios, rather than performance of the proposed approach to IoT simulation.

Again, this approach uses agent-based modeling in contrast to the approach in this paper. However, it is worth mentioning that OMNeT++ is used in a similar fashion as in this work to model the detailed communication between simulated entities.

2.1.3. MAMMotH

While the majority of related work thus far focused on simulation, in [15], Öooga, Ou, Deng and Ylö-Jääski instead aimed to emulate massive scale Internet of Things scenarios. The reason for their focus on emulation instead of simulation is justified by the lack of simplification, which simulation

scenarios resort to, and the fact that the correct timing ensured by network simulators may leave problems of message timing uncovered. First, they did a survey of 15 different Internet of Things simulators and emulators NS2 [16], NS3 [17], PDNS [18], GTNetS [19], J-Sim [20], Jist [21], COOJA [22], TOSSIM [23], DSSimulator [24], GlomoSIM [25], OMNeT++ [26], SensorSIM [27], SENSE [28], EMULAB [29] and ATEMU [30]. According to the authors, the survey showed that, at that time, current solutions were mostly appropriate for small- and medium-scale emulation of Internet of Things scenarios but not feasible for large-scale testing with millions of nodes. Again, the scalability seemed to be the main problem for the evaluated simulators and emulators. As a solution, they proposed MAMMoTH (A massive-scale emulation platform for internet of things), a large-scale IoT emulator. Their goal was to emulate tens of thousands of nodes in a single VM, with a future goal of up to 20 million nodes. In addition, it was aimed to achieve nearly linear scalability with this emulator approach. The architecture of this approach presumed three scenarios: mobile devices connected by GPRS in a star topology to a base station, a wireless sensor network connected by GPRS to a base station, and a number of constrained devices connected to proxies, which in turn are connected to a backend.

To achieve link emulation, the authors figured out that traffic between nodes on either a proxy or a base station had to fit either GPRS or 802.15.4 profile. To simulate TCP or UDP traffic, the planned to reuse models from NS2 coupled with a *netfilter*-type traffic scheduler. Gateway emulation should be achieved by either using EMULAB or using a Linux Virtual Machine with OpenWRT (<https://openwrt.org/>). Lastly, node emulation was achieved by using existing software, that supports the Constrained Application Protocol (CoAP) (<http://tools.ietf.org/html/draft-ietf-core-coap>). They used a proprietary Java-based node emulator together with libcoap (<http://sourceforge.net/projects/libcoap/>), which had to be modified to use instances of threads to circumvent the kernel limit for number of processes. This way, they realized 10,000 nodes per VM and were only limited by the maximum number of threads allowed per kernel and free UDP ports. As a possible drawback of their proposed approach for MAMMoTH, the authors stated the huge amount of data which the emulator would produce.

2.1.4. DEUS, COOJA and NS3

A Java-based general-purpose hybrid simulation platform was proposed by Brambilla et al. [31]. To simulate interconnected IoT devices, they used the concept of IoT Nodes, which represent generic Smart Objects, characterized by a mobility model, possible multiple communication models and an energy model.

The top layer describes the application to be tested, while the Adaption layer is tasked with the coordination between the IoT Nodes. Below, there is the layer containing the three aforementioned models and, thereunder, the Java-based general-purpose discrete event simulator DEUS is in charge of advancing the simulation. DEUS is directly used by the IoT Nodes' mobility model, which is implemented by OSMobility, a DEUS based simulation environment to simulate motion. The energy model and network models are realized with COOJA and NS3 simulators, respectively. According to the authors, DEUS they chosen mainly due to scalability and versatility reasons.

To verify their expectations, they conducted six different experiments within an urban smart parking scenario. The exact details of the use case, as well as the exact hardware configuration for the experiments, can be seen in [31]. In each of the experiments, Brambilla et al. increased the number of employed nodes, so that there were 4508, 9016, 22,540, 45,080, 112,700 and 225,400 nodes, respectively. Subsequently, they counted the number of events that arose during the simulation runs and obtained for the aforementioned number of nodes 2.7×10^6 , 5.6×10^6 , 1.4×10^7 , 2.7×10^7 , 6.3×10^7 and 1.2×10^8 events per run, respectively.

2.1.5. Comparison

In comparison with the aforementioned simulators (excluding MAMMoTH, which aims at emulating, not simulating), the here proposed simulator pursues an approach to tightly interconnect

different simulation levels by wrapping domain-specific simulators into models of the different simulation levels and evolving the simulation in a single event-loop. As a consequence, the boundaries between the levels of hierarchy are not as sharp as in the combined approach with S^3 and OMNeT++. This approach also allows recreating models as proposed by the approaches with ACOSO, OMNeT++, DEUS, COOJA and NS3, since the domain specific simulators are contained in the models and so the Smart Object approach can be contained in a special model. The proposed approach of dynamically interchange of models (which possibly contain wrapped domain-specific simulators) for finer grained time-steps also allows for the use of continuous-time approximating techniques (as mentioned in the following sections) without complicated synchronization processes, so that different kinds of models all can work together.

The different approaches to the topic of simulating or emulating large-scale IoT scenarios are overall too diverse, to draw a meaningful comparison between the approaches discussed.

2.2. Homomorphic Encryption Schemes

Soon after the idea of public key cryptosystems was introduced by Whitfield Diffie and Martin Hellman (see [32]) in 1975, the first implementation of such a system was given by Ronald Rivest, Adi Shamir and Leonard Adleman in 1977. This system, which was—after its inventors—called RSA [33], already showed homomorphic properties in multiplication: With the RSA public key modulus being m and the exponent e , the encryption ϵ of message x is given as:

$$\epsilon(x) = x^e \pmod{m}$$

For the multiplication of two plain texts x and y , it hence follows:

$$\epsilon(x \cdot y) = (x \cdot y)^e \pmod{m} = x^e \cdot y^e \pmod{m} = \epsilon(x) \cdot \epsilon(y)$$

Generally speaking, this means that multiplication of two cipher texts c_1 and c_2 can be done without decrypting them first. Ultimately, this means that this operation can even be performed by some third-party not in possession of the private key for decryption. This useful and convenient property encouraged researchers to further pursue and find more encryption schemes with homomorphic properties.

In [34], the author introduced yet another encryption scheme with homomorphic properties, which allows much broader applications such as the calculation of the sum over all input data. It has a very useful application for privacy and anonymization, as we show in Section 5.5.

It took another decade until Craig Gentry published his work on a fully homomorphic encryption scheme in 2009 (see [5,6]). In contrast to the semi- or partially homomorphic schemes known before, fully homomorphic encryption allows any arbitrary computation on cipher texts with no need to decrypt and giving out any cleartext information at all. Unfortunately, fully homomorphic encryption is more expensive in terms of computational demand, thus people need to decide on a trade-off between performance and versatility. We utilize the simulation framework presented in this paper to further analyze this trade-off and present our results in Section 6.

3. Simulation of IoT Networks

As powerful as it is, the discrete-event network simulation framework Omnet++ has its fair share of problems, when tasked to simulate the entirety of a smart city. One of these is the performance loss when simulating the work of and communication between the number of models that are necessary for the simulation of an entire city. However, with the possibility to simulate such a smart city, not only single applications but the entirety of the complex interactions between them could be simulated and used to further facilitate concepts of smart cities and the Internet of Things.

Therefore, this work aims to integrate the powerful Omnet++ framework for its capabilities of simulating network traffic and more important as a base for INET to simulate the Internet, with a

lightweight custom Simulation framework, to reduce performance problems of Omnet++. To achieve this goal, the possibilities of the hierarchical modeling of Discrete-event Systems (DEVS) models are used, to define a “time hierarchy” within the simulation. The idea here is to dynamically switch between models that simplify significant portions of the simulated city (e.g., a quarter or district of the city) with fairly rough-grained time resolution, and coupled networks of models that model smaller parts of the aforementioned models with increasingly finer grained time resolution. This allows us to reduce unimportant parts of a simulated city to resource saving rougher abstractions and dynamically observe important details, where they are of interest.

This paper brings the following novelties over state of the art: The proposed simulation framework supports multi-level simulation using only one simulation technique based on discrete-event simulation. The framework allows dynamic switching between models at different levels of abstraction that simplify significant portions of a simulated IoT network with fairly rough-grained time resolution. This further allows us to dynamically observe details that are relevant and filter ones that are not of interest for a particular simulation scenario.

3.1. Concept

As stated in Section 1, the aim of this paper is to create a general purpose DEVS simulator with hybrid simulation capabilities for large scale IoT and smart city simulations within the VICINITY use cases. Based on the requirements for general IoT simulations as well as requirements unique to the VICINITY project, which are presented in Section 3.2, an approach to the problem is devised in the following sections.

3.2. Requirements

From the various works on the subject presented in Section 2, the most prevailing requirements for a simulator of the IoT have been gathered. Additionally, some requirements have arisen from the involvement of the Chair Design of Cyber-physical Systems with the VICINITY project. Those requirements are examined in the following:

- The possibility to simulate thousands of interconnected devices [8]: Depending on the scenario, it can be necessary to simulate thousands of entities that partake in the IoT. While for example in the simulation of a smart home, there are only a relatively small few devices that need to be simulated, during the simulation of a new area wide service, the number of simulated entities has to rise rather quickly, to provide useful data.
- The ability to run in (almost) real-time for proactive approaches: While high detailed simulation runs can provide insight into the fine-grained processes within the interplay of different devices and technologies, with the sheer size of Internet of Things scenarios in the context of a smart city, these simulation runs tend to be too slow in order to enable proactive approaches. The framework under development should provide some techniques to enable these approaches.
- Hardware in the loop capabilities: To analyze the behavior of prototypes, it would be most beneficial if the developed simulation framework would provide hardware in the loop simulation capabilities or provide the interfaces to enable simple integration of already existing approaches.
- High scalability of scenarios: Somehow connected to the first two requirements is the high scalability of scenarios. The simulation framework’s real-time capabilities should not be lost, when thousands of entities that communicate with each other need to be simulated on a large scale.
- The possibility to employ parallel and distributed simulations: Based on the observations in [10], the developed framework should provide either out of the box parallel and distributed capabilities, or at least be built in a way that facilitates later adaption of these principles to further support the aforementioned requirements.

- Fast model development for fast employment in use cases: To be employed in the VICINITY project, the developed framework should offer possibilities that support rapid model development and possibly even the use of functional mock-up interfaces.
- The possibility to unify multiple heterogeneous technologies on all levels of IoT: The simulator should support modeling the prevalent technologies of the respective domain and enable the interplay between them.
- The possibility to integrate further domain specific simulators into the framework: This last requirement pretty much speaks for itself. If during the deployment of the developed simulation framework the necessity of integrating further domain specific simulators arises, it should be doable with little effort.

4. Approach

Discrete event systems have been topic of research for over 40 years now, thus techniques for parallel and distributed simulation are well-known and there exist multiple extensions to the original specification for various problems and fields of application, such as PDEVS, DynDEVS (which enables dynamically changing connections within DEVS [35]) and many more. In addition, modeling techniques for discrete event simulations are well known. Due to their relationship to finite state automata, they are easy to grasp for unexperienced developers. For that reason, this approach uses the DEVS specification as foundation.

The majority of the examined related work identified the scalability as crucial for large-scale simulations. Thus, the main focus of this work is to introduce a simulation framework for large-scale IoT simulations together with a modeling technique to enable rapid modeling of large-scale use cases.

Rather than combining different domain specific simulators to a multi-level simulation that invokes different simulators at different points in time during the simulation, this approach aims at dynamically altering the advancement of time and the models' level of detail during the simulation. This means that, to allow high scalability even in large scale scenarios, the proposed framework simulates areas that are of no interest for the user with far less detail and bigger steps in time advancement. Specifically, the framework uses multiple models with different degrees of detail and time advancement for the same simulated entity and changes between them during the simulation, depending on the interest of the modeler. A real-world comparison could be imagined as a magnifying glass that shows its focal point in great detail, while the everything else stays the same. This enables simulations of large-scale scenarios such as the introduction of a new service in the area of a whole smart city, for example, where at the same time one can study the relationships between simulated entities all over the area, as well as detailed processes in single simulated entities or hardware in the loop.

Similar to the multi-level simulations discussed in Section 2, this approach uses multiple levels of detail to simulate the desired scenarios. However, rather than implementing the different levels through different domain-specific simulators, here, the levels in the hierarchy of the simulation are defined by the degree of time advancement and detail in the used models. This means of course additional costs in terms of modeling, since different models with varying degree of detail for the same system have to be developed, but through the re-usability of the used models; this is a one time cost.

In practice, the use and substitution of models during simulation runs is achieved by a switch between a low detail atomic DEVS model and a whole network DEVS model, that models the atomic model in more detail with finer grained time advance steps, and thus creating a new level of simulation. Hence, a good balance between models of areas that are of special interest and models that provide only the necessary background should efficiently lower the amount of produced events by the simulation. Besides, when the coarse and finer grained models are placed inside partitions together, already well-used PADS techniques for discrete event simulators can be applied. Of course, such lower levels could be represented through domain specific simulators as in the approaches shown in Section 2. For

that reason, in this framework, OMNeT++ with its INET expansion is used to implement the more detailed lower level models of *smart things* and their interplay.

4.1. Level Hierarchy and Structure

Expanding DEVS

To define the proposed approach in terms of the DEVS specification, a new kind of model is introduced. Hereafter, it is called Hierarchical Atomic. It is a combination of an atomic and a network DEVS model that additionally provides functions to transfer one model's state into another and to select, which one should be the currently active:

$$atomic \equiv \langle atomic, network, \{transport\}, select \rangle, \quad (1)$$

where *atomic* and *network* are the contained atomic and network DEVS model, respectively. $\{transport\}$ is the set of transport functions that transfer the models' state and *select* is the function that chooses the currently active model. In more detail, the new type of model—in the following called *Hierarchical Atomic*—is defined:

$$atomic \equiv \langle S_A, X_A, X_N, Y_A, Y_N, \\ D, \{M_N\}, \{I_N\}, \{Z_N\}, \delta_{int}, \delta_{ext}, \delta_{con}, \\ \lambda, ta, select, \{transport\} \rangle, \quad (2)$$

where *A* and *N* denote if the component belongs to the atomic or the network part of the combined model, respectively. The *transport* functions

$$transport_{A \rightarrow N} : S_A \rightarrow S \subset \cup_i S_i \in D, \text{ for } S_i \in D_i \quad (3)$$

and

$$transport_{N \rightarrow A} : S \subset \cup_i S_i \in D \rightarrow S_A, \text{ for } S_i \in D_i \quad (4)$$

denote the functions that are necessary to transport the states of the contained models between each other, when a new level in the simulation is opened or closed. Note that neither of the functions has to be an isomorphism, since the contained network should be of course more expressive than the contained atomic. As a consequence, only a subset of possible states of the network models' components is used.

In more detail, the different contained models define the boundaries of the levels of a multi-level simulation: The encapsulated atomic DEVS model is on the same level as the new defined model, while the encapsulated network DEVS model belongs to the simulation level below. This way, the transport functions mark the transition between levels. Emanating from the top-level model in the simulation, all networks that are the same amount of encapsulation away from this root belong to the same level. This should be taken into account when developing models for a given scenario: When the different network models that define a level differ too much in terms of their step size in time advance, the use of that level for performance gain can degrade. That is, when one of the network models that belong to that level would advance in time in much smaller steps than the rest, it would impact their efficiency as well.

4.2. Model Reuse and the Model Tree

The organization in atomic and network DEVS models and the closure under coupling characteristic of DEVS allow for easy (re-)arrangement of model building blocks to more advanced and complex models in a tree-like structure. This procedure is supported by the here proposed approach. As can be seen in Figure 2, the here introduced model that contains parts of the level architecture can

easily be treated just as plain atomic DEVS model that is part of a network model in all positions in the tree structure. Moreover, the atomic models that belong to the contained network model can be exchanged with the new model as well, thus forming the different levels of hierarchy within the level of detail and time advancement of the simulation.

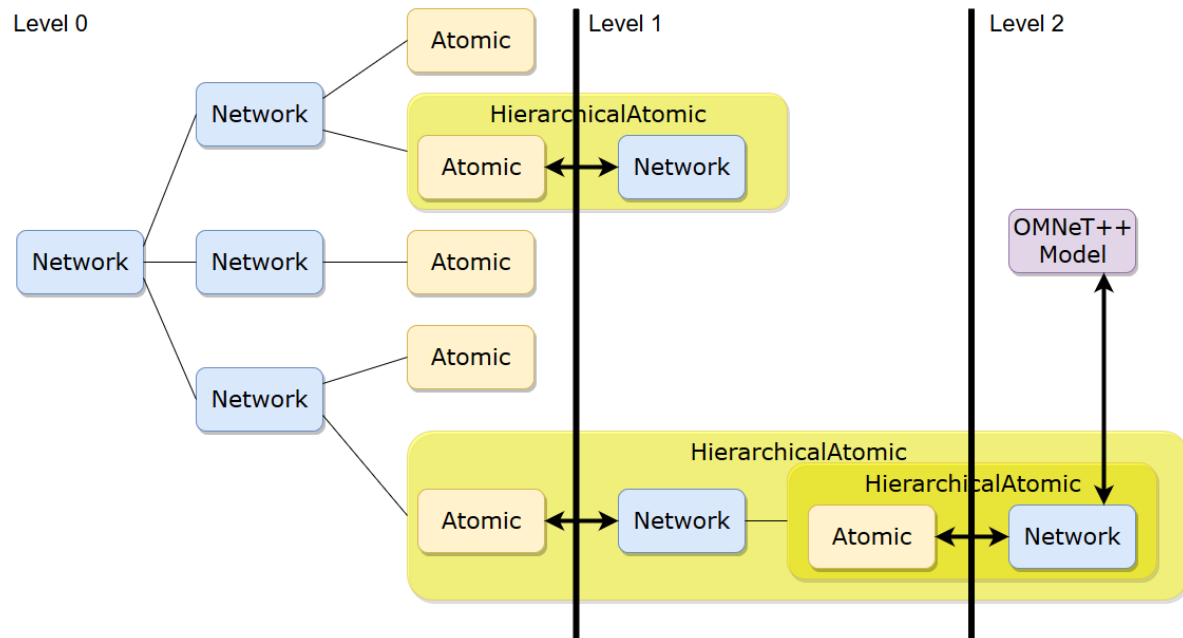


Figure 2. The model tree and organization of hierarchy levels.

The consequences of this also tree-like organization of the simulation’s levels are of importance for the general approach to modeling with the proposed simulator and influence the possible parallelization of the sequential approach.

First, regarding the approach to modeling, a bottom-up procedure is recommended, where the deepest and most detailed level is designed in its whole and then partitioned into smaller leaves of the tree. As the performance gain of the dynamic exchange between models with varying level of detail is dependent on the specific scenario, it is possible that more efficient partitions of models along the tree are only eventually evaluated under simulation. The bottom-up construction here ensures that the main part of the scenario has to be developed only once and then merely has to be partitioned accordingly. In addition, if the models that are to be exchanged are the kind of whole protocol-stacks or space-divided parts of the environment, they in turn can be heavily be reused. If the capabilities of exchanging models with more or less elaborated representations of the system under simulation are used efficiently, in this way, the environment of smart home scenarios could be simulated following a user of a service through his way through a smart city, exchanging models on the fly, as soon as they are needed. In the implementation of the proposed approach, it is of importance where in this tree the integrated domain specific simulators are placed. Obviously, a placement along the nodes of the tree as well as a simple hierarchy of one domain specific simulation kernel per level is conceivable.

Second, regarding possible partitioning techniques that could be used in a parallelized employment, it has to be taken into consideration how the different networks are placed. A high density of encapsulated networks in unluckily partitioned models could negatively affect the simulations performance. Therefore, modeling for a parallel scenario could be prolonged in comparison to other approaches.

During the implementation of the here proposed simulator, it was decided to place multiple instances of a simulator across the tree. The placement of simulators inside the introduced hierarchical atomics allowed them a more flexible implementation of their functionality. At the same time, it is ensured that the simulator that advances a specific model and the model itself remains separated.

Furthermore, as this approach is meant to exchange models on various levels of the tree at the same time, a placement of a single instance of a domain specific per level would prove impractical since several models along the whole simulation would be thwarted through a bottleneck.

4.3. Synchronization

Whenever an atomic model gets interchanged with a network model and vice versa, synchronization errors can occur. This is the result of already scheduled autonomous events, that now may become invalid. Looking at the mode of operation of the implemented abstract simulator, one can see that subsequently to an autonomous event the imminent model gets rescheduled with its time advance function. This little detail can be used to save complicated synchronization procedures: if the exchange of models can only occur during events, the subsequent rescheduling of the containing model will minimize the chance of falsely scheduled models.

However, depending on the concrete implementation of the Future Event Schedule (FES), perhaps there are still internal events scheduled that are now too early or too late. For example, when the simulator switches from a detailed, slower advancing network model to the respective atomic one, it can happen that internal events of one of the network's faster advancing components are still inside the FES. The easiest solution to this is of course a future event schedule, which keeps at most one event of every atomic model inside itself. Through the encapsulation of the network model inside what appears to be an atomic DEVS model and the immediate rescheduling after execution of the internal event, the model would be scheduled correctly. However, in practise, this is not always that easy. Different areas of application can demand for different scheduling strategies and the possibility exists that not all of them can respect the requirements of this approach. OMNeT++ already is designed with several possible scheduling classes. To maintain maximal flexibility of the proposed approach, it is therefore necessary to answer the problem directly. Thus, the proposed new kind of model has to keep track of its own schedule on its own and decide on how to react to a falsely scheduled autonomous event independently. While models can keep track of their own model time through their time advance function and the fact that they get informed about the elapsed time since the last internal event (which supposedly followed a correct time advance of the model), when the occurrence of internal events loses reliability, this is no longer sufficient. Hence, the model needs the ability to acquire the global simulation time. This can be achieved by, for example, allowing the single models to access the simulation environment or the simulation kernel, as OMNeT++ does, or it can be accomplished by providing the global simulation time to the model with every function call to one of the state transition functions or the output function.

4.4. HiL Interface

In many cases, interaction with objects outside the simulation (e.g., the real VICINITY network) is necessary, whether for direct interaction with real objects or just to control objects inside the simulation. For this situation, VICINITY provides a good communication platform because of its device and standard agnostic nature. To integrate a particular infrastructure into VICINITY, an adapter that provides the REST-API to communicate with the VICINITY agent is required. In the case of an infrastructure simulated by our simulation framework, a way to communicate with the VICINITY Agent outside the simulation is necessary. We implemented a socket based approach to communicate with the VICINITY Agent. With this approach, a full network simulation is not necessary to communicate with the VICINITY Agent.

4.4.1. OMNeT++ Socket Server

To communicate with the VICINITY Agent, a socket server has to be integrated into the OMNeT++ simulation. The purpose of this server is to accept API-Calls of the VICINITY Agent, exchange the appropriate data to the OMNeT++ simulation, and then to send a response.

The main difficulty for this task is to properly synchronize the socket requests with the discrete event scheduler of OMNeT++. First, the simulation has to be executed in real-time. This can be achieved by synchronizing event-time with wall-clock time, i.e., delaying the execution of events in the scheduler. The time between two events could then be utilized by the scheduler to listen and process socket data and, if necessary, inject new events into the event-queue to process received data. In this work, however, a separate socket server-thread is used to preprocess all incoming, and postprocess all outgoing data on the socket. By this way, the processing-impact on the simulation is minimized, since only relevant data are exchanged with the simulation thread. All other processing steps, e.g., communication and protocol overhead, are then transparently handled by the server-thread without impact on the simulation. The simulation-scheduler has then to wait for notifications of the server-thread instead of socket requests. This is achieved with promise and future objects from the C++ standard library. Figure 3 shows the basic concept of synchronization between the socket server-thread and the simulation thread.

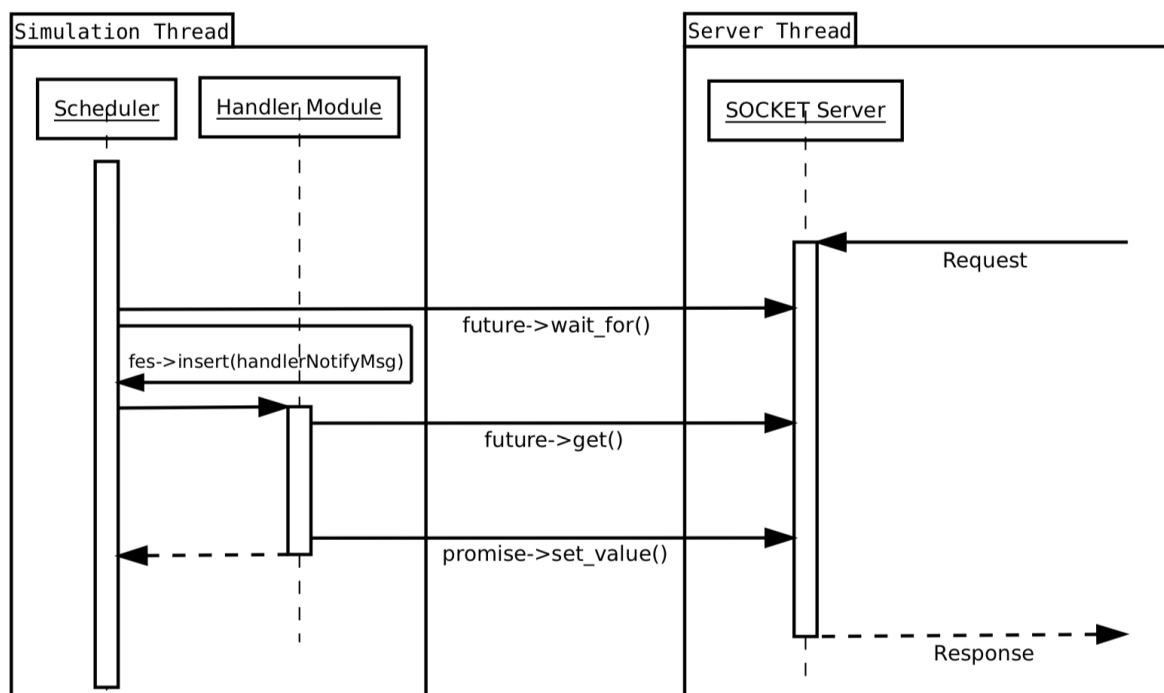


Figure 3. Simulation- and server-thread synchronization.

4.4.2. OMNeT++ VICINITY Adapter

To allow communication between VICINITY and simulated objects, independent of the network used inside the simulation, a simulated access-point is used. This access-point acts as a router between VICINITY and the simulation, i.e., relaying and protocol-conversion of requests and responses. The access point module VicinityAccesspoint communicates with VicinityAdapter, a subclass of SocketServer, which runs the socket server thread to handle requests from VICINITY. As for the concept of synchronization, as shown in Figure 3, VicinityAdapter has the role of the socket server, while VicinityAccesspoint is the handler module. In the following the functionality of the two modules is shown:

- VicinityAdapter: The VicinityAdapter is initialized with a port to listen on. When started by calling startAdapter(), a future-object is returned. As soon as a request is received, an object of type RequestStruct is made available, which holds all relevant data of the request. Additionally, promise-object is passed, which is used to set a ResponseStruct. In this struct, all data relevant for the response is transmitted, together with a new promise-object for the next incoming request.

- VicinityAccesspoint: The VicinityAccesspoint module is responsible for routing adapter-requests into the simulation. Additionally, VicinityAccesspoint has to provide object-discovery data when requested by the adapter. To collect all necessary data for object-discovery, the access-point broadcasts a message of type ObjectDiscoveryPkt (see Figure 4) with isRequest=true to all connected modules. Every module which shall be accessible via VICINITY then responds with an ObjectDiscoveryPkt message with isRequest=false and its own object-data JSON in objectJson. The access-point then composes a JSON object with all collected ObjectDiscoveryPkt responses and serves it when requested by the adapter. When a request to set/read a property or action is incoming, the access-point forwards it to the corresponding module with a VnetPkt message. The receiving module then processes the request and responds back to the access-point with another VnetPkt message. After the message is processed, the access-point generates a response for the adapter and the request is completed. If an addressed module does not respond in a given time, a timeout occurs and the access-point responds to the adapter with VININITY_RESPONSE_NOT_FOUND, which finishes the request with an error-code.

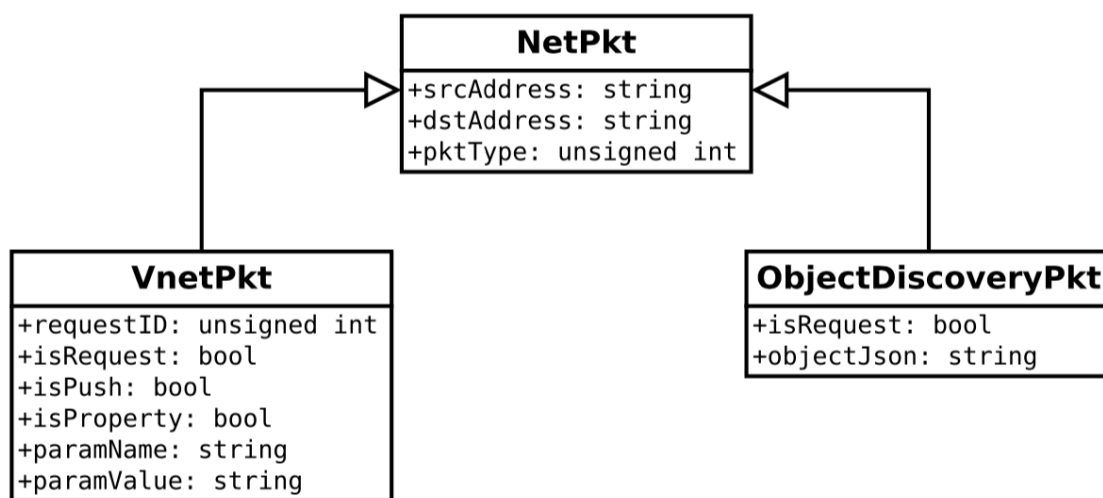


Figure 4. Class diagram of messages used inside the simulation.

4.5. Simulation Scheduler

As previously explained in Section 4.4.1, the OMNeT++ scheduler has to be adjusted to integrate the adapter. The original SocketRTScheduler was created to actively listen on a single socket in the time between simulation events. In our framework, however, the sockets are not managed by the scheduler, but by their own thread. This circumstance actually eases the task of the scheduler, because only a single future-object has to be monitored and, on change, the handler module (access-point) has to be notified. When the simulation is started, the access-point module (VicinityAccesspoint) registers a notification-message and a pointer, which always points to the latest future-object of the adapter-thread, with the scheduler. Since the scheduler is a real-time scheduler, it has to wait until the time of the next event (handled in cSocketRTScheduler::takeNextEvent()). During this wait time, the scheduler waits for the future-object to be ready. When the future-object is ready, the simulation time gets updated to match the actual time (to account for the time blocked on the future), and the notification-message of VicinityAccesspoint is scheduled to be delivered to VicinityAccesspoint at the current time (instantly). After receiving the notification-message, the access-point accepts and processes the data from the future-object. Additionally, the access-point has to update the future-object, which is pointed to by the scheduler, so the scheduler can wait for the next request. If the future-object is not valid, the scheduler simply sleeps until timeout (or next event).

5. Case Study: Smart Energy Use Case

To illustrate the applicability and performance of the developed multi-level simulator, we modeled and simulated a smart energy use case with applied homomorphic encryption. This use case is an integration of the use cases of Kölsch et al. [36] and Kölsch et al. [37]. This particular use case describes a smart energy scenario within a city. The city has a photovoltaic system and a windmill as power suppliers and a parking lot and a couple of houses as consumers. Electric vehicles can move inside the city and the parking lot. By using a smart parking service through a mobile APP, users of the system can request to reserve their parking slot of choice within the participating parking facilities. The availability of the parking slots is then displayed through the mobile APP as well as through the optical indicators located on the respective parking slots for random people who do not participate in the smart parking service.

The described scenario was modeled and simulated using the proposed approach at three distinct levels of abstraction: The first two higher levels were implemented only using classes provided by the implemented core simulator. The third (lowest) level was implemented with OMNeT++ 5.4.1 (<https://www.omnetpp.org/21-articles/3752-omnet-5-4-1-released>) and its INET extension 4.0 (<https://inet.omnetpp.org/2018-06-28-INET-4.0.0-released.html>).

5.1. The Highest Abstraction Level–Level 0

The highest level of abstraction models abstract processes that are needed to provide basic information for the following lower levels of the simulation scenario. Furthermore, the power generating entities are modeled on this high and abstract level. This is shown in Figure 5.

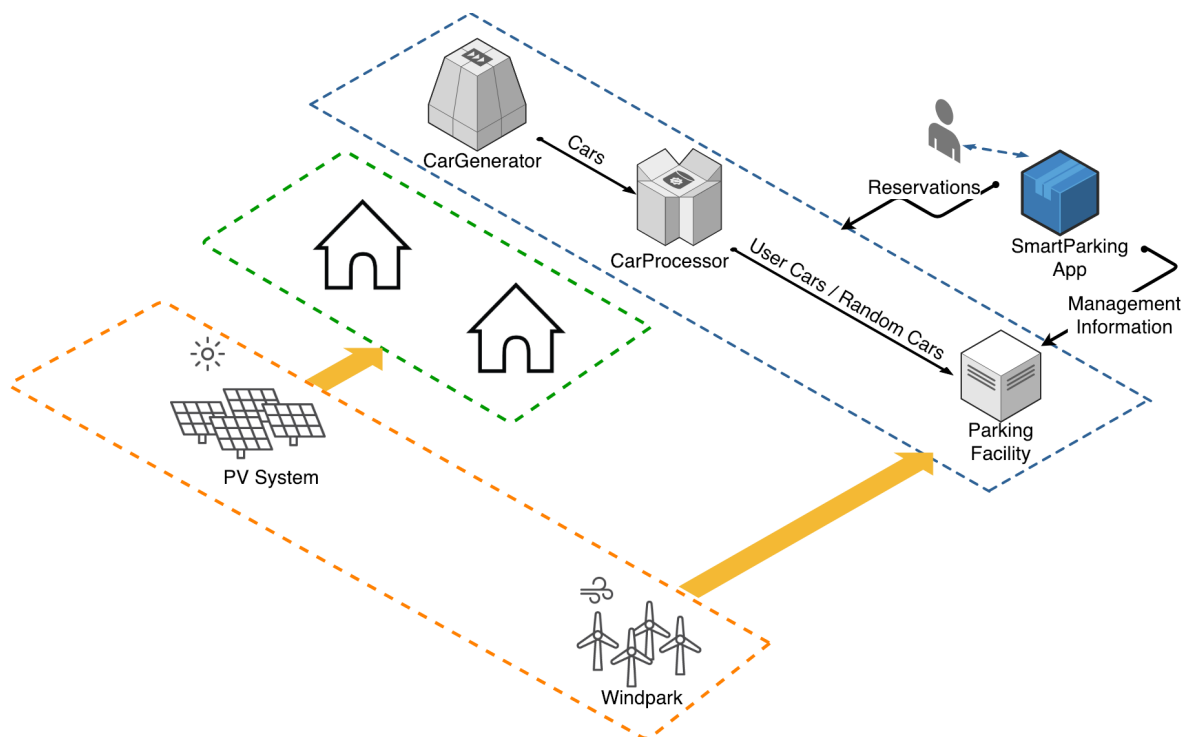


Figure 5. Smart energy use case: Level 0.

The CarGenerator atomic model acts as a source to the rest of the modules and provides the information needed to simulate users and random cars at the lower abstraction levels. This information is then passed to the CarProcessor.

The CarProcessor then determines if the received information is used to simulate a scenario with a random visitor of the parking facility or with a user of the smart parking mobile APP. In the latter case, information about the desired parking slot is generated and used in an attempt to make a reservation

via the model of the smart parking APP. If this reservation fails, the information of the APP user is treated as information about a random visitor of the facility and sent further to the ParkingFacility.

Once entering the ParkingFacility model, the received information is used to model the abstract behavior of both random visitors and smart parking service users competing for available parking slots, parking and subsequently leaving the facility again.

Users of the APP that succeed with a reservation will directly target their desired parking slots while random arrivals and users that failed to reserve their desired slot will choose the first free available parking slot. When arriving at the slot, it will be determined if it is still free or in the meantime has already been reserved by a user or taken by another random car that arrived first. If it has already been taken, they will head for the first free parking opportunity again. If they do not succeed in finding one, the car will leave the parking facility. If the parking process succeeds, the car will occupy the chosen parking slot for a while and then subsequently leave the parking facility again.

5.2. The Middle Abstraction Level—Level 1

The following level of abstraction was used to further detail the processes inside the ParkingFacility. It is shown in Figure 6. It divides the raw ParkingFacility into three different parking decks that internally mimic the behavior of the parking facility in Tromsø.

The information about car arrivals will be forwarded to the different parking decks in sequence. When entering the facility, the parking decks have to be traversed until the desired parking spot is reached. When a car leaves the ParkingFacility, the decks have to be traversed again in order to reach the exit. Additionally, the ParkingDeckControl is used to send information from and to the model of the APP.

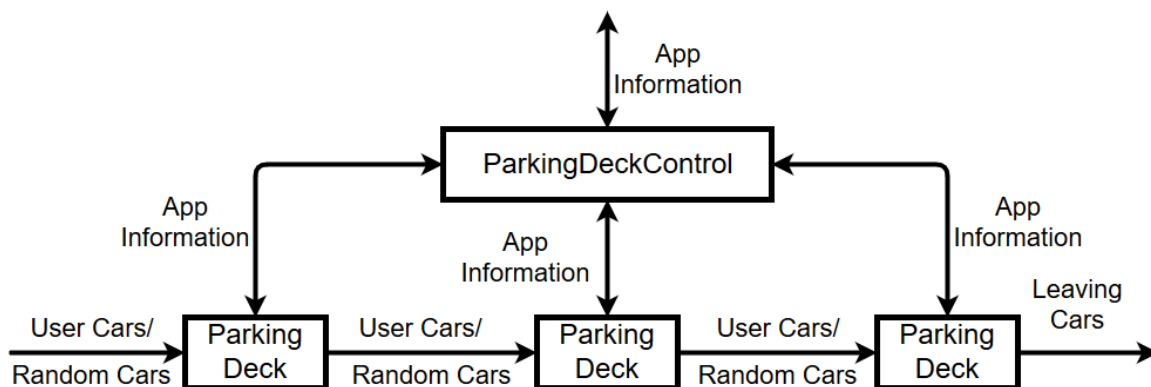


Figure 6. Smart energy use case: Level 1—the parking facility.

5.3. The Lowest Abstraction Level—Level 2

The lowest level of the simulation scenario was modeled with OMNeT++ 5.4.1 and INET 4.0. Here, the information produced by the higher levels described above was used to dynamically instantiate simulated entities and to represent the communication between sensors, actuators, and the APP with the advanced capabilities of INET.

At this level, the parking facility was modeled as an OMNeT++ compound module and the single parking decks as submodules of it. Although the different submodules can interact with each other across submodule boundaries, only those associated with the respective active higher level parts of the simulation will actually be active.

If the higher level parking deck model switches to the respective part of the OMNeT++ module at Level 2, cars will be created as mobile nodes with specific characteristics. The characteristics depend on the information generated at the levels above and the cars behavior is determined by the corresponding states at Level 1. Depending on if a car is now in the phase of searching for a parking slot or if it is

already parked or even leaving the parking deck, the wireless node will be created and its goals will be set accordingly. One such parking deck modeled in OMNeT++ can be found in Figure 7.

Every Car has a battery, that is discharging as long as the car is moving inside the environment. When a car is parked inside the parking facility, its accumulator is charged. When it is fully charged, it stops charging and the car is ready to drive away.

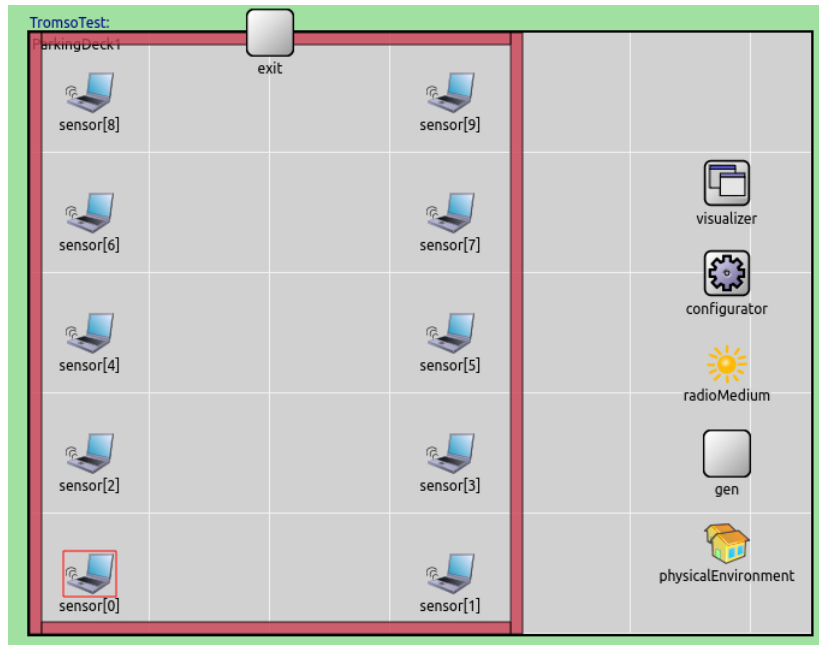


Figure 7. Smart energy use case: Omnet++ model of parking deck at Level 2.

5.4. VICINITY Bridge

To also demonstrate the functionality of the hardware-in-the-loop interface to the real VICINITY network, the use case was extended with a VICINITY bridge on the lowest abstraction level. As discussed in Section 1.1, VICINITY offers an infrastructure for connecting otherwise isolated islands in the IoT landscape. Naturally, our simulated network can be seen as such an island. As outlined in Section 4.4.2, we can integrate VICINITY and our simulation seamlessly by introducing a VICINITY adapter to the Omnet++ simulation. To VICINITY and all its connected devices and value-added services, all simulated devices that exist only virtually within our simulated network will appear as any other ordinary device. To our value-added service, they look and behave the same way as any physical device would, which enables us to:

- rapidly prototype any given scenario or use case and test it without first deploying devices in the field; and
- scale any existing use case up and test how it will behave with more attached devices

As installing and deploying new sensors on our parking deck outline in Figure 6 is costly and time consuming, this gives us a much quicker way to start and continue implementing our front-end application, while this deployment takes place in the real world. Furthermore, we can utilize this virtual setup to evaluate different approaches for our enhanced privacy modules based on homomorphic encryption. In the following, we utilize this setup to measure the impact of different homomorphic encryption schemes on the overall runtime behavior in a real, online test scenario.

5.5. Integration of the Homomorphic Encryption Micro-Service

As briefly discussed in Section 1.2, homomorphic encryption is particularly useful, when data are supposed to be processed by a third party, which cannot fully be trusted. One example could be a

value-added service by someone else. If a user is expecting some particular benefits or is even forced into using it, keeping his sensitive data private is a major concern. Missing trust can potentially even be a showstopper for the whole Internet of Things.

Homomorphic encryption can help us with the above mentioned privacy concerns. However, this privacy comes at the price of increased computational effort for encryption, decryption and also functions evaluated on cipher texts are more costly in terms of computational demand. Currently, the VICINITY project (see Section 1.1) is investigating the use of homomorphic encryption for these scenarios. To evaluate its practical feasibility, the computational overhead introduced by the use of homomorphic encryption needs to be analyzed. As there are numerous potential encryption schemes that offer homomorphic properties, it is also important to evaluate them. In particular, we want to compare partially and fully homomorphic encryption to evaluate the overhead introduced by the latter and also compare both to the same scenario executed without any encryption and hence without ensuring privacy at the VAS. To this end, we simulated one of VICINITYs use cases with partially (additive) homomorphic, fully homomorphic and without encryption as to calculate the overhead this introduces under Lab conditions. Our lab setup utilizes hardware-in-the-loop simulations of the VICINITY infrastructure and can be adjusted way quicker and cheaper than an actual re-deployment on site. Figures 8 and 9 visualize the two simulated use cases.

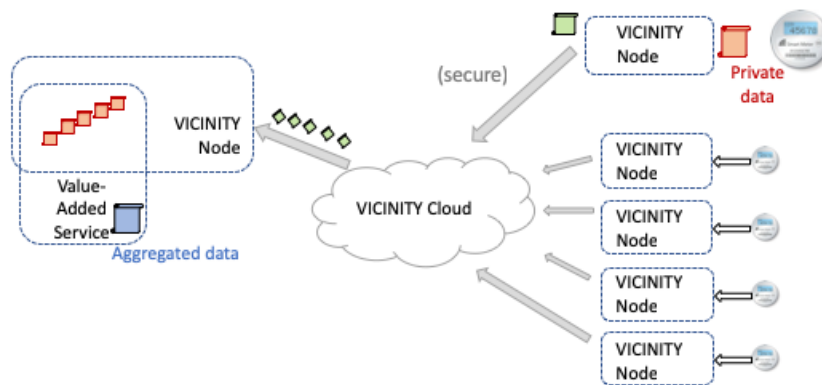


Figure 8. Use case integrated into the VICINITY network. Private data are available to value-added service in clear text.

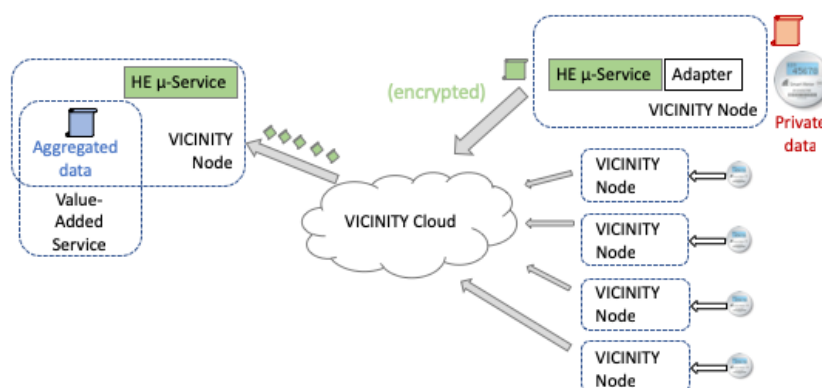


Figure 9. Homomorphic encryption micro-service applied to use case.

As shown in Figure 8, energy consumption data of the simulated cars are gathered and transmitted to the operators’ value-added service. The operator is only interested in the overall energy consumption of its whole fleet and so it calculates the sum over all inputs first. As the energy consumption allows to draw some conclusion about the owners behavior, we consider these data as private data and also wish to keep this data private and not share it with any third-party.

As neither the operator nor the users themselves have any particular interest in sharing their individual, private data, we introduce the homomorphic encryption micro-service. It is integrated into the VICINITY dataflow as shown in Figure 9 and can be compared to the setup in Figure 8: Input data (e.g., the Energy Consumption of each car) are encrypted using a homomorphic encryption scheme. In the case of partially homomorphic encryption, we used a simple Paillier encryption (see [34]), which allows addition of cipher texts. To calculate the overall energy consumption of the whole fleet, this is sufficient. As this concept will be applied to other use cases as well, addition might not suffice in all cases. Hence, we further simulated this scenario using the Brakerski–Gentry–Vaikuntanathan (BGV) scheme (see [38]), which is a fully homomorphic encryption scheme. In both cases, the encrypted payload is then transmitted through the VICINITY peer-to-peer network and to the operators value-added service again. The homomorphic encryption micro-service performs the addition on the encrypted inputs, followed by a decentralized decryption on the aggregated data. This way, only the anonymized, aggregated data are visible and handed over to the value-added service in the first place. All measurements are compared to the use case shown in Figure 8, which serves as a baseline.

6. Evaluation

To evaluate the overhead introduced into the dataflow, due to the homomorphic encryption, we evaluated three scenarios: without homomorphic encryption, with partially homomorphic encryption and with fully homomorphic encryption in order to compare their runtimes. Currently, the encryption service utilized open source libraries, providing wrapper interfaces to their low-level functions and integrating the encryption seamlessly into the VICINITY architecture. For the fully homomorphic encryption, we utilized the HELib Library [39]. The implemented encryption scheme of HELib is based on the Brakerski–Gentry–Vaikuntanathan (BGV) scheme, yet implementing numerous optimizations for a more efficient runtime behavior. For the partially homomorphic encryption, we integrated the libhcs [40] library into our encryption service. This library implements “a number of partially homomorphic encryption schemes” [40], yet we focused on the implementation of the Paillier encryption scheme for additive homomorphic encryption, which is necessary for our given use case. Finally, a simple Python script based on the Python Flask framework [41] was implemented to mimic the endpoints of the encryption service, yet operating on plain text input data. However, the simulation framework presented in this paper is not limited to either the given encryption service or any of the given encryption schemes. In further research, more encryption schemes will be tested and evaluated.

6.1. Experimental Setup

The simulations were performed within an Arch Linux virtual machine—that is at the time the development environment of the simulator—on a Mac Pro with a 3.5 GHz 6-Core Intel Xeon E5 CPU, 16 GB 1866 MHz DDR3 RAM, AMD FirePro D500 3072 MB GPU on MacOS Mojave. The virtual machine used 6 cores and 8192 MB RAM. The encryption service as well as the plain text aggregation resided on an external server, so that differences in the overall topology can be neglected in the comparison. It was run on a single 2.7 GHz Intel Xeon E5 Core. The simulation and the encryption service communicated through ethernet. This setup with separated simulation and encryption service also ensured that there is no computation power for the simulation lost to the encryption service.

6.2. Results

As can be seen in Figure 10, the runtime results of different simulation runs increase with an increasing number of cars which participated in the described VAS. This can be explained by the rising computational cost of simulating the necessary communications between the VAS participants. The increase of runtime of the simulation scenarios which use homomorphic encryption compared to the scenario without it stems from two separate causes: first, the pure computational cost on the

service side for the encryption; and, second, the time necessary to send the huge (in comparison to unencrypted data) response ciphers into the simulated network and over real ethernet.

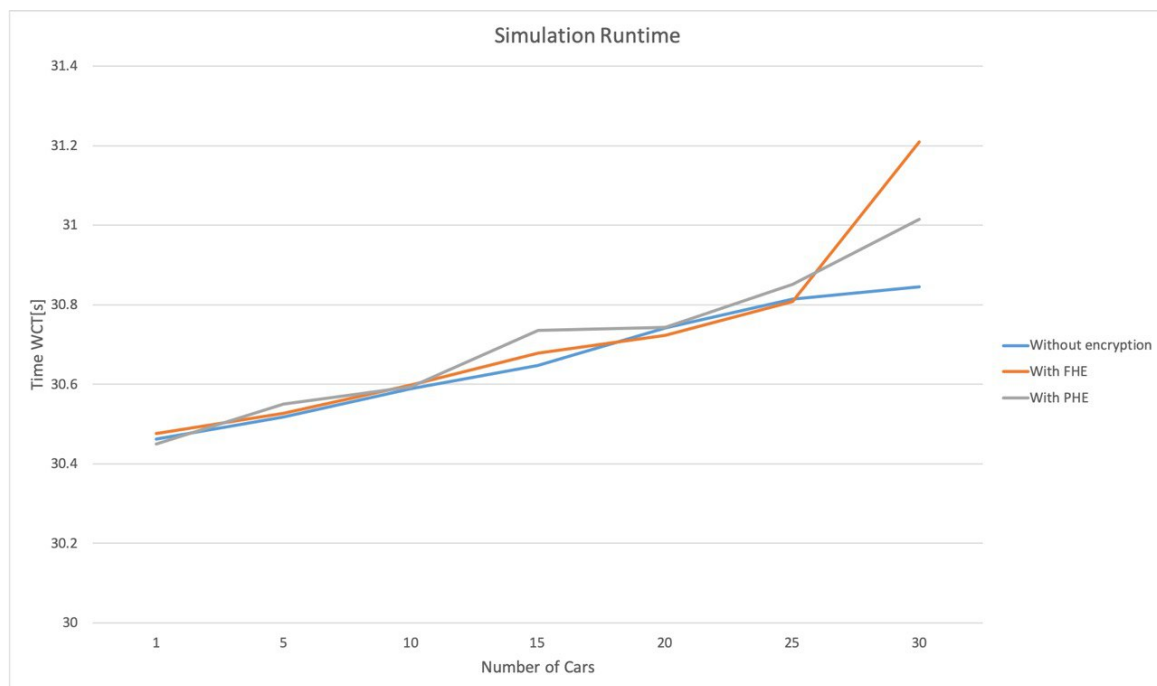


Figure 10. Runtime of the simulation with and without (fully, partial) homomorphic encryption.

This is probably due to the limits of the used realtime scheduler used for the hardware in the loop. With too many communications into the real network or too big payloads (e.g., huge numbers of ciphers), the used scheduler has to drop some in- or outgoing communications in order to keep its realtime capabilities. Apart from network performance and congestion on our lab/office network, which apparently has the biggest impact on the overall performance, we utilized one of the most recent and still actively developed implementations for fully homomorphic encryption (see [39]). The library itself implements the BGV encryption scheme, which already takes advantage of newly found improvements and performance tweaks to homomorphic operations. This does not hold for the library used in our partially homomorphic implementation. We used libhcs (see [40]), which offers implementations for the Paillier, El-Gamal and Damgard–Jurik cryptosystems. While all of these are well known, they are no longer actively worked on, since the dawn of practical fully homomorphic encryptions. Hence, they cannot take advantage of newly found improvements and performance tweaks and thus on some scenarios even perform worse than fully homomorphic encryption.

7. Conclusions

One major and very important result of the experiment is that the overhead introduced by using homomorphic encryption (either partially or fully homomorphic) only makes up for a fraction of the overall runtime, compared to the runtime of our plain text simulation. As can be seen with, e.g., our runtime comparison for 10 cars, fully homomorphic encryption and plain text are merely identical. For 30 cars, fully homomorphic encryption performs worst of all three approaches. However, the overhead is only 1.2%, thus, given the benefits this offers, it can be neglected for our application.

For our future applications, these results are very valuable. For practical applications of homomorphic encryption in the context of the VICINITY project, the number of simulated cars are realistic and feasible. Hence, we expect no penalties in terms of runtime, due to the use of homomorphic encryption. However, this new approach will greatly improve privacy of the car owners and thus greatly boost the acceptance of VICINITY to them.

Another result of the experiment is that we proved the applicability of hardware-in-the-loop- and software-in-the-loop-simulation within the proposed simulation framework. While the used realtime scheduler for the simulation definitely had an impact on the overall simulation runtime (which has obviously to be expected, since it needs to communicate with real hardware and has to slow down simulation time), it fits perfectly in the approach of dynamically interchangeable models at runtime, which can contain wrapped simulators and depict simulation time in finer grained and therefore more detailed steps.

8. Future Work

As stated in [36] already, in its current state, the prototype supports the usage of continuous-time models through hybrid models. However, continuous interaction between such models can still pose some problems regarding the discrete-time architecture of the simulator. A solution to this was proposed in the generalized DEVS specification [42]. It uses polynomial events to approximate continuous output.

Large-scale IoT scenarios can massively profit from parallel and distributed simulation techniques. Through the integration of MPI, OMNeT++ supports parallel and distributed execution. Future work should use this provided architecture and enable the developed simulator to make full use of OMNeT++ capabilities.

The proposed simulation framework can be used to evaluate other software or hardware especially designed for the internet of things. For example, the approach in [43] can be evaluated for performance and design.

The importance of the functional mock-up interface (FMI) for the simulation of cyber-physical systems (CPS) is quite obvious. For the integration of more domain-specific languages and simulators at the lowest level, FMI needs to be integrated into OMNeT++. We already integrated SystemC models into the simulation without using FMI. However, the need for more exact simulation results can greatly benefit from the integration of FMI and languages such as Modelica.

Finally, the runtime evaluations showed that homomorphic encryption is indeed feasible for practical implementations. In future work, we will integrate a fully homomorphic micro-service (see Figure 9) into the VICINITY architecture. Our results have shown that this implementation introduces a negligible runtime overhead, yet offers a great added benefit in terms of privacy (compared to the plain text scenario) and versatility (compared to the partially homomorphic encryption). Using the VICINITY Bridge Feature (see Section 5.4), we can further test this approach in a simulated environment and without the risk of affecting real world applications.

Author Contributions: conceptualization, J.K., A.R. and C.H.; methodology, J.K., A.R. and C.H.; software, J.K., A.R. and C.H.; validation, J.K., A.R. and C.H.; formal analysis, J.K., A.R. and C.H.; investigation, J.K., A.R. and C.H.; resources, C.G.; data curation, J.K., A.R. and C.H.; writing—original draft preparation, J.K., A.R. and C.H.; writing—review and editing, J.K., A.R., C.H. and C.G.; visualization, J.K., A.R. and C.H.; supervision, C.G.; project administration, J.K. and C.G.; funding acquisition, C.G.

Funding: As part of the VICINITY project, this work was supported by EU (European Union) program Horizon 2020 under grant agreement number 688467.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mynzhasova, A.; Radojicic, C.; Heinz, C.; Kölsch, J.; Grimm, C.; Rico, J.; Dickerson, K.; García-Castro, R.; Oravec, V. Drivers, standards and platforms for the IoT: Towards a digital VICINITY. In Proceedings of the 2017 Intelligent Systems Conference (IntelliSys), London, UK, 7–8 September 2017.
2. Kölsch, J.; Heinz, C.; Schumb, S.; Grimm, C. Hardware-in-the-loop simulation for Internet of Things scenarios. In Proceedings of the 2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), Porto, Portugal, 10 April 2018.

3. García Castro. VICINITY D2.2: Detailed Specification of the Semantic Model. Available online: <https://vicinity2020.eu/vicinity/content/d22-detailed-specification-semantic-model> (accessed on 19 October 2019).
4. D1.6: VICINITY Architectural Design. Available online: https://vicinity2020.eu/vicinity/sites/default/files/documents/vicinity_d1_6_architectural_design_1.0_0.pdf (accessed on 19 October 2019).
5. Gentry, C. A Fully Homomorphic Encryption Scheme. Available online: <https://crypto.stanford.edu/craig/craig-thesis.pdf> (accessed on 19 October 2019).
6. Gentry, C. Computing Arbitrary Functions of Encrypted Data. *Commun. ACM* **2010**, *53*, 97–105. [CrossRef]
7. Evans, D. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. Available online: https://pdfs.semanticscholar.org/e434/2d4687233ae12aa689407f97502d87a9f27b.pdf?_ga=2.102884352.1869453041.1571462370-231942018.1567670099 (accessed on 19 October 2019).
8. Ferretti, S.; D'Angelo, G. Smart Shires: The Revenge of Countrysides. In Proceedings of the IEEE Symposium on Computers and Communications, Messina, Italy, 27–30 June 2016.
9. D'Angelo, G.; Ferretti, S.; Ghini, V. Multi-level simulation of Internet of Things on smart territories. *Simul. Model. Pract. Theory* **2017**, *73*, 3–21. [CrossRef]
10. D'Angelo, G.; Ferretti, S.; Ghini, V. Distributed Hybrid Simulation of the Internet of Things and Smart Territories. *Concurr. Comput. Pract. Exp.* **2018**, *30*. doi:10.1002/cpe.4370. [CrossRef]
11. Markel, T.; Brooker, A.; Hendricks, T.; Johnson, V.; Kelly, K.; Kramer, B.; O'Keefe, M.; Sprik, S.; Wipke, K. ADVISOR: A systems analysis tool for advanced vehicle modeling. *J. Power Sour.* **2002**, *110*, 255–266. [CrossRef]
12. Fortino, G.; Russo, W.; Savaglio, C. Agent-oriented modeling and simulation of IoT networks. In Proceedings of the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), Gdansk, Poland, 11–14 September 2016.
13. Fortino, G.; Russo, W.; Savaglio, C. Simulation of Agent-oriented Internet of Things Systems. Available online: <http://ceur-ws.org/Vol-1664/w2.pdf> (accessed on 19 October 2019).
14. Fortino, G.; Gravina, R.; Russo, W.; Savaglio, C. Modeling and Simulating Internet-of-Things Systems: A Hybrid Agent-Oriented Approach. *Comput. Sci. Eng.* **2017**, *19*, 68–76. [CrossRef]
15. Looga, V.; Ou, Z.; Deng, Y.; Ylä-Jääski, A. Mammoth: A massive-scale emulation platform for internet of things. In Proceedings of the 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, Hangzhou, China, 30 October–1 November 2012.
16. NS2 Wiki Mainpage. Available online: [http://nslam.sourceforge.net/wiki/index.php/Main\[_\]Page](http://nslam.sourceforge.net/wiki/index.php/Main[_]Page) (accessed on 19 October 2019).
17. NS3 Wiki Mainpage. Available online: [https://www.nslam.org/wiki/index.php/Main\[_\]Page](https://www.nslam.org/wiki/index.php/Main[_]Page) (accessed on 19 October 2019).
18. Parallel/Distributed NS. Available online: <https://www.cc.gatech.edu/computing/pads/> (accessed on 21 October 2019).
19. GTNetS. Available online: <http://griley.ece.gatech.edu/MANIACS/GTNetS/> (accessed on 19 October 2019).
20. Sobeih, A.; Hou, J.C.; Kung, L.C.; Li, N.; Zhang, H.; Chen, W.P.; Tyan, H.Y.; Lim, H. J-Sim: A simulation and emulation environment for wireless sensor networks. *IEEE Wirel. Commun.* **2006**, *13*, 104–119. [CrossRef]
21. Krop, T.; Bredel, M.; Hollick, M.; Steinmetz, R. JiST/MobNet: Combined Simulation, Emulation, and Real-World Testbed for Ad Hoc Networks. In Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, Montreal, QC, Canada, 10 September 2007.
22. Contiki: The Open Source Operating System for the Internet of Things. Available online: <http://www.contiki-os.org/> (accessed on 19 October 2019).
23. TinyOS Wiki, TOSSIM. Available online: <http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM> (accessed on 19 October 2019).
24. Shamsi, J.; Brockmeyer, M. DSSimulator: Achieving Million Node Simulation of Distributed Systems. Available online: <https://www.researchgate.net/publication/228351353> (accessed on 19 October 2019)
25. Zeng, X.; Bagrodia, R.; Gerla, M. GloMoSim: A library for parallel simulation of large-scale wireless networks. In Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation PADS'98 (Cat. No. 98TB100233), Banff, AB, Canada, 29–29 May 1998.

26. Omnet++ Simulation Manual. Available online: <https://omnetpp.org/doc/omnetpp/manual/> (accessed on 19 October 2019).
27. Park, S.; Savvides, A.; Srivastava, M.B. SensorSim: A Simulation Framework for Sensor Networks. In Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Boston, MA, USA, 20 August 2000.
28. Chen, G.; Branch, J.; Pflug, M.; Zhu, L.; Szymanski, B. SENSE: A wireless sensor network simulator. In *Advances in Pervasive Computing and Networking*; Springer: Santa Clara, CA, USA, 2005; pp. 249–267.
29. Emulab. Available online: <https://www.emulab.net/> (accessed on 19 October 2019).
30. ATEMU. Available online: <http://www.hynet.umd.edu/research/atemu/> (accessed on 19 October 2019).
31. Brambilla, G.; Picone, M.; Cirani, S.; Amoretti, M.; Zanichelli, F. A Simulation Platform for Large-scale Internet of Things Scenarios in Urban Environments. In Proceedings of the First International Conference on IoT in Urban Space, Rome, Italy, 27–28 October 2014.
32. Diffie, W.; Hellman, M. New Directions in Cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654. [CrossRef]
33. Rivest, R.L.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
34. Paillier, P. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques, Prague, Czech Republic, 2–6 May 1999.
35. Wainer, G.A. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*; CRC press: Boca Raton, FL, USA, 2009.
36. Kölsch, J.; Ratzke, A.; Grimm, C. Co-Simulating the Internet of Things in a Smart Grid use case scenario. In Proceedings of the 2019 7th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), Montreal, QC, Canada, 15 April 2019.
37. Kölsch, J.; Ratzke, A.; Grimm, C.; Heinz, C.; Nandagopal, G. Simulation based validation of a Smart Energy Use Case with Homomorphic Encryption. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019.
38. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory (TOCT)* **2012**, *6*, doi:10.1145/2633600. [CrossRef]
39. Halevi, S.; Shoup, V. HELib. Homomorphic-Encryption Library. Available online: <https://github.com/shaih/HElib> (accessed on 19 October 2019).
40. Tiehuis, M. libhcs. Available online: <https://github.com/tiehuis/libhcs> (accessed on 19 October 2019).
41. Ronacher, A. Python Flask. Available online: <https://palletsprojects.com/p/flask/> (accessed on 19 October 2019).
42. Giambiasi, N.; Escude, B.; Ghosh, S. GDEVs: A generalized discrete event specification for accurate modeling of dynamic systems. In Proceedings of the 5th International Symposium on Autonomous Decentralized Systems, Dallas, TX, USA, 26–28 March 2001.
43. Kim, H.; Ben-Othman, J. A Collision-Free Surveillance System Using Smart UAVs in Multi Domain IoT. *IEEE Commun. Lett.* **2018**, *22*, 2587–2590. [CrossRef]



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).