Simulation Module and Tools for XDense Sensor Network

João Loureiro, Pedro Santos, Raghuraman Rangarajan, Eduardo Tovar
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto
Porto, Portugal
{joflo,pjsol,raghu,emt}@isep.ipp.pt

ABSTRACT

We present a ns-3 module developed for wired 2D mesh grid sensor network systems, that resemble Network-on-Chip architectures. It has been designed to enable complex feature extraction from sensed data in realtime with distributed processing. We provide the design specifications, communication and processing delay models and a high level system model for XDense using ns-3. We validate our module by comparing its performance with a hardware implementation.

CCS CONCEPTS

•**Networks** →**Network simulations;** *Network experimentation;*

KEYWORDS

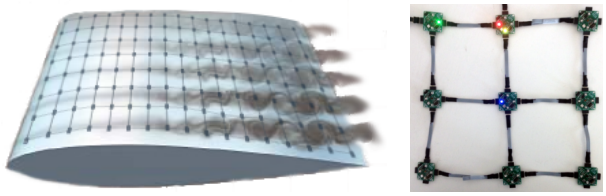
XDense, ns-3, Sensor Networks, Network-on-Chip, NoC

ACM Reference format:
João Loureiro, Pedro Santos, Raghuraman Rangarajan, Eduardo Tovar. 2016. Simulation Module and Tools for XDense Sensor Network. In *Proceedings of the 2017 Workshop on ns-3, Porto, Portugal, June 2017 (WNS3 2017),* 8 pages.
DOI: http://dx.doi.org/10.1145/3067665.3067680

1 INTRODUCTION

Thanks to current level of miniaturization on microelectronics and microelectromechanics (MEMS), cyber physical systems can now rely on dense deployments of sensors, allowing the sense of phenomena with granularity as small as few millimeters of sensor inter space and sampling rates up to kilohertz [2]. This is leading to the next generation of aerospace systems, able to sense its structural state and the environment, to effectively interpret and react to sensed data in real-time [9]. For example, sensors can be deployed on aircraft wings, to detect undesirable turbulent air-flow and enable closed-loop actuation for active-flow-control (AFC). In [7] and [5], authors survey MEMS sensors and actuator for AFC.

AFC systems have very high spatial and temporal constraints [1], and such dense sensing poses huge challenges in terms sensing, specially regarding interconnectivity and timely data acquisition and processing. Current sensor networks fail to address this requisites due to key scalability issues of cost, communication time, interconnectivity, processing time, power and reliability [6]. Wired solutions are usually based on shared buses, that lack of electrical scalability, with limited number of nodes sharing the same bandwidth. Wireless sensor networks (WSN) exceed in complexity, have finite power supply and limited bandwidth. They are also commonly susceptible to concurrency and noise issues. This has been the trigger for us to reason about a different network design that could be optimized in terms of latency.



(a) (b)

Figure 1: (a) Deployment of XDense on a wing for Active Flow Control (AFC); (b) XDense hardware prototype.

So in order to deal with the key challenges related to eXtremely Dense deployments of sensors we introduced XDense [11]. It is a sensor network composed of regular structures (nodes) interconnected in a 2D-mesh network that goes physically attached to the phenomena of interest. Figure 1a shows the deployment of XDense on a wing surface, whereas Figure 1b shows our hardware prototype. It resembles Network-on-Chip (NoC) architecture, and shares similarities in routing schemes and distributed computing capabilities [10]. Targeting AFC, we validated XDense in [13], where we perform distributed feature detection/extraction to achieve low latency real-time sampling. We "feed" nodes with data from computational fluid dynamics (CFD), and detect turbulent air-flow in real-time.

To realize XDense, we needed to examine its feasibility in many aspects, being: (i) communication and routing protocols; (ii) temporal granularity; (iii) spatial granularity; (iv) resource requirements; (v) practicality of distributed processing algorithms and how to benefit from them; (vi) scalability; (vii) accuracy. For this, we need a robust simulator for our model that is modular, to allow the development of reusable abstractions. It should be expandable, for example, for adding support to new network architectures and protocols, and allow simulating dense networks with low computational cost. We have kept portability in mind to make the simulator suitable not only to XDense, but to 2D mesh NoC in general. With this goal we developed a module for XDense on top of Network-Simulator-3 (ns-3). We provide facilities for 2D mesh networks, with configurable links, packets, communication ports, routers and applications. We use packets, routing algorithms and addressing schemes with low overhead tailored to this kind of network. We provide examples

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WNS3 2017, June 2017, Porto, Portugal
© 2017 ACM. 978-1-4503-5219-2/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3067665.3067680

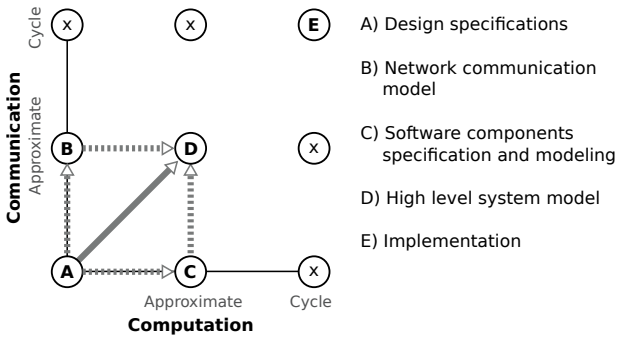


Figure 2: Modeling stages

along with the module that should serve as a starting point for custom network designs.

In this paper we present and detail each component of our module. We explain its features, from pre-processing to simulation and post-processing tools developed to enable detailed analysis.[1] We compare our simulator with the performance achieved on real hardware to validate module's temporal accuracy.

2 METHODOLOGY

Both XDense and NoCs have custom design architectures, that suite their application target. For this kind of architectures, the network performance is one of the major bottlenecks [15]. Network topology, link speed and width, communication protocol algorithms, protocols processing overhead, resource requirements and timeliness. These are all crucial aspects that must be considered during the design phase of such systems, and it is fundamental to be able to test and debug its functionalities, measure its performance and identify its limitations at design time. For so, the simulation of such systems can provide good insights on how to dimension their components. Therefore, choosing the right abstraction level and complexity of the simulation is important to achieve the right trade-off between speed and accuracy [14].

So in order to simplify the design process, the different abstraction levels can be implemented independently, as a set of intermediate models, each one with its specific objectives. Since the models can be simulated and estimated, the result of each of these design stages can be independently validated. Relating the different stages of our system model, in Figure 2 we adapt the model representation graph introduced in [4], to show the different modeling stages we adopt. The x-axis shows the time-accuracy of the computation component, and the y-axis shows the same for the communication component for each model stage. The components at each stage are quantified as un-timed, approximate-timed, and cycle-timed (defined over the next paragraphs). However, for this work, we only consider some of these stages, as follows:

(A) Design specifications: This is an un-timed component model, used to validate system's functionalities and principles of operation, without accounting for the delays associated. At this point, given the application requirements, we defined XDense's network architecture, routing and communication protocols in a simplistic implementation using event-driven simulation.

(B) Network communication model: Next stage consists of an approximate-timed model, in which delays are taken into account, but in a simplified manner, such that the computational cost of simulating the system is reduced. Queuing and communication delays are accounted at the packet-level, therefore we do not model the physical layers aspects such as electrical interference and packet transmission errors.[2] Communication baudrate and packet size are defined, so that the temporal behavior approximates the one observed on hardware implementation. Computation delays are absent at this point. At this abstraction level nodes behave like a synchronous system, since its an event-driven simulation with a single absolute time reference, with no non-deterministic delays associated.

(C) Software components specification and modeling: Here, processing delays are analyzed separate from communication delays. It is a model of the delays associated with the execution of different networking and data processing functions on the destination hardware. Each of these functions can be seen, for example, as software function of the protocol, or as a delay imposed by dedicated hardware peripheral.

This is also an approximate model, since it simply accounts a delay every time that function gets used. This delay is not fixed value, on the contrary, it is random, derived from statistical distributions that reflect measurements from hardware implementation.

(D) High level system model: At this stage we integrate (B) and (C) into a single system model, and model the delays associated with communication and processing approximately. It still has a reasonable computational cost, but is very useful in the study of protocols, and for the validation of analytical models for time predictability. The performance observed at this simulation level should be approximately the one observed on the real hardware. It enables capturing the effects of node's asynchronism due to non-deterministic delays.

(E) Implementation model: Cycle-timed models account communication delays at the bit-level. That is, each bit transmitted/received is timed and represents an event during simulation. Computation delays are accounted at the Register Transfer Level and Instruction Level. The model also accounts for delays associated with hardware peripherals. Cycle-timed models are usually expensive to compute, hard to implement and prone to bugs, and therefore unpractical[3].

We refrain from implementing this model abstraction, and instead, we perform the desired validations directly with real implementation [12], that is also useful to validate and tune the accuracy of each of the previous modeling stages.

Having defined the modeling and validation steps adopted on the development of our simulation model, in the next section we provide details on the outcome of each of the modeling steps, and the overall result.

[1] Pre and post processing tools were developed using Python, and all of this source code, as well as the simulator, is available online at **https://bitbucket.org/joaofl/noc**.

[2] The 2D mesh networks we are interested on, utilize physically very short range wired links, which are minimally subject to transmission errors, and for this reason we give low priority to transmission error modeling.

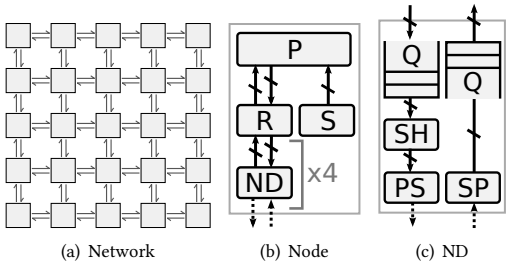


Figure 3: Overview of XDense architecture. (a) It is a 2-D mesh network; (b) Node internals: processor (P), router (R), net-device (ND) and the sensor (S); (c) net-device's internals: output queue (Q), traffic shaper (SH), and a serializer/deserializer (PS/SP).

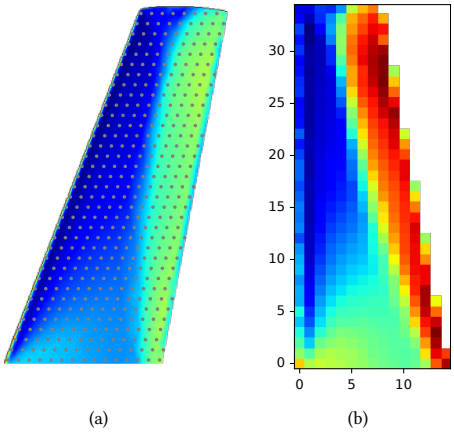


Figure 4: Pressure distribution over wing's surface (view from top). Data of a single time-frame, imported from Computational Fluid Dynamics (CFD), used as input for XDense. (a) Sensors displacement; (b) Normalized data, as seen by each sensor.

3 MODELING STAGES

3.1 Design Specification

XDense is a 2-D mesh network architecture inspired by NoC (Figure 3a shows a 5×5 network). Despite the similarities, they differ greatly in physical dimension and node count, since XDense is meant to be deployed on surfaces (like wings), and node count will tend to be much higher compared to the number of cores on actual NoCs. Each node can be seen as a self-contained system on chip (SoC), with dedicated hardware peripherals and a CPU. The node internals are shown in Figure 3b. Each node is composed of a sensor, processor, router and four communication ports (one in each direction). In the following paragraphs, we detail each of the components, detailing its implementation using ns-3.

Processor. The processor abstraction model can be seen as the application layer. It provides high level functionalities that are essential to fulfill XDense goals. In our case, the processor is connected to the sensor, and communicate over the network through the router. The processor implements different mechanisms to exchange and process sensed data between nodes. It allows nodes to request for sensed data from a single node, from groups of nodes, or from the whole network, by unicasting, multicasting or broadcasting requests for node(s)' data. The same applies for the transmission of sensed data.

Data processing algorithms are also implemented at this layer, consonant to the application scenario goals, but utilized, for example, to compress or to detect features of interest on collected data. Another functionality of the processor is the one of packet generation, utilized to set up nodes to perform periodic transmissions.

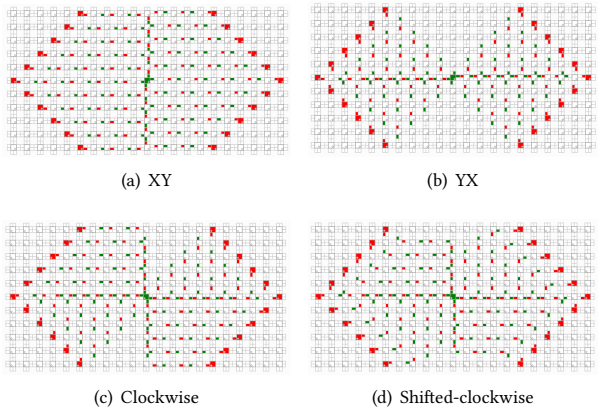
Sensor. The sensor abstraction interacts with the physical world, and is connected to the processor through it's analogue-to-digital interface. The processor can interface with any kind of sensor, consonant to the application's monitoring goal. Also, the processor should be able to accommodate applications that might require measurements of quantities of various kinds of phenomena, from more than one sensor simultaneously.

This is important in cases where the user is intended to investigate on distributed processing algorithms, when it is indispensable to have access on the expected input data (for AFC for example). That is because the efficacy of each data processing algorithm can be tightly related to the nature of the input data, and its spatial and temporal granularity. This should allow taking decision on the density of sensors deployment for example. Not only, but the nature of the data may influence on node's behavior, and therefore on network load and performance.

For this reason, we developed a sensor model that, connected to the Processor, provides it the capability of sampling temporal data. That is, we "feed" each Sensor of the network with temporal data extracted from a reliable representation of a real computational fluid dynamics (CFD) phenomena. Sensor's data are according to each node "physical" location relatively to the data set. To illustrate that, in Figure 4a we show a virtual deployment of nodes evenly distributed, superimposed by the air pressure distribution on a wing surface. It shows a single snapshot of actual temporal data from CFD simulation. More specifically, we use the SU2 integrated computational environment for multi-physics simulation to generate such data [17]. Files from CFD output are converted into a file format readable by our sensor abstraction, that provides to the processor on-demand "sensed" data.

Figure 4b shows a grid that represents our network, in which each pixel reflects a node and the corresponding data observed by its sensor. This is an intermediate results from our pre-processing tools, that are also part of our contribution. This is an interchangeable model, potentially useful to any other ns-3 module for sensor networks, in cases where the nature of sensor's data influence on the network operation.

Router. The router (R) is the interface between each networking device (ND) and the processor (P). The router is able to receive and transmit packets in parallel, from/to the processor and networking devices. Packets generated by the processor are transferred and queued at the router, that holds a dedicated queue for that. Input



(a) XY (b) YX

(c) Clockwise (d) Shifted-clockwise

Figure 5: Many-to-one scenarios, in which all nodes transmit to the node located in the center, using the different routing algorithms provided.

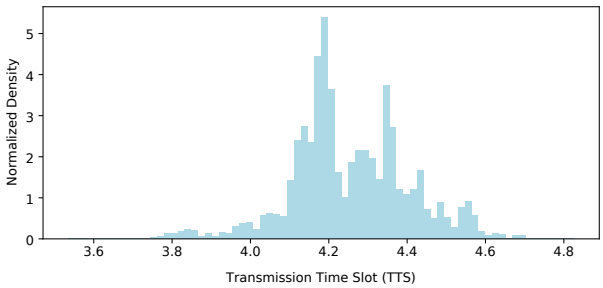


Figure 6: Single-hop internal delay distribution.

Table 1: XDense packet structure

Field	Protocol	x_o	y_o	x_d	y_d	Payload	CS
Bytes	1	1	1	1	1	10	1

queues may be also enabled, depending on the application goals. Packets may compete for output ND, in which case they are served using predefined arbitration policy. We provide implementation for both first-in-first-out (FIFO) and round-robin (RR) arbitration policies.

Routers are connected to a single processor and four NDs (one in each direction), that connect the node to the network. However, the router is extensible, and can contain multiples of four NDs, allowing it to be connected to more than one network simultaneously. This is useful for example, to simulate networks in which signaling is required for packet flow control, in a way that signals are transferred in a network apart, without interfering with the data. This implementation aims to suite more elaborate NoC architectures, [3] since XDense only uses a single network for its operation.

Packets are transmitted in the network in a multi-hop fashion from a source node to a unicast, multicast or broadcast destination. The router analyses incoming packet's headers, and applies configured routing protocols to decide where to forward the packet. We use different Dimension Order Routing (DOR) protocols known from NoC [8] in order to provide deterministic and minimal deadlock-free routes. In *DOR* protocols, all packets must follow the same order when traversing. First, the progress occurs only on one of the axis, and upon reaching the desired coordinate of the destination, (if necessary) the transfer is continued along the other axis, until reaching the destination. We provide implementation of common *DOR* routing mechanisms such as *XY* or *YX* routing [8], as well as clockwise, counterclockwise routing and other two variants of it proposed for XDense. Figure 5 shows many-to-one scenarios, in which all nodes transmit to the node in the center, using four different routing algorithms. This is an output of our visualization tool, that shows networking devices and processor's activity (red for outgoing and green for incoming packets).

[3] For example, the Epiphany processor uses three separate mesh networks to interconnect its many cores for data exchange. These are: one for "read" transactions, another one for "write" transactions and the third one for off-chip communication [16].

We use custom addressing scheme, based on the Cartesian coordinate system. It can be configure to use either absolute or relative addressing. When using absolute addressing, the origin is fixed at the lower-left corner of the network, and nodes are addressed accordingly with fixed address. When using relative addressing, all nodes see themselves as located at the origin, and transmissions between nodes are done on the basis of offsets, by specifying the number of hops that a packet should travel before reaching its destination ($\pm x, \pm y$). Relative addressing adds scalability in many aspects, but mainly because nodes do not need to be uniquely addressed in a distinguished setup phase, and it allows having a network greater than the actual address space. The address space only limits nodes to a confined "horizon of events", which is how far each node can communicate to.

Network Devices. Networking devices (ND) are full-duplex, with queue (Q), traffic shaper (SH) and serializing unit (PS) at the output port, and at the input port, a queue and a deserializing unit (SP). The internal representation of ND is shown in Figure 3c. Incoming packets are deserialized and queue, then dequeued and served by the router according, to configured arbitration policy. Outgoing packets are first queued, then dequeued by the traffic shaper, serialized and transmitted. In case *FIFO* arbitration is used, packets only get queued at the output port, and incoming packets are immediately served. In case of *RR*, packets are queued at the input port, and dequeued as their targeted output port becomes available. In this case packets do not get queued at the output port.

The purpose of the traffic shaper is to make packet transmissions periodic, such that we can provide determinism to the outgoing traffic, and consequently make it amenable to real-time analysis and real-time applications. It is configured in terms of an initial offset and desired transfer rate, therefore this is an optional peripheral, disabled by default. The serializer and deserializer are used on XDense, once we have established communication between nodes to use serial links. However, it is an optional configuration, and can be disabled for NoC architectures with parallel links.

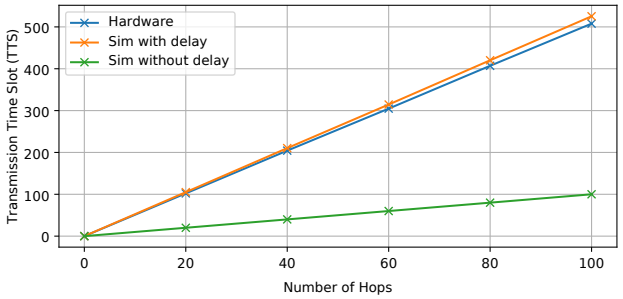


Figure 7: Average trip delay in a multi-hop scenario for varying trip distances.

Packet. XDense packet structure was designed for low resource utilization and for ease of routing and processing by simple hardware. It uses 16 bytes size, which is a common UART buffer size on microcontrollers (μC), allowing easy handling of packets for increased reliability and decreased delays. Its current structure is defined in Table 1.

The first byte is allocated for the communication protocol (P) and defines the routing protocol used, and hence determines how to interpret and use the remaining content of packet's header. Two coordinate pair are used to specify origin (x_o, y_o) and destination (x_d, y_d) of the packet. If relative addressing scheme is used, origin and destination coordinates are updated by router at each hop, to maintain its relativity to the origin, otherwise kept fixed. Payload (PL) is used by the application layer to transmit sensed data, and any other application protocol required. Checksum (CS) is utilized for error checking.

Link. Links are meant to simulate a full-duplex serial (UART) port, commonly found on COTS μCs. It adds two extra bits for each byte of the packet as start and stop bits. We chose UART to interconnect nodes because of its low complexity, low cost and availability. This allows XDense nodes to be implemented on mid to low-range μCs, having the router and NDs are implemented in software. However, like mentioned earlier, links are configurable, and can be easily setup to simulate parallel buses found in NoC architectures.

3.2 Network Communication Model

Having specified our design, the next stage consists of defining the temporal aspects of our network model. For that we need to chose a hardware platform that suites XDense, keeping in mind the trade-off between performance and cost. For this, we survey COTS μCs that meet the minimum requirements of XDense. These are, with at least four UART ports, with minimum 16 bytes buffer per port, preferably with dynamic memory access (DMA) between each UART port and the CPU for acceptable performance.

In [12] we presented XDense node based on the Atmel ATSAM4N8A chip, with a 32-bit ARM Cortex-M4 processor. This is a mid-range general purpose μC that run at up-to 100 MHz, with 5 UART ports, each one with an individual DMA channel. The UART ports communicate 16 byte packets at 3Mbps, leading to 53.3μs packet duration (further details can be found in the referenced paper). With this information in hand, we are able to add communication delays to our network model.

3.3 Software Components Specification and Modeling

Chosen the target hardware, at this stage, we want to measure how much internal delays that may impact on the network performance. This is a challenging task to accomplish, since there are numerous sources of internal delays, either imposed by the hardware components of the μC, or due to processing at the software layers, and determining the exact sources is not trivial.

In order to simplify this task, we opt to perform measurement base modeling, which means that we did a statistical survey of the delay imposed by each node on forwarding a packet. As said, this is a simplified internal delay model, which accounts statistically only for the most significant source of delays overall.

For this purpose, we use a single hardware node (running dedicated firmware), and perform automated measurements on the time it takes to forward a single packet (with the aid of an oscilloscope). We perform ten thousand measurements, in order to converge to a consistent statistical distribution. The result is shown in Figure 6. It is a random distribution, whose form is strictly related to the μC firmware implementation. With this data in hand, we can move to the next stage, which is to take internal delays into account along with our network communication model.

3.4 High Level System Model

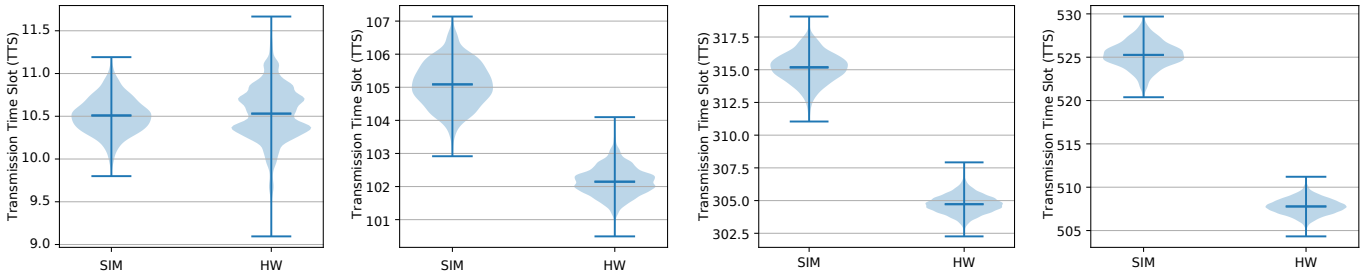
Having in hand information on communication and internal delays, we are able to consolidate the High Level System Model. For that, we feed the router model with the list of delay measurements taken. The router randomly picks a value from this list before forwarding a packet, delaying its transmission by that value.

With this, we are able to simulate communication and internal delays approximately. This lowers computational cost, but at the same time bringing our model much closer to reproduce the performance observed in real hardware. This is specially useful to study the effects of asynchrony between nodes due to non-deterministic delays. This also provides statistical bounds on communication delays.

4 MODEL STAGES VERIFICATION

To evaluate our model accuracy, we compare it against our hardware implementation. We want to identify how the high level system model perform compared to the hardware implementation, in terms of communication delay. We also measure the packet drop ratio observed in hardware to quantify its significance. We also want to identify the effect of composing the software delay model with the network delay model, by looking at the last one separately. For clarity, we show delay values in terms of transmission time slots (TTS), where 1 TTS is the time required to transmit a single packet (packet duration, equal to 53.333μs).

We start by measuring both for simulation and hardware deployment, the time taken for a single packet to travel through many hops, from origin to destination (on the network without concurrent workload). We vary the trip distance from 2 to 100 hops and



(a) 2 hops (b) 20 hops (c) 60 hops (d) 100 hops

Figure 8: Comparison between simulation and hardware of the packet trip delay distribution, for different number of hops.

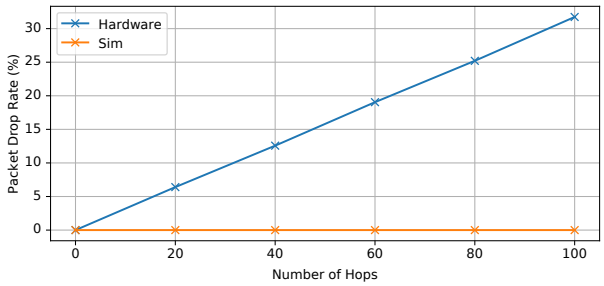


Figure 9: Packet drop ratio in a multi-hop scenario for varying trip distances.

measure the end-to-end delay at each scenario ten thousand times. Figure 7 shows the average end-to-end delay for each scenario. It shows the measurements for our hardware deployment, and simulation, with and without using the internal delay model. There is an expected linear growth on the trip delay as we increase the trip distance, for all three scenarios. Adding internal delays to the network model represents an approximate five-fold increase in the total delay. Furthermore, in this scenario, our high level system model approaches considerably the hardware performance, with linear increasing error.

Lets look closer and compare the measurements for the hardware and high level system model. In Figure 8, we show the statistical distribution of our measurements, for trip distances 2, 20, 60 and 100 hops. For short trip distances (2 hops), both scenarios show approximate same average value, whereas simulation present more contained normal distribution, compared to the random and more wide distribution observed in hardware. Therefore, that does not hold as we increase the trip distance, quite the contrary, the averages diverge, while the distribution for the hardware tends to get more concentrated around the average, more than compared to the simulation.

Finally, in Figure 9 we show the comparison between packet drop ratio for simulation and hardware. Obviously, since we did not implement error models to complement our high level system model, for the simulation we experience zero packet drops. While in hardware, we experienced linear growth on packet drops, reaching around 30% drops at 100 hops trip distance.

5 CONCLUSIONS AND FUTURE WORK

We provide a complete module implementation of our network model, that approaches satisfactorily the performance observed on real implementation platform. The implementation proved to be robust, with stable operation, providing access to a diversity of performance metrics from XDense.

This is still a simplified model. We must still account for transmission errors, and any other source of internal delays, which has yet to be investigated. In addition, efforts are still needed to make this module more suitable to specific NoC architectures. It is still a good starting point.

As future work, we intend to compare our model with the hardware in a bigger diverse set of scenarios and metrics. We also intend to bring concurrent workloads in many-to-one scenarios, taking into account queue sizes.

A POST-PROCESSING TOOLS

For the verification we developed post processing tools, to allow analyzing the results of our simulations. Figure 10 shows our visualization tool. It works based on the analysis of log files, that contains information on packets traffic (packet trace). It shows the activity by the application layer and NDs. Incoming and outgoing traffic is shown in red and green respectively, from and to each ND and application.

Our post processing tools also gives access to relevant information extracted from the simulation logs. For example per-node maximum queue size and delay (Figure 11a and 11b), as well as the arrival/departure curves at any node (Figure 12).

B HARDWARE PROTOTYPE

To validate our model we used our hardware prototype shown in Figure 13. We detail each of the main components seen in the board. It is based on the Atmel IC ATSAM4N8A, a 32-bit ARM Cortex-M4 RISC processor, which is a mid-range general purpose μC. It runs at up to 100 MHz and has a good balance between power consumption and processing power. It has a small 48-pin footprint, with five high speed serial ports (one USART and four UART ports), and 23 DMA channels that allows efficient communication and sensor reading.

We placed four different sensors on the top of the board for sensing, a 3-axis accelerometer (A), a pressure sensor (P), a temperature (T), and a visual-range light sensor (L). Along with this,

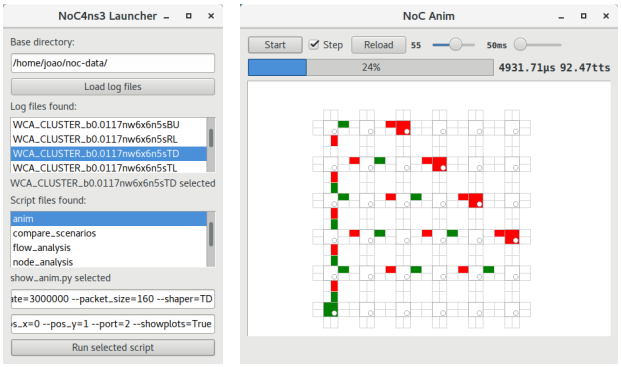


Figure 10: (a) Launcher for post processing tools. It shows the simulation scenarios found and the post-processing scripts available; (b) Visualization tool showing application layer and net-devices activity (green means incoming and red outgoing packets).

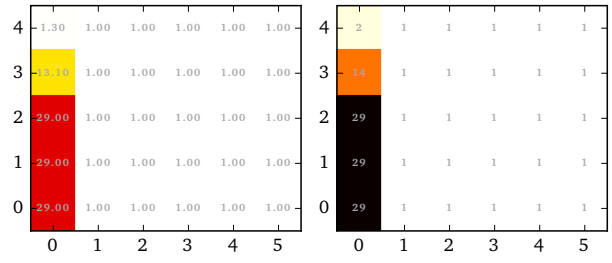


Figure 11: Example of (a) per-hop maximum delay and (b) queue size extracted by our post-processing tools.

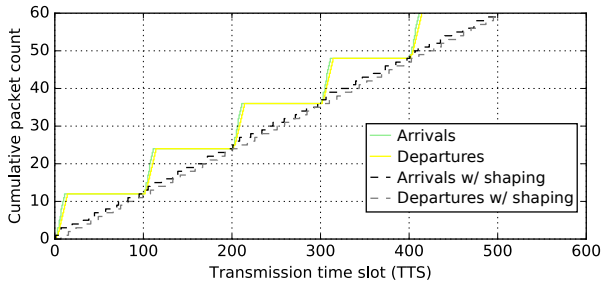


Figure 12: Cumulative arrival/departure curve at a single node, with and without using the traffic shaping unit.

for actuation, we have four RGB LEDs to transduce sensed values to colors, and represent any kind of distributed actuation for debugging proposes.

With a digital signal processing extensions (DSP) and floating point unit (FPU) co-processor, the IC chosen has enough computational power to allow us to demonstrate how XDense benefit of distributed processing. We should be able to identify patterns or

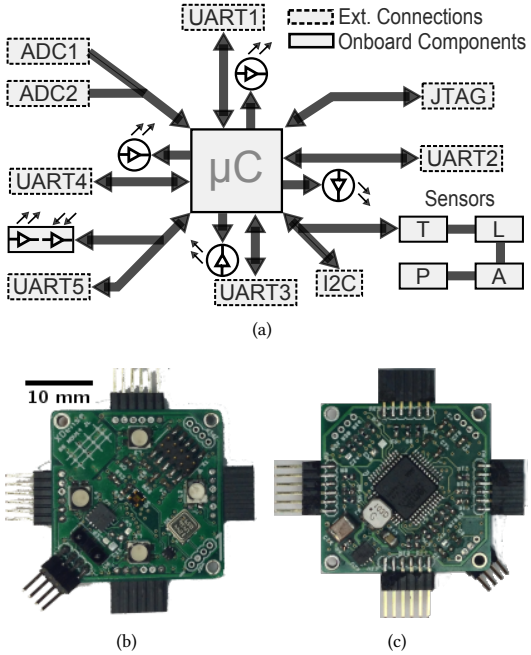


Figure 13: XDense sensor node schematic and prototype. Figure (a), the node's schematic shows each of major components of the system. Dotted lines are actually headers for external connections. Others are on-board components, like the LED's, the IrDA transceiver and the temperature (T), pressure (P), light (L) and acceleration (A) sensors. (b) and (c) shows the top and bottom side of the PCB respectively.

features on the input data, by distributively processing the sampled data, and in the same way being able to react to it.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0004/2013 - JU grant nr. 621353 (DEWI, www.dewi-project.eu).

REFERENCES

[1] A. Berns and E. Obermeier. 2009. AeroMEMS Sensor Arrays for Time Resolved Wall Pressure and Wall Shear Stress Measurements. In *Imaging Measurement Methods for Flow Analysis*. Springer, 227–236.

[2] U. Buder, R. Petz, M. Kittel, W. Nitsche, and E. Obermeier. 2008. AeroMEMS Polyimide Based Wall Double Hot-Wire Sensors for Flow Separation Detection. *Sensors and Actuators A: Physical* 142, 1 (2008), 130–137. DOI:http://dx.doi.org/10.1016/j.sna.2007.04.058

[3] L. Cai and D. Gajski. 2003. Transaction Level Modeling: An Overview. In *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '03)*. ACM, New York, NY, USA, 19–24. DOI:http://dx.doi.org/10.1145/944645.944651

[4] L. Cai, S. Verma, and D. Gajski. 2003. Comparison of Specfic and SystemC Languages for System Design. *CECS, University of California, Irvine, CA, USA, Tech. Rep* (2003).

[5] L. N Cattafesta III and M. Sheplak. 2011. Actuators for Active Flow Control. *Annual Review of Fluid Mechanics* 43 (2011), 247–272.

[6] D. Ganesan, A. Cerpa, W. Ye, Y. Yu, J. Zhao, and D. Estrin. 2004. Networking Issues in Wireless Sensor Networks. *J. Parallel and Distrib. Comput.* 64, 7 (2004), 799–814.

[7] N. Kasagi, Y. Suzuki, and K. Fukagata. 2009. Microelectromechanical Systems-Based Feedback Control of Turbulence for Skin Friction Reduction. *Annual review of fluid mechanics* 41 (2009), 231–251.

[8] N. K. Kavaldjiev and G. J. M. Smit. 2003. A Survey of Efficient On-Chip Communications for SoC. http://eprints.eemcs.utwente.nl/833/. In *4th PROGRESS Symposium on Embedded Systems, Nieuwegein, The Netherlands*. Technology Foundation STW, Utrecht, The Netherlands, 129–140.

[9] F. Kopsaftopoulos, R. Nardari, Y. Li, P. Wang, and F. Chang. 2015. State Sensing and Awareness for a Bio-inspired Intelligent Composite UAV Wing. (2015).

[10] S. Kumar, A. Jantsch, J-P Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. 2002. A Network-on-Chip Architecture and Design Methodology. In *2002 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2002)*. 105–112.

[11] J. Loureiro, V. Gupta, N. Pereira, E. Tovar, and R. Rangarajan. 2013. XDense: A Sensor Network for Extreme Dense Sensing. *Proceedings of the Work-In-Progress Session at the 2013 IEEE Real-Time Systems Symposium* (2013), 19–20.

[12] J. Loureiro, R. Rangarajan, and E. Tovar. 2015. Demo Abstract: Towards the Development of XDense, A Sensor Network for Dense Sensing. *12th European Conference on Wireless Sensor Networks (EWSN)* (2015), 23.

[13] J. Loureiro, R. Rangarajan, and E. Tovar. 2015. Distributed Sensing of Fluid Dynamic Phenomena with the XDense Sensor Grid Network. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA 2015)*. 54–59. DOI: http://dx.doi.org/10.1109/CPSNA.2015.19

[14] L. Maillet-Contoz and F. Ghenassia. 2005. Transaction Level Modeling. In *Transaction Level Modeling with SystemC*. Springer, 23–55. http://link.springer.com/chapter/10.1007/0-387-26233-4_2

[15] S. Medardoni. 2009. *Driving the Network-on-Chip Revolution to Remove the Interconnect Bottleneck in Nanoscale Multi-Processor Systems-on-Chip*. Ph.D. Dissertation. Università degli studi di Ferrara.

[16] A. Olofsson, T. Nordstrm, and Z. Ul-Abdin. 2014. Kickstarting High-Performance Energy-Efficient Manycore Architectures With Epiphany. In *2014 48th Asilomar Conference on Signals, Systems and Computers*. 1719–1726. DOI: http://dx.doi.org/10.1109/ACSSC.2014.7094761

[17] F. Palacios, J. Alonso, K. Duraisamy, M. Colonno, J. Hicken, A. Aranake, A. Campos, S. Copeland, T. Economon, A. Lonkar, and others. 2013. Stanford University Unstructured (SU 2): An Open-Source Integrated Computational Environment for Multi-Physics Simulation and Design. In *51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. 287.