An Open-Source Extendable, Highly-Accurate and Security Aware CPS Simulator

Nikolaos Tampouratzis*, Antonios Nikitakis†, Andreas Brokalakis†, Stamatis Andrianakis*, Ioannis Papaefstathiou†, Apostolos Dollas*

*Telecommunications Systems Institute, Technical University of Crete, Greece
†Synelixis Solutions Ltd, Chalkida, Greece

Abstract—**In this paper, we present an open-source Cyber Physical Systems (CPS) simulation framework that aims to address the limitations of currently available tools. Our solution models the computing devices of the processing nodes and the network that comprise the CPS system and thus provides cycle accurate results, realistic communications and power/energy consumption estimates based on the actual dynamic usage scenarios. The simulator provides the necessary hooks to security testing software and can be extended through an IEEE standardized interface to include additional tools, such as simulators of physical models.**

I. Introduction

Nowadays, Cyber Physical Systems (CPS) are growing in capability at an extraordinary rate, incorporating computing systems that vary from simple microcontrollers to high performance units connected with each other through a multitude of networks. One of the main problems CPS designers face is the lack of simulation tools that can offer realistic insights beyond simple functional testing, such as the actual performance of the nodes, accurate timing, power/energy estimations and network deployment issues. On top of that, none of the existing solutions provide security testing to assess potential problems at design time.

In this paper, we present the COSSIM Simulation Framework, an open-source CPS simulation framework that aims to address all the aforementioned limitations. The proposed solution integrates a series of software packages that model the computing devices of the processing nodes and the network that comprise the CPS system. It provides cycle accurate results by simulating the actual application and system software that is executed on each node of the CPS, realistic communication modeling of the different networks that are employed and power/energy consumption estimates for both the processing elements and the network based on the actual dynamic usage scenarios. The simulator provides the necessary hooks to security testing software, making it possible to determine vulnerabilities as well as examine the robustness of the system under design. Last but not least, by employing a standardized interconnection protocol between its components (IEEE 1516.x HLA[1]), the framework can be extended to include additional tools, such as simulators of physical processes.

[1]IEEE 1516.x: Standard for Modeling and Simulation High Level Architecture (HLA).

II. CPS Tools and the COSSIM Approach

The available tools for CPS simulation fall under two main categories. The first one deals with the actual design of a CPS system from a mostly functional perspective. As such, they try to model different entities in a CPS system: a physical process, an electromechanical physical component, user behavior, events, message exchanges between components etc. They are mostly based on the use of well-defined models of computation. Among those tools the most profound ones are Ptolemy [1], Matlab Simulink [2] and Modelica[2]-based simulation environments. While these tools can model physical processes, they can only be used during the initial design of the CPS application (they can even generate application code) since they offer only functional simulation as a proof of concept. They do not handle cycle-accurate processing simulation with power estimation of the actual components (e.g. a microcontroller), the source code that is actually executed on those devices (including OS, libraries, drivers and other system software components) nor the actual networks that are going to be employed.

Simulators that do handle those aspects can typically be found in the field of Wireless Sensor Networks (WSNs), which has close affinity with the CPS one. Most of these tools are designed around a specific WSN-related operating system or a specific device. For example, TOSSIM [3] can simulate WSN applications designed for the TinyOS operating system; it can simulate the whole OS stack including the nesC applications while offering models for specific microcontrollers (AVR ATmega128) along with its peripherals. Similarly, COOJA [4] is built to simulate sensor networks whose nodes run the Contiki operating system. While it can simulate the actual software stack that is executed, it can only handle specific network protocols (IEEE 802.15.4) and the software it supports has to be compiled for the MSP430 microcontroller. Other WSN simulators [5], [6] focus on specific microcontrollers or motes

[2]Modelica is an object-oriented, equation based language used to construct models of systems. Multiple free or commercial simulation environments that can import Modelica models exist such as OpenModelica, JModelica, CyModelica and even other environments such as Simulink can import Modelica models.

(they can model the whole sensor node including sensors and the radio communications chip). These simulators can effectively model both the actual software executed in hardware and the network between the devices, however their applicability cannot be generalized neither in software (support for different operating systems, different software packages), nor hardware (support for other devices besides the ones supported) and network (they can only simulate a specific type of wireless protocol). As such, they are of limited usefulness for a general CPS simulation environment that may require the modeling of very diverse devices and a multitude of networks.

From the above, it becomes apparent that no integrated solution exists that can handle the simulation of an actual CPS system, including its complete software stack and network dynamics. For COSSIM, an approach that combines several well-established tools, one for each specific task, has been chosen. The goal is to bind them tightly together in a single framework that works transparently for the user (as if it were a single tool rather than a framework) and offers functionality greater than what can be achieved by using each component separately.

III. COSSIM Framework

A. Processing Simulator

A CPS is comprised of a set of nodes and a number of networks that connect those nodes together. The diversity of nodes used in CPSs is large; there can be simple microcontrollers that control an actuator device or provide readings from a sensor, network devices, powerful main control units, server systems and so on. Thus, to accurately simulate a CPS node that can vary as much, a system simulator that is cycle-accurate, Instruction Set Architecture (ISA) independent, configurable (in terms of supported devices and system features), able to boot real-world operating systems and execute software compiled for them is required. GEM5 [7] is a computer system simulation platform that can cover almost all the aforementioned requirements. It can simulate single or multi-core homogeneous or heterogeneous systems including their peripherals. The main engine of the simulator is ISA agnostic and it can be configured to support a broad range of ISAs. GEM5 does not support natively any power/energy estimations, however it can be integrated with other tools to provide this functionality. The most common power estimator used in tandem with GEM5 is McPAT [8].

B. Network Simulator

GEM5 can provide system simulation up to the level of the Network Interface Card (NIC). It does not support network modeling. Therefore, COSSIM employs a dedicated network simulator that handles all network related modeling from the physical layer of a NIC and beyond. The simulator chosen for the task is OMNET++ [9]. Through OMNET++ different network protocols and topologies can be supported and a realistic network behavior of the CPS system can be modeled (devices such as bridges, switches, routers that are part of the infrastructure rather than the CPS system developed can also be modeled to increase accuracy). Through OMNET++'s extensions, it also possible to estimate the power consumed at the network as well as the radio devices (i.e radio transceiver power consumption) of the nodes (MiXiM add-on [10]). MiXiM can also model the radio wave propagation, interference estimation with respect to the 3D space of the physical environment.

C. Integration of Components

Binding the aforementioned processing and network simulators together is not trivial, as it requires carefully designed communication interfaces and synchronization schemes. These bidirectional interfaces have to pass information on the type and timing of events and to provide a common data representation, throughout the framework. To achieve this task, COSSIM employs the IEEE High Level Architecture (HLA) [11]. HLA is a general purpose software architecture specifically designed for the development and execution of large distributed simulation applications. It determines the functional entities, design rules and interfaces for computer-based simulation systems and specifies the communication between the individual components. It requires a runtime infrastructure (RTI) to perform tasks such as synchronization and simulation control. Among the numerous RTI implementations, COSSIM has adopted CERTI [12].

IV. Implementation

Figure 1 demonstrates the COSSIM simulator with all its components and interfaces. Multiple instances of a node simulator module (GEM5) cover the processing nodes of a CPS. The network that binds together the different CPS nodes is simulated by the network simulation module (OMNET++). The GEM5 instances are connected with the OMNET++ simulator through IEEE HLA compliant interfaces (Interface #1). Additionally, each GEM5 instance is interfaced through a custom XML interface with a McPAT instance in order to estimate the energy and power consumption of each node, while the OMNET++ employs internally the MiXIM add-on to estimate the power consumption of the network. Both GEM5 instances and OMNET++ have been properly modified to provide hooks that allow security software components to interact with the simulation process so that security tests can be performed (interfaces #3, #4, #5).

Figure 1 illustrates the primary inputs that a user has to provide to the overall system and the primary outputs of the simulation process. Specifically, *System Configuration* (number of CPUs, memory sub-system, peripherals etc.) have to be defined either using a configuration file or the supplied graphical interface. Furthermore, *image files* of the OSes that will be executed on the nodes of the simulated platform have to be provided by the user. An image file includes the OS kernel, the *application executable*[3] and all the libraries that are required. Finally, through a *Network & Topology description*, the user can define a network topology (distance

[3]The simulator includes a number of preassembled linux-based OS images in which the user can execute C/C++ and Java applications.

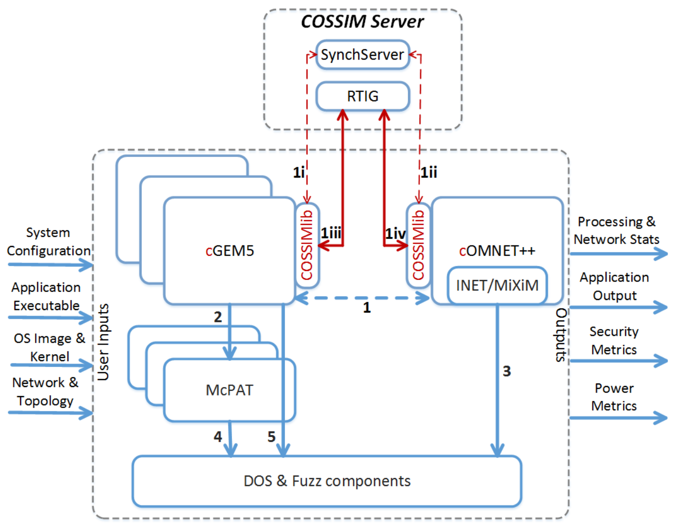


Fig. 1. Top-level view of the COSSIM framework

between nodes, topology, mobility between nodes, etc.) as well as describe channel definitions and network protocols (noise level, bandwidth rate, Ethernet, Wireless, GSM, etc.).

Upon completion of a simulation, the COSSIM simulator provides a number of outputs. *Processing & Network* statistics contain all statistics of an execution run (clock ticks, real time execution, cache misses, number of packets sent, number of packet drops, delay of received packets, peak throughput, etc.). The *output of the actual application* that is executed is also provided. When security tests are performed, *security metrics* (increase in response time, rejection rate, vulnerability density, system robustness, etc.) are reported as well. Finally, power and energy estimations for each node (including further details such as peak power, runtime dynamic power, leakage etc) and for the network are provided.

As mentioned above, interface #1 is responsible for the integration of the processing simulation instances and the network simulator through the HLA framework. In practice, the HLA compliant interfaces that have been added to the GEM5 and OMNET++ simulators perform both interconnection and synchronization functions. Interface #1 consists of four parts per node; (i) two for RTIG (CERTI HLA server) to exchange Data Packets and synchronize the Processing with the Network simulator and (ii) two for *SynchServer* which is proposed by the authors to initialize the HLA Federates.

COSSIMlib enables the interoperability between GEM5/OMNET++ and CERTI/HLA Federation. It is built on the top of the CERTI API, a C++ binding of CERTI based on HLA 1.3 version. Since the HLA communication mechanisms are based on TCP/IP packets, both RTIG and SynchServer can be executed either in the same physical machine or in a remote server. The same applies for all components of the simulator (each GEM5 instance and OMNET++), thus enabling distributed simulation to increase parallelization when simulating large CPS systems.

A. GEM5 Adaptation for COSSIM Integration

In order for the GEM5[4] to serve as the processing simulation sub-system in the COSSIM framework, it has to be properly extended to support certain features. Specifically, in order to simulate a CPS node, it must support the simulation of a CPU including several levels of memory hierarchy and complex peripherals (such as network cards, accelerators or other components), as well execute a full operating system on top of the simulated system. At the current state of development, cGEM5 can support single and multi-core *ARM* and *X86* architectures, *real network cards* and a complete *TCP/IP protocol stack* included in a Linux-based Operating System *Kernel* module with the appropriate drivers. However, GEM5 can execute safety-critical systems (such as RTOS) [13].

In the following subsections the limitations of the current, publicly available, version of GEM5 are described in tandem with the modifications and extensions that have been implemented in order to alleviate those restrictions.

1) Extending the Network Interface Card Support: In GEM5's publicly available repositories, the only network interface card implemented, tested and verified is the Intel 8254x based gigabit Ethernet adapter. It is provided as a PCI GEM5 network device using the e1000 Linux driver.

However, the latest version of GEM5 supports this real-network device only on ARM-based architectures. GEM5 for x86 ISA has been modified to support a custom-made network card [14]. On top of that, proper drivers to support it have been built; it should be noted though that the available drivers required Linux kernel 3.x support and therefore such a kernel had to be custom built since the GEM5 repositories only offered Linux kernel 2.x for x86 simulated systems.

2) Extending the Network Model of GEM5: In addition to the network interface cards, GEM5 supports networking through a simple Etherlink device. Etherlink is a virtual dummy link which emulates a cable over which Ethernet packets are sent and received without any delay (no switching or routing functionality is implemented).

In the scope of COSSIM, these limitations (only a single NIC and a single type of network) are unacceptably restrictive. Therefore Etherlink could not be used in its current form at all and it had to be modified, while NICs supporting more protocols have to be developed. Since device models are not easy to develop without inside information from their manufacturer (the Intel NIC model used in GEM5 has been contributed by Intel itself), in order to support a varying number of physical networks, the COSSIM approach is to tap the Ethernet packets from Etherlink and send them to a Networking Simulator configuring at the same time the NIC (i.e. Intel 8254x) according to the specific network protocol required by the CPS application (i.e. Ethernet, WiFi, 3G, etc).

To achieve the aforementioned objectives the CERTI HLA interface [12] has been employed. Specifically, *COSSIMlib* has been integrated to the main core of the GEM5 system through Etherlink. *COSSIMlib* is a wrapper to an RTI Ambassador

[4]The adapted version GEM5 for COSSIM will be referred as *cGEM5*

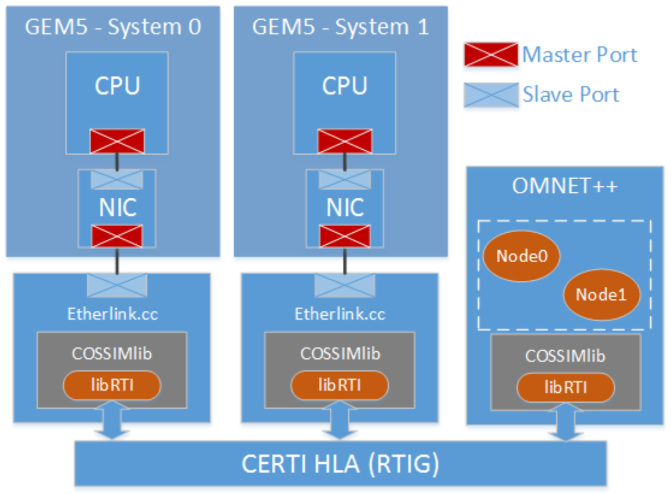


Fig. 2. COSSIM system interconnection

Class serving to exchange messages over the network with the HLA Server (RTIG process) via TCP (and UDP) sockets. *COSSIMlib* exchanges Ethernet Packets captured from the Etherlink Device and sends (and accordingly receives) them to (from) the HLA Server (RTIG). The HLA Server forwards these messages to a proper interface of a Network Simulator that implements all the network related functionality (NIC physical layer and actual network). Figure 2 provides a visualization of the implementation.

3) Extending Support for Multiple Systems Simulation: The simplistic network model of GEM5 has another serious limitation. It only supports the simulation of *two identical networked systems* (for example two identically configured ARM processors with exactly the same peripherals and memory configuration [15]). Furthermore, the simulation of both systems is executed within the same thread, thus a serious performance penalty occurs while no synchronization primitives between the two systems are provided.

By using HLA-powered GEM5 interfaces combined with a network simulator as described in the previous subsection, a natural connection between the different GEM5 systems is taking place. Each GEM5 instance models a single node and different GEM5 instances are connected through a simulated network (more precisely through HLA links and a network simulator). As a result, all the following limitations of a conventional GEM5 simulation can be overcome:

- there are no limitations for identically configured systems, as would be the case if a single instance of GEM5 was used.
- there are no limitations on the number of GEM5 systems that participate in the network. Therefore, the overall system can be scaled to support as many CPS nodes as required.
- parallelism can be extracted at the process level, since multiple GEM5 instances can be run in parallel in a multi-core physical machine. Furthermore, as CERTI HLA functions over IP, the different GEM5 instances do not even need to be on the same physical machine and the overall COSSIM simulator can thus be considered a distributed implementation.

The parallel nature of this implementation requires a mechanism to synchronize the different GEM5 instances and the network simulator. The need for synchronization arises from the non-fixed notion of time. GEM5 is an event driven simulator that schedules operations (events) on clock ticks. As a result, different GEM5 instances modeling different systems and applications will require varying amounts of time (as in wall-clock time) to reach certain application milestones and the notion of actual time in those systems will be different. If these different systems are connected through a network, communication through actual network protocols cannot be achieved, as ordering or other time-related functionalities cannot be accomplished. For this reason, the COSSIM simulator synchronization is achieved through CERTI HLA as described in Section 4.4.

B. OMNET++ Adaptation for COSSIM Integration

The network sub-system of the COSSIM simulator is designed to model all the network related behavior of each simulated node along with all the lower network protocols. The upper protocol stack (from Layer 2 and above) will be accurately simulated from the processing sub-system.

In this respect the COSSIM approach requires each simulated node to consist of two parts, the processing and the network simulation one communicating through the HLA framework. Those nodes are called HLA Enabled nodes inside OMNeT++ as it is possible to have in the same simulation conventional nodes without connection with the processing simulator.

Each of the HLA Enabled Nodes has a minimum functionality that allows them to communicate with their counterpart on the cGEM5 side. This communication is established through an HLA run-time infrastructure (RTI) wrapper.

More specifically a CERTI-HLA compliant wrapper was developed, offering a unique interface to each node simulated in the network subsystem in order to communicate consistently and synchronized with the processing subsystem of the COSSIM simulator. Figure 3 presents an overview of the OMNeT++ top level architecture augmented with two HLA-enabled nodes. These nodes communicate (via the HLA sockets) transparently with the processing simulator. All the added functionality was deployed in the user space to assure 100% compatibility with OMNeT++ and its adopted libraries (e.g INET). As mentioned earlier in order for the COSSIM simulator to increase the simulation accuracy the upper protocol stack of the network should be simulated in the processing sub-system (that is cGEM5). On the other hand the network sub-system should be able to forward network packets in the data link layer (L2) or in the network layer (L3) in order for other aspects of the network to be modelled (e.g packet latency). Furthermore OMNeT++, like any other network simulator (e.g ns2), does not send the actual payload data (from the application layer) across the simulated network

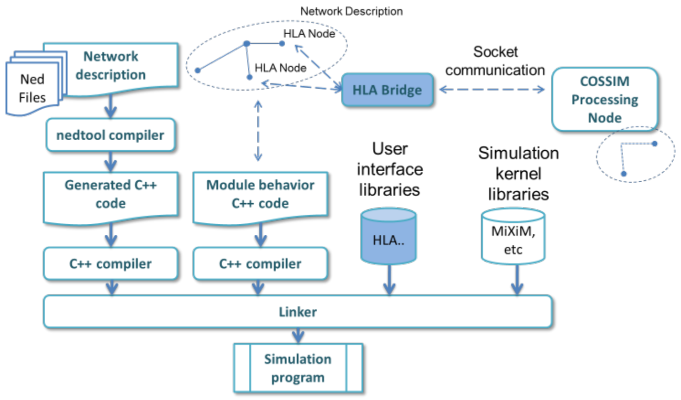


Fig. 3. COSSIM network subsystem (two HLA-enabled nodes are illustrated)

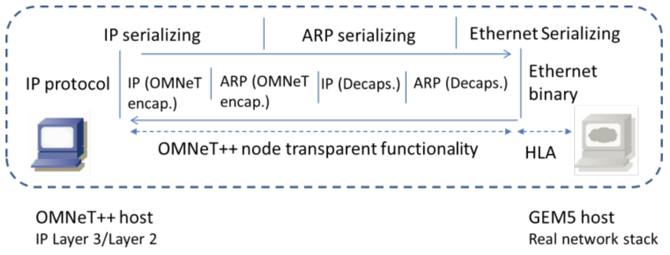


Fig. 4. The En/De-capsulation vs serialization process inside OMNET++ HLA nodes

to gain performance. Usually this kind of information is not evaluated as most of the time the Source/Destination and Payload length are crucial for the latency estimation.

In the case of COSSIM, since the goal is to provide cycle-accurate simulation, the whole protocol stack is executed, thus producing a fully RFC-compliant binary IP packet. In order for these packets to be forwarded, they have to be properly encapsulated inside the OMNeT++ L2/L3 packet structure.

The first step towards this direction was to modify the standard OMNeT++ cPacket structure to include payload data. The second step was to develop a custom-fit functionality that will be automatically inherited in each of the HLA enabled nodes of the simulation in order to seamlessly convert the cGEM5 packets to OMNeT++ packets and vice versa. Each packet sent from the cGEM5 subsystem into the OMNeT++ passes through a sequentially de-capsulation procedure (i.e parsing) from the Ethernet / L2 level to the L3 level (the IP level) followed by an encapsulation procedure in the OMNeT++ protocol stack. In the opposite direction (from the OMNeT++ to the cGEM5) all the L2/L3 fields had to be properly de-serialized into a single RFC-compliant binary packet (a valid Ethernet packet) that will form the payload for the HLA channel. Figure 4 summarizes this procedure.

All the aforementioned functionality is abstracted from the user as it is built as a shared object that can be automatically linked in any simulation scenario that demands precise processing simulation.

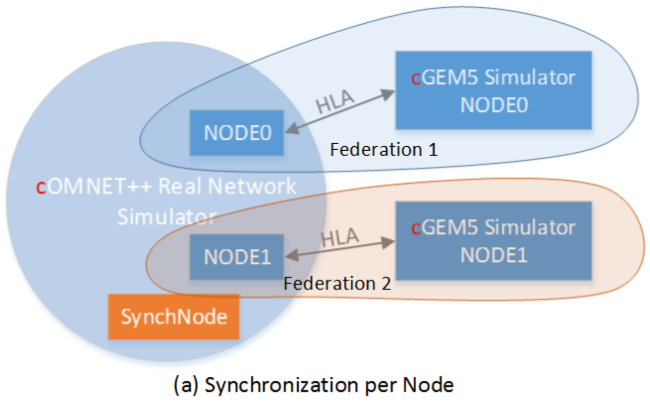
Furthermore, the OMNeT++ functionality was extended in terms of the GUI in order to integrate the capabilities of the COSSIM tool for precise processing simulation. This GUI has the form of a wizard which is installed as a plugin in OMNeT++ and guides the user through the GEM5 configuration process for each of the simulated nodes. The GEM5 configuration process is a very time-consuming one as it is usually performed through the command line and needs a large number of parameters to be set for each of the nodes. With the use of the wizard we roughly calculated a 90% reduction for the configuration time for just a 10 node system. In addition, our sophisticated GUI prevents the user from setting wrong various parameters and thus minimizing risk of starting a time-consuming simulation that will latterly will be proven wrong or inadequate.

C. Synchronization Between COSSIM Modules

COSSIM simulator synchronization is achieved through CERTI HLA in two stages:

1) ***Synchronization per node*** Each node simulator needs to communicate, in a consistent way, with its representation in the network simulator to exchange data packets. This type of synchronized communication is necessary because network data between the two simulators must be exchanged while preserving the exact same time ordering. For this reason, one federation is created per node to achieve the same synchronization time as illustrated in the upper part of Figure 5. The user can define the minimum simulated time in which the two simulators can receive Data Packets.
2) ***Global Synchronization*** The COSSIM simulator needs to periodically synchronize all nodes. This is because it supports different types of CPUs with potentially different clock cycles and/or different network protocols, all resulting in varying workload for the simulators engine. Therefore, the simulated time (the timing of the modelled system) in each node can be completely different given the same real-time (the simulation time). For this reason, a Global Synchronization federation is created to achieve a unified notion of time which contains all cGEM5 nodes and one OMNET++ helper Node (SynchNode) as illustrated in the lower part of Figure 5. The user can define the simulated time in which all COSSIM instances are synchronized periodically. The SynchNode is also a normal user-space instantiated node (as the rest of the HLA Enabled Nodes) inside the OMNeT++ simulator that follows the standard Node structure and as a result it is 100% compatible with OMNeT++.

The *Synchronization time per node* and the *Global Synchronization time* are two different entities that can be separately defined by the user. The first one is mostly defined by the latency of the network interface and it doesnt constrain the simulation speed while the second is a trade-off between simulation speed and simulation accuracy.



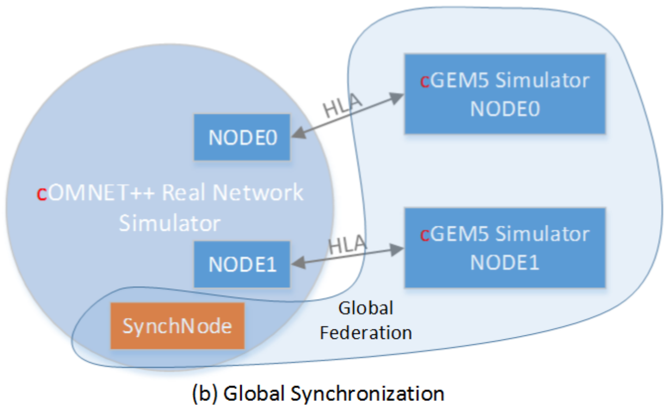


Fig. 5. COSSIM HLA federations

The proposed global synchronization scheme also functions as a way to preserve the cycle-accurate notion of the simulation process. OMNET++ is natively an event driven simulator, however by employing the global synchronization scheme, it becomes hooked to the ”cycle-events” of each of the GEM5 simulated nodes. This not only prevents the clocks of all the nodes from any drift but also implicitly ”forces” the OMNET++ to act like a ”cycle-driven” event simulator. In this respect every component of the simulated CPS acts within the same notion time (i.e. clock cycles).

D. *Integration of Security Tools*

The COSSIM simulation framework integrates a number of security tools that allow the CPS designer to evaluate the security of the system simulated. COSSIM's security module consists of three main sub-modules; the *DoS Testing System* (DTS), the *Fuzz Testing System* (FTS) and the *Metrics Management* (MM). The DTS is responsible for testing of the resilience of a CPS under simulation against various types of denial-of-service (DoS) attacks [5]. As DoS is a network attack, DTS is tightly connected with the network simulation components of the COSSIM framework. The FTS provides automated testing of CPS component interfaces for discovering vulnerabilities. It produces input test vectors that are fed in a CPS application under test. This process (fuzzing) is capable of exposing errors that arise as a result of the processing of these input vectors. Fuzzing is not only used for security testing; it is also employed in quality assurance for evaluating a systems robustness aiming to increase the quality of the software under test. The MM observes the state of the simulated CPS to calculate various types of security metrics, allowing the systems developers to assess the systems behaviour in certain types of situations.

[5]One of the main goals of a DoS attacker is to make a system inaccessible to legitimate users without actually getting access to that system.

E. *Simulation of Physical Parts of A CPS*

The COSSIM framework can simulate the computation along with the networking aspects of a CPS in a holistic approach. Since this creates a bias towards the cyber part rather than the physical part of a CPS system, COSSIM provides two main options to integrate simulation of components of the physical world. The first one, extends GEM5 by adding support to include sensor devices in the processing system. GEM5 provides a Memory Bus to interconnect all architecture components such as CPUs, Caches, RAMs as well as I/O devices using master and slave ports. In COSSIM, each sensor is connected through programmed I/O using one memory bus master port to read the sensor values. In order to incorporate efficiently each GEM5 with the sensors a set of device drivers has been developed, while an *ioctl* [16] function was developed to achieve efficient user-kernel space communication.

In addition, the COSSIM framework is able to seamlessly interconnect with other open-source approaches simulating physical aspects of the system (such as Ptolemy II [1]) through the CERTI HLA interface. Specifically, our tool could potentially connected with any other tool that has the IEEE stadardized HLA interface in order (for each processing node) to have access to the physical processes that ”generate” the sensing data. On the other hand, regarding the physical aspects that relate to the network performance, those are already included in the tool through the OMNET++ simulator which offers an extension that models the physical world regarding the reaction between the physical and network processes. For example we could model the movement of the nodes in the space, obstacles in the wave propagation, attenuation, noise, re-transmissions etc.

V. Validation and Performance Analysis

Within the context of the COSSIM EU Project [17], the COSSIM Simulation Framework has been validated using two real-world applications with very different requirements and characteristics. The first application is a Visual Search application in which mobile nodes dispatch image descriptor data to a main server that performs the main image processing and analysis functions. The application is written in C/C++ and uses Linux-based OSes for both mobile nodes and the server system. In the second application, a multi-tiered smart-building system is designed. The Building Management System (BMS) is comprised of sensor nodes, intermediate processing and networking nodes and server systems. The application mostly uses Java on top of Ubuntu and Gentoo Linux OSes.

The validity of the application results that the COSSIM simulator produces has been tested against actual native implementations. In the Visual Search case, a single client - server implementation has been used, using an Odroid XU3 platform as a mobile node and an x86 multicore system as a server. For the BMS case, the experimental infrastructure of the KUBIK building has been employed [18]. In both cases, the COSSIM Simulator provides the exact same application results as the native ones, thus proving the correctness of operation of the simulator.

The performance of COSSIM simulation framework has been analysed using light versions of two Linux distributions; Gentoo Base System (v1.12.11.1) and BusyBox (v1.15.3) for X86 and ARM processors respectively. In both systems Ubuntu-minimal package and JRE7 are installed so as to enable execution of C, C++ and Java applications (thus resembling a realistic deployment scenario). All experiments illustrate the simulation time (and instructions per seconds) required to boot the OSs with their network card configuration.

COSSIM uses a modified version of the mainstream GEM5 simulator. However all modifications are mainly related to the Ethernet interface, hooks to HLA components and finally the checkpointing system used by the security tools. As such, the main computation engine of the GEM5 simulator is left intact and therefore the performance of the GEM5 component of the COSSIM simulator is in line with the reported performance of the publicly available version of the simulator. GEM5s performance varies greatly with the complexity of the system that is being simulated. A typical simple CPU (InOrder CPU) can be simulated at a rate of 1 to 3 MIPS (Million Instructions / Sec). More complex CPU structures (OoO CPU) and memory subsystems can reduce the rate of simulated instructions to as low as 0.1 - 0.3 MIPS [19], [20].

The key concept of COSSIM's approach is to execute the OMNET++ simulator in a typical workstation due to the GUI that facilitates the orchestration and visualization of the simulation, while the GEM5 instances are run in one or more servers (distributed simulation). For this reason, OMNET++ simulation was carried out on a laptop based on an Intel i7-3632QM[6] processor (2.2GHz) running Ubuntu Linux 14.04 with 16GB of RAM (Machine 1). Simulation of GEM5s as well as SynchServer with HLA Server were run on a server based on two Intel Xeon E5-2430v2[7] processors (2.5GHz) running CentOS 7.2 with 64GB of RAM (Machine 2). The main configuration of the simulated processors is described in Table I.

Figure 6 illustrates the instructions per second that are simulated for both X86 and ARM processors under the deployment scenario described above. It demonstrates that as the number of nodes increases, the rate of simulated instructions (thus performance) decreases. This is mainly attributed to Global Synchronization. For the specific experiments presented here, the synchronization interval is set at 10ms. Consequently, after every 10ms of simulated time, all nodes are halted so that they can be synchronized with each other and then resume operation. Apparently, this synchronization interval determines the accuracy of captured events. A short synchronization interval will yield the most accurate results at the cost of significantly lower overall performance, as each simulation instance will be forced to halt very frequently and the time required for synchronizations will become significant compared to the actual computation time. A more relaxed (i.e. longer) synchronization interval will result in no practical slowdown, with a potential loss of accuracy, as interactions between nodes that should happen in this interval are not properly handled. The CPS designer should therefore carefully select this interval. As a general guide, when an application is sensitive to frequent irregular events or employs very fast network technologies, then a small synchronization interval is required for accurate results. When a CPS application involves more regular events and higher latency communication mechanisms, then a longer synchronization interval may be used.

[6]i7-3632QM is a 4-core CPU with 2 threads per core.

[7]Xeon E5-2430 is a 6-core CPU with 2 thread per core.

TABLE I
THE MAIN CONFIGURATION OF THE PROCESSORS SIMULATED

CPU Type	**CPU/System Clock**	**Memory**	**L1I/L1D/L2/L3 Cache Size**
Atomic (In Order)	2GHz/1GHz	DDR3 (512MB)	32KB/64KB/2MB /16MB

Figure 7 demonstrates the effect of the synchronization interval in the performance of the system, as measured by the overall (wall-clock) time required to complete a simulation with 4 nodes (all nodes are configured as in Table 1 for ARM ISA). There is a dramatic drop in performance for very short interval times (smaller than 10ms). It should be noted that in the proposed setup of a distributed simulation, the exchange of HLA synchronization messages is expected to introduce some further delays as those messages have to travel through actual networks with non-negligible latency to reach the different physical machines. Figure 8 demonstrates the performance of the simulator when all COSSIM components (OMNET++, HLA servers and all GEM5 instances) are executed on the same physical machine (machine 1) and when they are executed in the distributed scenario described above (machines 1 and 2).

As can be seen from the Figure 8, the effect of the network has a non-measurable impact on the overall performance (consider only the performance up to 4 nodes) the distributed system is actually faster since each core of machine 2 has a slightly faster single thread performance. From the same figure though, as well as from Figure 6, it becomes evident that performance is heavily impacted when the number of GEM5 instances spawned becomes higher than the number of physical processor cores present in the simulation machine (there is a steep performance drop when more than 4 GEM5 instances are executed on the 4-core machine 1 and more than 12 GEM5 instances are executed on machine 2).

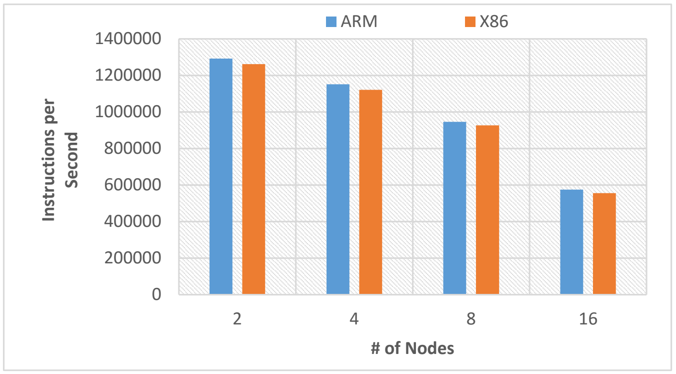


Fig. 6. Performance results using typical GEM5 configuration

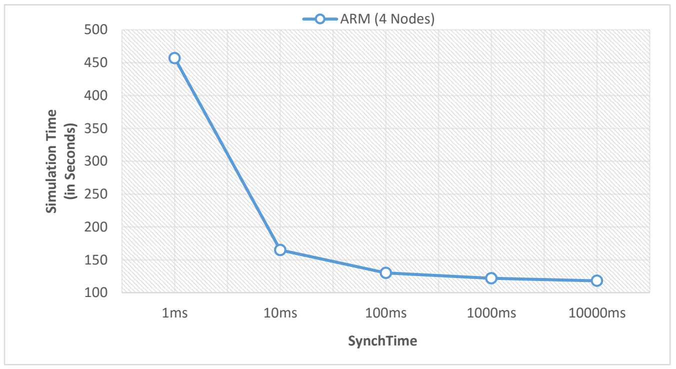


Fig. 7. COSSIM simulation time using different synchronization intervals

VI. Conclusions and Future Work

In this work, we present COSSIM Simulation Framework, an open-source CPS simulation framework that aims to address in a single integrated solution both the processing and the network part of a CPS while taking into account both security and power aspects of the nodes. By using a novel dual-stage synchronization scheme it always assures accurate simulation, distributed processing and extensibility.

As a future work, we are working on providing a comprehensive set of accurate processor descriptions in GEM5 and McPAT so as to enable easy and accurate out-of-the-box simulation experience for a variety of commonly available systems. To extend the functionality of the Simulation Framework, we aim to integrate COSSIM with a physical process simulator like Ptolemy by using the already established HLA hook. Last but not least, we have already started work on accelerating the framework through the use of FPGA devices. At this stage, we have focused on accelerating the power/energy estimation processes.

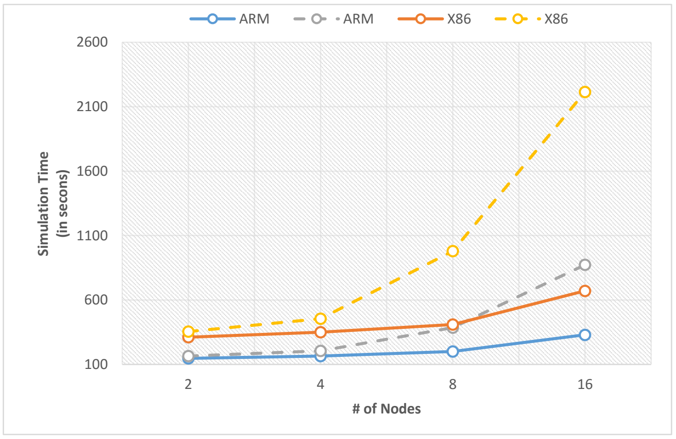


Fig. 8. COSSIM simulation time (dashed lines describe the case of all GEM5s and OMNET++ residing on the same physical machine, straight lines are for the distributed simulation scenario)

Acknowledgement

This work is supported by the European Commission in the context of the H2020 COSSIM project (#644042).

References

[1] The ptolemy project. [Online]. Available: http://ptolemy.eecs.berkeley.edu/
[2] Model-based design of cyber-physical systems in matlab and simulink. [Online]. Available: https://www.mathworks.com/discovery/cyber-physical-systems.html
[3] Tinyos simulator. [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM
[4] Cooja simulator. [Online]. Available: http://anrg.usc.edu/contiki/index.php/Cooja_Simulator
[5] Avrora - avr simulation and analysis framework. [Online]. Available: http://compilers.cs.ucla.edu/avrora/
[6] Worldsens: Development and prototyping tools for application specific wireless sensors networks. [Online]. Available: http://wsim.gforge.inria.fr/tutorials/wasp/files/wsim-tutorial.pdf
[7] The gem5 simulator. [Online]. Available: http://gem5.org/
[8] H. Labs. Mcpat an integrated power, area, and timing modeling framework for multicore and manycore architectures. [Online]. Available: http://www.hpl.hp.com/research/mcpat/
[9] Omnet++ discrete event simulator. [Online]. Available: https://omnetpp.org/
[10] Mixim simulator. [Online]. Available: http://mixim.sourceforge.net/
[11] Ieee 1516-010 - standard for modeling and simulation high level architecture - framework and rules. [Online]. Available: https://standards.ieee.org/findstds/standard/1516-2010.html
[12] Certi project. [Online]. Available: http://savannah.nongnu.org/projects/certi
[13] Execution time analysis of audio algorithms. [Online]. Available: http://repository.tudelft.nl/islandora/object/uuid:6c01692a-f5a4-4283-ae7c-9ebe8987ce26?collection=education
[14] P. Stevenson. Gem5 mailing list, 2014. x86 full system dual. [Online]. Available: http://www.mail-archive.com/gem5-users@gem5.org/msg09897.html
[15] Gem5 options python file. simulate two systems attached with an ethernet link. [Online]. Available: https://github.com/andysan/gem5/blob/master/configs/common/Options.py
[16] Wikipedia ioct function. [Online]. Available: https://en.wikipedia.org/wiki/Ioctl
[17] Cossim: A novel, comprehensible, ultra-fast, security-aware cps simulator. [Online]. Available: http://www.cossim.org/
[18] Kubik: Intelligent energy building. [Online]. Available: http://www.tecnalia.com/images/stories/Catalogos/CAT_KUBIK_EN_dobles.pdf
[19] A. Saidi, "Accelerating Simulation with Virtual Machines," 2012. [Online]. Available: http://www.gem5.org/wiki/images/c/c3/2012_12_gem5_workshop_kvm.pdf
[20] T. Carlson, "Full-System Simulation at Near Native Speed," 2015. [Online]. Available: http://www.gem5.org/wiki/images/4/4f/2015_ws_11_20150614_-_Trevor_E._Carlson_-_gem5_workshop.pptx