Extensive Analysis of a Real-Time Dense Wired Sensor Network Based on Traffic Shaping

JOÃO LOUREIRO, CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
RAGHURAMAN RANGARAJAN, CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
BORISLAV NIKOLIC, CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal; Institute of Computer and Network Engineering, Technische Universität Braunschweig, Germany
LEANDRO SOARES INDRUSIAK, University of York, United Kingdom
EDUARDO TOVAR, CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal

XDense is a novel wired 2D-mesh grid sensor network system for application scenarios that benefit from densely deployed sensing (e.g. thousands of sensors per square meter). It was conceived for cyber-physical systems (CPS) that require real-time sensing and actuation, like active flow control (AFC) on aircraft wing surfaces. XDense communication and distributed processing capabilities are designed to enable complex feature extraction within bounded time and in a responsive manner. In this paper we tackle the issue of deterministic behavior of XDense. We present a methodology that uses traffic shaping heuristics to guarantee bounded communication delays and the fulfillment of memory requirements. We evaluate the model for varied network configurations and workload, and present a comparative performance analysis in terms of on link utilization, queue size and execution time. With the proposed traffic shaping heuristics, we endow XDense with the capabilities required for real-time applications.

CCS Concepts: • **Networks** → **Network performance modeling**; **Network performance analysis**; *Network performance evaluation*; *Network simulations*;

Additional Key Words and Phrases: Dense sensor networks, real-time communication, traffic shaping

ACM Reference Format:
João Loureiro, Raghuraman Rangarajan, Borislav Nikolic, Leandro Soares Indrusiak, and Eduardo Tovar. 2018. Extensive Analysis of a Real-Time Dense Wired Sensor Network Based on Traffic Shaping. *ACM Transactions on Cyber-Physical Systems* 1, 1, Article 1 (January 2018), 28 pages.
https://doi.org/10.1145/3230872

1 INTRODUCTION

As Moore's law remains valid, single embedded computers equipped with sensing, processing and communication capabilities are tending to be minimally priced. This makes it economically feasible to densely deploy sensor networks with very large quantities of computing nodes. Accordingly, it is possible to take very large number of sensor readings from the physical world, perform computation on sensed quantities and make decisions from the results. Very dense networks offer information about the physical world with greater resolution and therefore offer better opportunities in detecting the occurrence of an event; this is of paramount importance for a number of applications with high-spatial sensing (and actuation) resolution requirements.

XXXX-XXXX/2018/1-ART1 $15.00
https://doi.org/10.1145/3230872

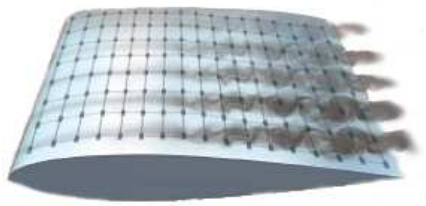


Fig. 1. Conceptual deployment of XDense for active flow control.

Such densely instrumented systems pose however huge challenges in terms of interconnectivity and timely data processing. It is important to note that the need for high spatial and temporal resolutions are often contradictory requirements, which are often not easily simultaneously fulfilled.

To further motivate our approach, let us consider an aerospace application scenario that may benefit from such dense CPS. The drastic increase in demand for air transportation, naturally motivates measures to reduce its environmental impact. The reduction of fuel consumption is important regarding both environmental effects and cost efficiency. It is known from the Breguet range equation [2] that improvements in aerodynamics, engines, and structure have major importance, and efforts in that direction aim at reducing aircraft drag and weight of the aircraft. In fact aerodynamic drag due to skin friction is known to be one of the relevant factors contributing to increased aircraft fuel consumption, that constitutes approximately one half of the total drag for a typical long range aircraft at cruise conditions [38].

A significant part of this skin friction is due to turbulent[1] airflow over the wing [28]. Turbulence can be highly undesirable, as it increases drag and noise. Additionally, it causes loss of energy [4], and an important goal is to minimize this loss. Figure 1 exhibits an example in which homogeneous laminar airflow transits to turbulent along the wing.

Several solutions have been proposed already to reduce turbulence. Cattafesta et al. [5] have surveyed the state-of-the-art actuation mechanisms used to reduce turbulent skin friction. A promising approach is based on a concept known as *Synthetic Jet Actuators* (SJAs). SJAs are actuators that run at key positions on the wing and continuously energize the airflow to avoid the formation of turbulence [39]. The recent advances in miniaturization and materials technology enable the development of a (thin) smart skin feasible, and hence SJAs are becoming an achievable technology [33]. The weakness of SJAs is to *not* use sensors to detect and trace the turbulent flows and hence offer only open loop actuation. This compromises the efficiency of active flow control (AFC), leading to waste of energy resources when there is no turbulent flow or when the turbulence lies outside the actuators' control field.

Therefore, implementing closed-loop AFC implies that physical quantities are tracked through sensors (for example, pressure, temperature and vibration sensors), which are deployed with some high density (eventually a few centimeters apart). Figure 1 shows an envisioned deployment of such sensing/detecting infrastructure on a wing surface to detect the occurrence of turbulent airflows.

XDense was developed to deal with the key challenges related to eXtremely Dense deployments of sensors [20]. XDense has a network architecture composed of regular structures (nodes) interconnected in a 2D-mesh network (see Figure 2(a)). This resembles common Network-on-Chip

[1] Turbulent airflow is composed of coherent structures of chaotic temporal evolution, such as vortices. Turbulent airflow causes an increase in interaction between the air and the wing (and the fuselage, in general), and consequently an increase in the total skin friction [29].

(NoC) architectures [14] and there are also similarities in routing schemes and distributed computing capabilities [16]. XDense exploits low-cost local communication and distributed processing strategies to enable distributed feature detection/extraction.

Even though we focus on AFC as the application scenario in this work, we believe XDense can be useful for other applications that require fast response times and dense networks of sensors. To mention a few: (i) aerodynamic tests [26]; (ii) structural monitoring [30]; (iii) biomedical devices for electroencephalogram [37]; (iv) robotic e-skins [31]; (v) health monitoring wearable sensor networks [25].

To investigate benefits of performing distributed processing, we take into account the nature of the input data. This because the efficacy of data processing algorithm are tightly related to the nature of the input data, in the sense that data clustering strategies and feature detection algorithms must be developed to fit specific scenarios. As well, the spatial and temporal granularity of the input data provides the necessary information to take decisions on the density of the deployment and minimum sampling rate requirements and consequently network load. For example, in [20], targeting AFC, we proposed algorithms to enable distributed turbulent air flow detection using computational fluid dynamics (CFD) data as our input.

However, the practicality of XDense for efficient feature detection and extraction is a necessary but not sufficient condition. We also need to provide execution time guarantees and bounds on the resource utilization for real-time applications like AFC, for which timeliness guarantees are essential to achieve closed loop actuation. Providing bounds on resource utilization is also crucial to correctly dimension the nodes, to avoid overload and consequent data loss. These are factors that have great influence on hardware requirements, cost and consequently on the applicability XDense. These two properties are therefore the focus of this paper.

Contribution

In this work, we extend XDense with real-time capabilities, by implementing traffic shapers in every node such that the network traffic is predictable and analyzable. Further, we propose an analysis framework to accurately model the network in terms of communication delay characteristics and memory requirements. Specifically, the paper makes the following two contributions: (i) propose three heuristics to shape the traffic in the network; (ii) develop a mathematical framework to model and analyze the application and network and provide upper-bounds on communication delays, application execution time, and maximum buffer requirements.

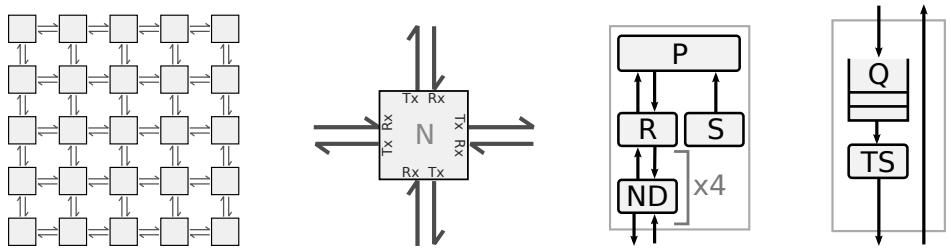
In extension to the work presented in [18], in this work we evaluate the heuristics proposed using homogeneous and heterogeneous traffic sources and compare it with the best effort approach.

The remainder of this paper is organized as follows. Section 2 introduces the basics of the XDense architecture; Section 3 formalizes our XDense model for real-time applications; Section 4 evaluates the model; Section 5 discusses the related works and Section 6 concludes the paper along with comments on potential future research directions.

2 XDENSE ARCHITECTURE AND PRINCIPLES OF OPERATION

2.1 Architecture and topology of XDense

XDense is a 2-D mesh network whose topology and node architecture are inspired from traditional Network-on-Chip (NoC) designs. Despite its similarities with NoCs, XDense differs in its size and node count. XDense is meant to be deployed on large surfaces (like aircraft wings) and deliver high-precision and high granularity measurements, thereby requiring a high number of sensors/nodes (in contrast to the few tens of interconnected nodes in modern NoCs).



(a) Network (b) Node (c) Node (d) ND

Fig. 2. (a) The XDense 2-D mesh network; (b) nodes use four bidirectional links to connect with neighbors located in the four cardinal directions (North, South, East, West); (c) node internals: processor (P), router (R), net-device (ND) and sensor (S); (d) net-device architecture. Output port includes a queue (Q) and a traffic shaper (TS).

Figure 2 illustrates the components of an XDense network at different levels of abstraction. Each node is composed of a sensor (S), a processor (P) and a router (R) and is connected to its neighboring nodes located in the four cardinal directions using bidirectional communication ports; termed networking devices (ND). Because they are bidirectional ports, we refer to their input and output independently as the input ports and the output ports.

The sensor is specified according to the nature of the phenomena to be monitored. For example, to enable high-precision AFC, pressure and temperature sensors can jointly provide better sensing of the airflow [13].

The processor runs the application layer. It interfaces with the sensor and implements high level application-specific protocols for data sharing and processing. The router arbitrates the exchange of data. It can receive and transmit packets in parallel, from/to the processor and networking devices. Networking devices are full-duplex serial communication ports. We use serial links as they are widely available in COTS micro-controllers and provide low complexity and low footprint at low cost (compared to parallel links found in NoC[14]). For example, in [8] the authors show that the utilization of parallel links beyond 2 millimeters represent up to 150% larger on-chip area utilization and up to 30% increase in power consumption when compared to high performance serial links. Their results clearly show that serial links present overall better performance compared to parallel links larger than a few milliliters. Thus, we believe that serial links are more appropriate to the deployment scales we envision.

Each one has a queue (Q) and a traffic shaper (TS) (see Figure 2(d)). Input packets are directly delivered to the router whereas output packets are first queued (in FIFO order) at the target output port before they are dequeued by the traffic shaper to be then transmitted serially over the network. All network transfers are non-preemptive and packet-switched, and all packets have a fixed and equal size.

The purpose of the traffic shaper is to provide determinism to the output traffic, and consequently make it amenable to real-time analysis. Its function is two-fold: it implements a release offset to the output packets and makes the transmission periodic. Shaping the traffic enables us to formulate the output traffic as a linear cumulative function of the input traffic. We will discuss our traffic shaping techniques in detail in Section 3.

It is important to remark that for this work, we ignore the internal delays of the nodes and focus exclusively on the communication delays. Also, all temporal units are normalized and quantified in terms of Transmission Times Slots (TTS), which is the time required to transmit a single packet.

2.2 Hardware implementation

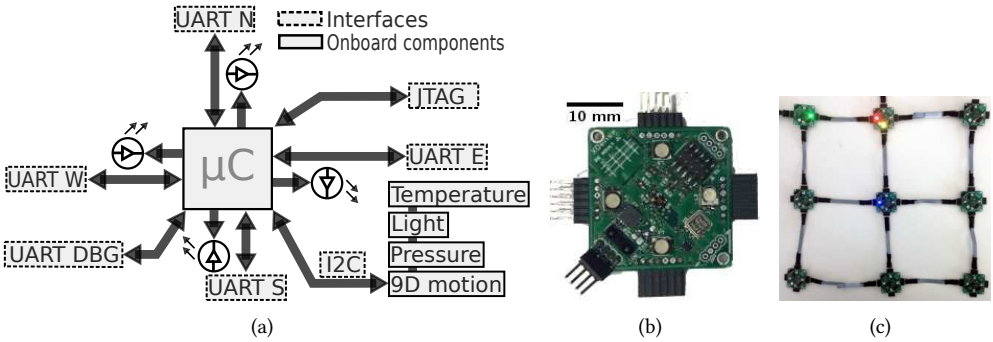


Fig. 3. (a) node's schematic showing each major components of the system. (b) Node prototype. (c) 3×3 network.

For realizing the above design of XDense, a custom design integrated circuit (IC) provides the best-fit solution; But this reduces design flexibility and might become a single application solution. For this reason, we use a microcontroller (μC) and other COTS to prototype the XDense node and network. The rest of this section provides context to this discussion with a short overview of the prototype that we have developed, shown in Figure 3.

To implement XDense, we chose the Atmel ATSAM4N8A μC. It is based on the 32-bit ARM Cortex-M4 RISC processor, which is a mid-range general purpose μC, that runs at up to 100 MHz and provides a good balance between power consumption and processing power. It has a small 48 pin footprint, with five high speed UART ports, each with dedicated DMA channels that allows efficient communication. We use the FreeRTOS[3] real-time operational system (RTOS) in our nodes. It provides device drivers and additional high level abstractions for context switching and multi tasking.

The schematics, prototype node and a 3x3 network deployment is shown in Figure 3. We placed four sensors on the top of the board for: motion sensing with 9 degrees-of-freedom; pressure; temperature, and visual-range light sensing. We have presented more details on the hardware prototype using COTS in [19].

2.3 AFC application scenario

We now revisit the AFC application discussed in the introduction. While not used as input in the analysis ahead, the focus being to prove the realtime nature of XDense, it is useful to understand how XDense fits into the AFC application workflow.

As shown in Figure 1, XDense is positioned on the wing of the aircraft to collect the environment data. Experimentally, a wind tunnel would be desired to pose the aerodynamic conditions over a wing surface embedded with an XDense network. Computational Fluid Dynamics (CFD) simulation allows us to simulate airflow over a wing and collect this information by a virtual deployment of XDense. This is a well studied area in aerodynamic research and one important test case is for the ONERA M6 wing in viscous flow [32].[2]

[2]The ONERA M6 wing was designed for studying three-dimensional, high Reynolds number flows with complex flow phenomena (transonic shocks, shock-boundary layer interaction, separated flow). It has become a classic validation case for CFD codes due to its simple geometry, complicated flow physics, and availability of experimental data.

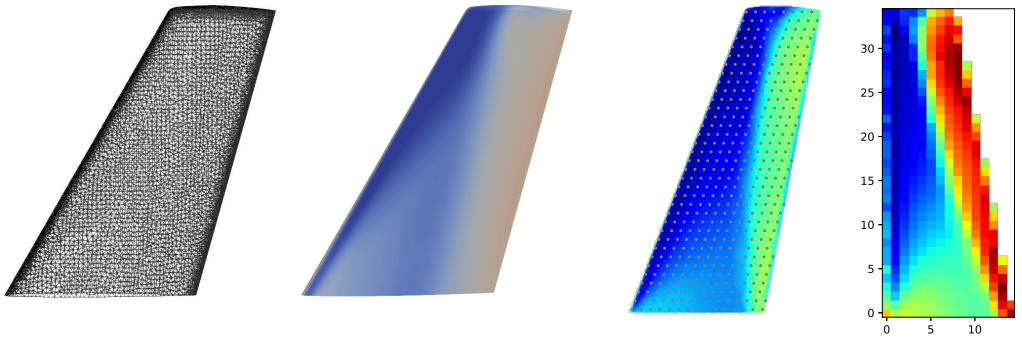


Fig. 4. (a) Pressure distribution over a wing's surface. (b) Data of a single time-frame, from Computational Fluid Dynamics (CFD) simulation, as input for XDense. (c) Sensors displacement; (d) Normalized data, as seen by each sensor.

We have integrated this CFD simulator into our workflow for simulation of XDense inputs. Now, the AFC input simulation workflow consists of the following steps:

(i) **Generate wing model:** A 3D mesh of a wing is generated and imported into the CFD simulator (Figure 4(a));
(ii) **Simulate wing performance:** The CFD simulator is run to simulate a pitching wing flowing through high speed air flow. Temporal pressure and temperature data of the surface of the wing are produced. (Figure 4(b));
(iii) **Extract sensor data:** These temporal pressure and temperature data are extracted, but only from the points in space that correspond to the XDense node deployment. Figures 4(c) and 4(d) illustrate sensor deployment and sensor data, respectively.

More specifically, the sensor data is generated and imported using the SU2 simulator for multi-physics [24]. SU2 is a reliable open-source CFD simulator which is widely used in aerodynamics research.

Even though we integrate our simulation model and the SU2 CFD simulator, note that the input data is indifferent to the network operation, as the analysis pertains to the communication aspects only. Therefore not influencing in any way the results presented in this paper.

2.4 Principles of XDense Operation

Consider the AFC use case depicted in Figure 1 as a working example. The objective is to collect information on the nature of the airflow and identify whether it is laminar or turbulent by quantifying its characteristics along the wingspan.

A naive solution to this problem is to request each node to continuously sense information about the airflow and send it back to a sink. The information collected from the sink can then be used to compute the airflow's properties. Clearly, this approach generates a tremendous load on the network, requires large buffers in each node, and leads to significant delays between the time at which the information is requested and the time at which it is eventually processed (sensed information may have a maximum lifetime).

Instead, we use XDense to efficiently build a global picture of the airflow by organizing the nodes in clusters and perform local data processing. In each cluster, one node serves as the cluster

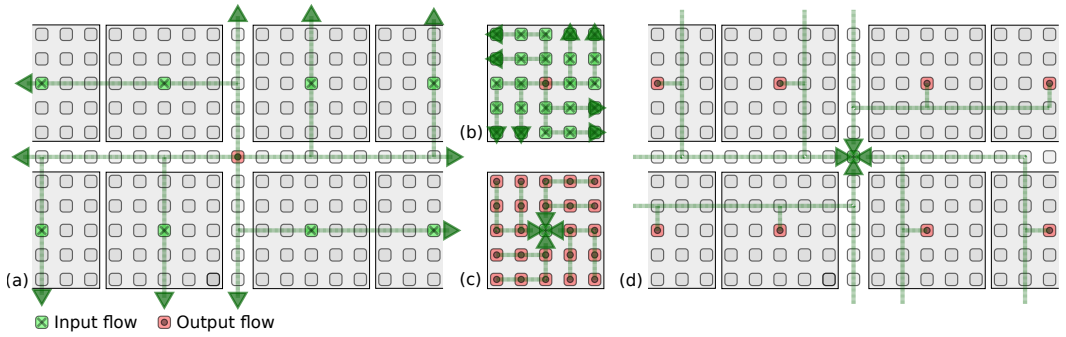


Fig. 5. Example 45×45 network, with a single central sink. In this case, with $n_{\text{radius}} = 2$. Application phases: (a) ϕ_1 – Sink requests data from cluster-heads; (b) ϕ_2 – Cluster heads in turn send a multicast request to nodes in their cluster; (c) ϕ_3 – Nodes send sensor data back to their respective cluster-head; (d) ϕ_4 – Cluster heads process received data and send result to sink.

head node. It performs data aggregation within its cluster and is responsible for processing (and/or compressing) the data locally to send only meaningful information to the sink. Another example of utilization is to program the cluster heads to inform the sink *only* upon the occurrence of meaningful events (e.g., airflow changes from laminar to turbulent and conversely).

The routing protocols elected should ideally exploit the network topology to avoid congestion. It is also required to define application protocols to allow coordination of clusters by the sink.

To tackle the challenge of analyzing and computing upper-bounds on the application execution time and the buffer requirements of the nodes through distributed processing, XDense uses three operative principles: (1) the nodes are clustered and one node in each cluster (cluster head) is in charge of aggregating and pre-processing the data; (2) the execution of the application is divided logically in subsequent phases; (3) the network implements routing schemes which guarantees spatial isolation between the clusters.

2.4.1 Clustering nodes. The reason for grouping the nodes into clusters is to reduce the load on the network by performing in-cluster data pre-processing at the selected cluster heads. Our tested solution implements non-overlapping "square" clusters – the network topology being a 2D grid of X times Y nodes, all clusters are non-overlapping and of size $n_{\text{size}} \times n_{\text{size}}$, with $n_{\text{size}} \leq X$ and $n_{\text{size}} \leq Y$. n_{size} must be a positive odd number and the cluster head is the node located at the "center" of the square. The cluster size n_{size} is defined through the system parameter n_{radius} that denotes the maximum distance from the cluster head to the farthest node in the cluster (considering rectilinear distance, a.k.a. Manhattan distance). Figure 5 shows a scenario with $n_{\text{radius}} = 2$. Thus, the resulting total number of nodes in each cluster is a function of the n_{radius} given by $(2 \times n_{\text{radius}} + 1)^2$.

Nodes arbitrate their role on the network at run time (to act either as cluster head or normal node). They do this on reception of a packet from the sink containing the packet origin and the n_{radius} parameter. Each node then calculates, based on its position in the network relative to the sink, if it is supposed to act as a cluster head or as a normal node.

As discussed earlier, the purpose of local in-cluster processing is to extract high level aerodynamic information of the airflow, which is transmitted in a smaller number of packets (when compared to the number of packets required to transmit the raw data). The pre-processing and compression algorithms to be used are application-specific and are not in the scope of this paper. We have though

discussed application specific data processing issues in previous works (see [20] for a discussion on this topic).

2.4.2 Executing application in phases. The execution of the application is logically divided into a set of four consecutive phases ϕ_1, ϕ_2, ϕ_3 and ϕ_4. The first phase is started by the sink, when it requests data from the cluster heads. Specifically, the four phases are:

- Phase ϕ_1. The sink requests the cluster heads of all clusters to send the processed data;
- Phase ϕ_2. On receiving the request from the sink, the cluster heads in turn request the nodes of their respective clusters to send their data;
- Phase ϕ_3. Every node of each cluster transmits its sensed data to its cluster head;
- Phase ϕ_4. The cluster heads process the received data and transmit the result back to the sink.

Note that the clusters may *not* always be in sync with respect to the phase of their execution. The second phase (ϕ_2) for instance, start in each cluster with a different time offset; This offset being proportional to the distance between the cluster head of each cluster and the sink. We assume cluster heads' clock are synchronized during ϕ_1, at the time they receive the request from the sink. The same applies to sensor node's clock, which are synchronized during ϕ_2, at the time they receive the request from their cluster head. That is, all nodes co-participating on a phase (in the same cluster) have a common time basis, which is an important assumption for the proposed heuristics to work. We believe this is a reasonable assumption, since nodes synchronization point happen just before synchronism is required, providing a momentary synchronism during a given phase, even when there is considerable clock skews between nodes.

Despite their special role, the sink and cluster heads sense as any other node. The sink is the only node to act as the gateway with the outside world and has a backhaul link (for example, a wireless link). Figures 5(a) to 5(d) show the four phases in a chronological order.

2.4.3 Spatial isolation through routing schemes. The four phases described above require spatial isolation so that packets do not compete with each other for network resources when traversing it. We use the well-known dimension-order routing algorithms known as X-Y and Y-X routing protocols [14]. In X-Y routing (resp, Y-X), packets are first routed along the X (resp, Y) dimension and then along the Y (resp, X) dimension. These protocols always find the shortest path between the source and destination nodes (again, in terms of the Manhattan distance) and are proven to be deadlock-free [11].

Phases ϕ_1 to ϕ_3 use one of the following two routing algorithms, sometimes called Counterclockwise Dimension Routing (see Figures 5(a)-(c)). The starting dimension (X or Y) depends on the quadrant in which the destination node is, relatively to the origin of the packet. For phase ϕ_4 we propose another routing protocol hereafter referred to as Shifted Clockwise Dimension Routing. This protocol adds an initial change in dimension on the first hop and then uses a regular clockwise routing (see Figure 5(d)).

The nodes aligned with the sink are not part of any cluster. They provide an exclusive route for packets of ϕ_4, sent by the cluster head to the sink. This routing scheme results in flows from phase ϕ_4 to travel orthogonal to the flows from phases ϕ_1, ϕ_2 and ϕ_3 and therefore they do not compete for the same output port at any node on the way. This enables spatial isolation between the flows from the different phases.

3 EXTENDING XDENSE WITH REAL-TIME APPLICATION CAPABILITIES

We endow XDense with real-time capabilities by *shaping the traffic at every output port of every node in the network*. In simple terms, by controlling how and when packets are sent by each node,

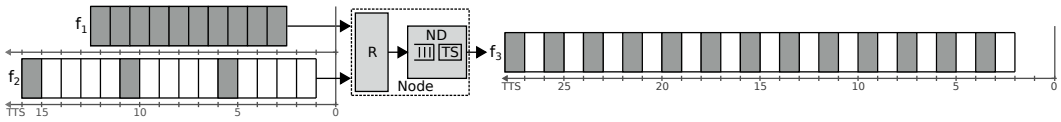


Fig. 6. Traffic shaper example scenario: two input flows shaped by an intermediate node as an output flow. Parameters for the input flows are $f_1 = \{O = 2.5, \beta = 1, \sigma = 10\}$ and $f_2 = \{O = 1, \beta = \frac{1}{5}, \sigma = 3\}$. The resulting flow is $f_3 = \{O = 2, \beta = \frac{1}{2}, \sigma = 13\}$.

we are able to compute the maximum buffer requirement and determine precise upper-bounds on the application execution time.

3.1 Networking model

The real-time application deployed on the network is characterized by a set $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ of n consecutive event-triggered phases (communication and processing primitives) that constitute the logical part of the application execution. In this work, we assume $n = 4$ (as explained in the previous section) but the approach can be extended to any arbitrary number n of phases. Every phase $\phi_i \in \Phi$, with $i \in [1, n]$, is characterized by a set $\mathcal{F}_i$ of $m_i \geq 1$ communication traffic flows exchanged between the nodes involved in phase ϕ_i. Each flow $f_{i,j} \in \mathcal{F}_i$, with $j \in [1, m_i]$, consisting of one or more packets, has an unique source node from which the communication is initiated, and may have multiple destination nodes. Formally, a flow $f_{i,j}$ is modeled as:

$$f_{i,j} = \{O_{i,j}\ ,\ \sigma_{i,j}\ ,\ \beta_{i,j}\} \tag{1}$$

The offset $O_{i,j}$ is a constant delay before the sending of the first packet of flow $f_{i,j}$. The message size $\sigma_{i,j}$ is the number of packets that are sent in each flow $f_{i,j}$ and the burstiness $\beta_{i,j} \in [0, 1]$ represents the rate at which those packets are released. A burstiness of 0 means that no packets are transmitted, and a burstiness of $x \in\ ]0, 1]$ means that a packet is transmitted every $\frac{1}{x}$ TTS. These three parameters together describe a finite constant-rate flow with an initial offset. The flow parameters σ and β were conceived to couple the application sampling requirements with the communication model, in the sense that they allow modeling application scenarios with different data sampling requirements. A few example flows are illustrated below.

Example 3.1 (9 Degrees of Freedom (DOF) motion sensor). Consider a 9 DOF motion sensor whose data has to be transmitted as nine separate packets in a single flow (one packet for each degree of freedom). In this case, we want the data to be transmitted together. Therefore, we set $\beta = 1$ with $\sigma = 9$ for that flow.

Example 3.2 (Pressure sampling). Consider a use-case in which ten samples of pressure data need to be transmitted, using one packet per sample. We are interested in having periodic sampling, equally distributed in time. By setting the burstiness to $\frac{1}{5}$ for instance, one packet will be sent every 5 TTS. Therefore, for that flow we set $\beta = \frac{1}{5}$ and $\sigma = 10$.

3.2 Shaping flows and traffic throughout the network

As discussed above, the sending of all packets by the source node of the corresponding flow f is done according to its parameters (O, σ, β); these three parameters allow for a precise timing and sending rate at the source node of f. Note that for simplicity, we shall use hereafter the symbol f to denote a flow. We will mention the indexes i and j that indicate the phase and flow indexes respectively only if necessary.

Although the flows are shaped at their source, when multiple flows (say, $f_1^{\text{in}}, f_2^{\text{in}}, \ldots, f_k^{\text{in}}$) traverse the network at the same time, pass through the same router, and compete for the same output port, the resulting output flow f^{out} at that port is a superposition of all these competing flows. As such, f^{out} may present an irregular packet transmission pattern and a rate that can no longer be modeled using the three parameters (O, σ, β).

For example, let us look at Figure 6, which illustrates two input flows $f_1^{\text{in}} and f_2^{\text{in}}$ competing for a same output port of a node. Each of these flows f_k^{in} starts at time O_k and has a duration defined as $\ell_k = \frac{\sigma_k}{\beta_k}$. That is, flow f_k sends all its packets after ℓ_k TTS, at $t = O_k + \ell_k$. In this example, thanks to the traffic shaper *TS*, the interference of these two input flows lead to an output flow f_3^{out} that is not a superposition of the two input flows, but rather it preset deterministic patterns that can be modeled using the three parameters (O, σ, β).

That is, to make the network amenable to timing analysis, we shape the traffic *at every output port* of every node and make it fit the linear model (O, σ, β). For that, we first identify the set of input flows f_k^{in} (with $k = 1, 2, \ldots$) at every output port of every node in the network, and based on the respective parameters (O_k, σ_k, β_k) of these flows, we compute the parameters $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ that are used to shape the resulting output flow at that output port.

In Figure 7, we present a more detailed example to illustrate how the traffic shaping is done. Figure 7(a) shows packet arrivals curve $S(t)$ due to four incoming flows $f_1^{\text{in}}, f_2^{\text{in}}, f_3^{\text{in}} and f_4^{\text{in}}$ (see Figure 7(b)). The arrival curve correspond to the incoming flows that define the number of packets to be sent over time from the output port, that depends on the starting time and duration of all the competing input flows. Three possible resulting flows f^{out} are computed and shown in Figure 7(c), each with its corresponding departure curve in Figure 7(a).

The computation of $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ is therefore performed at every output port of every node in the network interactively, starting at the source node of every flow and iterating, one port at the time, throughout the network until a shaper is defined for all the output ports. [3] We make two important assumptions regarding the flows and their routing.

Assumption 1. During phases ϕ_3 and ϕ_4, in every node, all the packets entering by a given input port are assumed to exit through a single output port.

Assumption 2. There are no circular dependency between the flows. For any output port, say p_1, the computation of the parameters of its traffic shaper requires each of its competing input flows to be modeled already by the three parameters (O, σ, β). If any of these input flows, say f_k^{in}, comes from the output port (say p_2) of an upstream router, it is required that the parameters $(O_k^{\text{in}}, \sigma_k^{\text{in}}, \beta_k^{\text{in}})$ of the shaper of that upstream output port p_2 have been computed already. Similarly, this requirement must be satisfied for all the input flows competing for p_2, and interactively it must be satisfied as well for all the output ports of the upstream routers till the traffic shaper at the source nodes of all the interfering flows. Therefore, computing traffic shaping parameters is an iterative process that must be executed until $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ is calculated for all nodes. In simple terms, there cannot be a flow f_1 competing for an output port with a flow f_2 that competes for an output port with a flow f_3, and so on until reaching a flow f_k that competes for an output port with f_1.

Assuming no cyclic dependencies between the flows, the parameters $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ of every traffic shaper may be computed in many different ways for a same set of interfering input flows. In the next section, we propose three different methods of computation.

[3] Note that it has been proven in [35] that to calculate optimal shaping parameters in a multihop scenario can be computationally intractable, and thus finding optimal solution at runtime is not feasible.

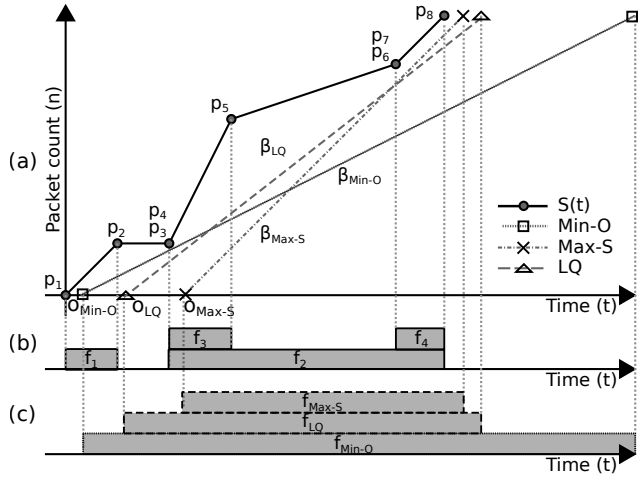


Fig. 7. Traffic shaping heuristics: (a) input, and output flows using the proposed heuristics; time-line showing offset and duration of (b) arriving flows and (c) departure flows.

3.3 Shaping output traffic at a single output port

We propose three heuristics to compute the parameters $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ of the shaper used at a given output port. Let F^{in} denote the set of input flows that compete for the output port under analysis. Every $f_k^{\text{in}} \in F^{\text{in}}$ is characterized by the three parameters $(O_k^{\text{in}}, \sigma_k^{\text{in}}, \beta_k^{\text{in}})$. For each $f_k^{\text{in}} \in F^{\text{in}}$, we define the function $S_k^{\text{in}}(t)$ as

$$S_k^{\text{in}}(t) = \begin{cases} 0 & t \leq O_k^{\text{in}} \\ \beta_k^{\text{in}} \times (t - O_k^{\text{in}}) & O_k^{\text{in}} < t < O_k^{\text{in}} + \ell_k \\ \sigma_k^{\text{in}} & t \geq O_k^{\text{in}} + \ell_k \end{cases} \tag{2}$$

Broadly speaking, every function $S_k^{\text{in}}(t)$ represents the number of packets sent by the flow f_k^{in} at a given time t (TTS). When t is earlier than the starting instant O_k^{in} of the flow, the function returns 0 since the flow has not sent a packet yet; For t larger than the finishing time of the flow $(O_k^{\text{in}} + \ell_k)$, the function returns the total number σ_k^{in} of packets sent by f_k^{in}, with ℓ_k being the duration of the flow; Between the two bounds O_k^{in} and $O_k^{\text{in}} + \ell_k$, the function increases steadily from 0 to σ_k^{in} with a constant slope of β_k^{in}.

Let $S(t) = \sum_{f_k^{\text{in}} \in F^{\text{in}}} S_k^{\text{in}}(t)$ be the sum of the functions $S_k^{\text{in}}(t)$ of all the input flows f_k^{in}. This function $S(t)$ is depicted in Figure 7(a). Informally, $S(t)$ gives the number of packets that arrive at the considered input port in a time window of length t (TTS). We further denote by $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ the finite set of time-instants (sorted in chronological order) corresponding to the discontinuity points of the function $S(t)$. These discontinuity points are denoted as $p_1, p_2, \ldots, p_m$ in Figure 7. With these new notations, we can introduce our three heuristics for the computation of the parameters $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ of the shaper used at the analyzed output port.

For a given shaper $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ represented by a straight line L^{out} of slope β^{out} and passing through the point $(O^{\text{out}}, 0)$, the *vertical* distance dv_j^{out} between a point $(t_j, S(t_j)) \in S(t), \forall t_j \in \mathcal{T}$, and the line L^{out} represents the number of packets being buffered at time t at that output port. The *horizontal* distance dh_j^{out} between a point $(t_j, S(t_j)) \in S(t), \forall t_j \in \mathcal{T}$ and L^{out} represents the delay

(induced by the shaper) that all the packets that have arrived at that output port at time t_j will incur because of the shaper.

We start by computing the output flow size σ^{out} that is the same for all the heuristics proposed. Since the shaper is not allowed to drop any packet, it is naturally the sum of the size of all the input flows f_k^{in}, i.e.

$$\sigma^{\text{out}} = \sum_{f_k^{\text{in}} \in F^{\text{in}}} \sigma_k^{\text{in}}$$

In the remainder of this section we discuss the intuition behind each heuristic and explain how they derive the two other flow parameters, O^{out} and β^{out}.

*3.3.1 Minimum offset (*Min-O*).* This first heuristic aims at avoiding bursty traffic while coping as much as possible with the bandwidth demand of the input flows. This traffic shaper forwards the first packet as soon as it can, i.e. one TTS after the packet has arrived, at time $O^{\text{out}} = t_1 + 1$ TTS, and forwards all the subsequent packets at the highest admissible rate; that is, with the highest burstiness β^{out} such that the number of packets sent at any time $t \geq O^{\text{out}}$ never exceeds $S(t)$. This burstiness corresponds to the highest slope among the slopes of all the lines passing through the point $(t_1 + 1, 0)$ such that, for every $t_j \in \mathcal{T}$, the point of x-coordinate t_j in the line has an y-coordinate $\leq S(t)$ – In simple terms, the line is "below" the function $S(t)$, $\forall t \geq 0$. This slope is simply given by

$$\beta^{\text{out}} = \left[\min_{t_j \in \mathcal{T}} \left(\frac{S(t_j)}{t_j - (t_1 + 1)} \right) \right]_0^1$$

where $[x]_y^z = \max(\min(x, z), y)$. Note that by definition of t_1, we have $t_1 = \min_{f_k^{\text{in}} \in F^{\text{in}}}(O_k^{\text{in}})$. Figure 7(a) shows Min-O departure curve with β^{out} as $\beta_{\text{Min-O}}$.

*3.3.2 Maximum slope (*Max-S*).* The second heuristic aims at *not* consuming any bandwidth for as much time as possible and then send all the packets in a burst. Similarly to the Min-O heuristic, the Max-S approach selects one "anchor" point of $S(t)$ and computes the *maximum* slope β^{out} such that the line with slope β^{out} passing through the selected point is "below" the function $S(t)$. In Min-O, we selected the anchor point $(t_1 + 1, 0)$ whereas Max-S selects the point $(t_m, S(t_m))$. The maximum admissible slope such that the line remains below $S(t)$ is given by:

$$\beta^{\text{out}} = \max_{t_j \in \mathcal{T}} \left(\frac{S(t_m) - S(t_j)}{t_m - t_j} \right) \tag{3}$$

Figure 7(a) shows Max-S departure curve with β^{out} as $\beta_{\text{Max-S}}$. The offset O^{out} in Max-S is simply set to the X-intercept of the line of slope β^{out} and passing through the anchor point $(t_m, S(t_m))$ to which we add 1 TTS, to make sure that packets are not forwarded before the first packet arrives (like we did in Min-O), i.e.

$$O^{\text{out}} = t_m - \frac{S(t_m)}{\beta^{\text{out}}} + 1$$

After computing the offset O^{out}, it is now safe to readjust the slope as $\beta^{\text{out}} = [\beta^{\text{out}}]_0^1$ to model the fact that the shaper cannot forward a negative number of packets and neither it can forward more than one packet at a time. Note that this re-adjustment must be performed after computing O^{out} as doing it before would in some cases allow a packet to be forwarded before it even arrived, that is, the line would not be completely below the function $S(t)$.

Figure 7(a) shows the departure line of Max-S, initially calculated with a slope > 1 as a result of Equation 3. That slope is then adjusted to $\beta^{\text{out}} = 1$ as depicted on that Figure. As seen, after adjusting its slope, the line corresponding to the parameters of the Max-S traffic shaper does not intersect with the function $S(t)$ – It seems to be "too much shifted to the right". An easy patch to reduce this gap between $S(t)$ and the shaper is to set its offset to the *minimum* offset such that the line remains below all the points of $S(t)$. That is,

$$O^{\text{out}} = \min_{t \geq 0} \left\{ t \text{ such that } \beta^{\text{out}} \leq \min_{\substack{t_j \in \mathcal{T} \\ t_j > t}} \left(\frac{S(t) - S(t_j)}{t - t_j} \right) \right\} \tag{4}$$

Note that this value of O^{out} can be computed easily by positioning the line of slope β^{out} on every point $(t_j, S(t_j))$, $\forall t_j \in \mathcal{T}$, and retaining the maximum X-intercept of all these lines.

3.3.3 Least-square regression (LQ). The intuition behind this third heuristic is to minimize both the queue size and the delay by finding the line L^{out} that minimizes the distance between every point $(t_j, S(t_j)) \in S(t)$, $\forall t_j \in \mathcal{T}$ and L^{out}. This line is commonly known as the *regression line* of the points $(t_j, S(t_j)) \in S(t)$. Using the least-squares method, which is the most common method for fitting a regression line, the slope of that line is given by

$$\beta^{\text{out}} = r \times \frac{\sqrt{\frac{1}{m} \sum_{t_j \in \mathcal{T}} (S(t_j) - \bar{S})^2}}{\sqrt{\frac{1}{m} \sum_{t_j \in \mathcal{T}} (t_j - \bar{t})^2}} \tag{5}$$

where

$$\bar{t} = \frac{1}{m} \sum_{t_j \in \mathcal{T}} t_j$$

$$\bar{S} = \frac{1}{m} \sum_{t_j \in \mathcal{T}} S(t_j)$$

and r is the correlation coefficient computed as

$$r = \frac{\sum_{t_j \in \mathcal{T}} (t_j - \bar{t})(S(t_j) - \bar{S})}{\sqrt{\sum_{t_j \in \mathcal{T}} (t_j - \bar{t})^2 \sum_{t_j \in \mathcal{T}} (S(t_j) - \bar{S})^2}}$$

Once we have computed the slope, we choose the smallest offset O^{out} such that the line of slope β^{out} and passing through $(O^{\text{out}}, 0)$ is never above any point $(t, S(t))$, $\forall t \geq 0$. This is done using Equation 4.

3.4 Worst-case per-hop delays and maximum queue sizes

Having stated the heuristics, we can now apply them to all the phases of the application. We perform this in a hop-by-hop strategy, starting from the output ports of the nodes for which the parameters $(O_k^{\text{in}}, \sigma_k^{\text{in}}, \beta_k^{\text{in}})$ of all the interfering flows f_k^{in} are known. For each such output port, the resulting output flow f^{out} is shaped using the same model $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ that is then propagated as the input flow in the next hop. The process continues until the parameters of the shaper of

every output port of all the nodes of the network are defined (the output ports that no flows ever traverse and that are thus unused are naturally ignored). As mentioned earlier, we assume that there are no cyclic dependencies between the flows at any output port, which implies that the process eventually terminates.

After that step, we can now compute at each output port the maximum transmission delay caused by its traffic shaper $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$, as well as its maximum queue size. To ease the explanation, we shall use the same visual representation as that used in the previous section for the shaper and the function $S(t)$. The shaper is represented by a straight line of slope β^{out} that intersects with the x-axis at the point $(O^{\text{out}}, 0)$. We denote this line L^{out} and write its equation as

$$L^{\text{out}}(t) = \beta^{\text{out}} t - \beta^{\text{out}} O^{\text{out}} \tag{6}$$

We define $S(t)$ as in the previous section and keep the notations $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ to express the finite set of time-instants (sorted in chronological order) corresponding to the discontinuity points of the function $S(t)$.

As explained previously, the number of packets buffered at the output port at any time-instant t is given by the *vertical* distance dv_j^{out} between the point $(t, S(t))$ and the point $(t, L^{\text{out}}(t))$ on the line L^{out}. This vertical distance is simply equal to

$$\text{dv}_j^{\text{out}} = S(t) - L^{\text{out}}(t)$$

and thus the maximum number MaxQueue of packets buffered at that output port is given by

$$\text{MaxQueue} = \max_{t \geq 0} \left(S(t) - L^{\text{out}}(t) \right)$$

Since $S(t)$ is a continuous piecewise function for which every sub-function is linear, it can easily be showed that the maximum of the previous equation can be found by looking only at the time-instants $t_j \in \mathcal{T}$ rather than at all $t \geq 0$, i.e.,

$$\text{MaxQueue} = \max_{t_j \in \mathcal{T}} \left(S(t_j) - L^{\text{out}}(t_j) \right) \tag{7}$$

This holds true because every sub-function of $S(t)$ is a segment that is either:

- parallel to L^{out}. In this case, all the points on that segment are at the same distance from L^{out}, including its two extremities that are discontinuity points with an x-coordinate included in $\mathcal{T}$.
- converging towards L^{out}. In this case, the *leftmost* point on the segment (whose x-coordinate is an instant $t_j \in \mathcal{T}$) is the furthest to L^{out}.
- diverging from L^{out}. In this case, the *rightmost* point on the segment (whose x-coordinate is an instant $t_j \in \mathcal{T}$) is the furthest to L^{out}.

Similarly, the transmission delay at any time-instant t is given by the *horizontal* distance dh_j^{out} between the point $(t, S(t))$ and the point of y-coordinate $S(t)$ on the line L^{out}. According to Equation 6, that point of y-coordinate $S(t) \in L^{\text{out}}$ has an x-coordinate x such that $S(t) = \beta^{\text{out}} x - \beta^{\text{out}} O^{\text{out}}$ and thus $x = \frac{S(t)}{\beta^{\text{out}}} + O^{\text{out}}$. The horizontal distance is then simply given by:

$$\text{dh}_j^{\text{out}} = \frac{S(t)}{\beta^{\text{out}}} + O^{\text{out}} - t$$

and thus the maximum delay MaxDelay at that output port is:

$$\text{MaxDelay} = \max_{t \geq 0} \left(\frac{S(t)}{\beta^{\text{out}}} + O^{\text{out}} - t \right)$$

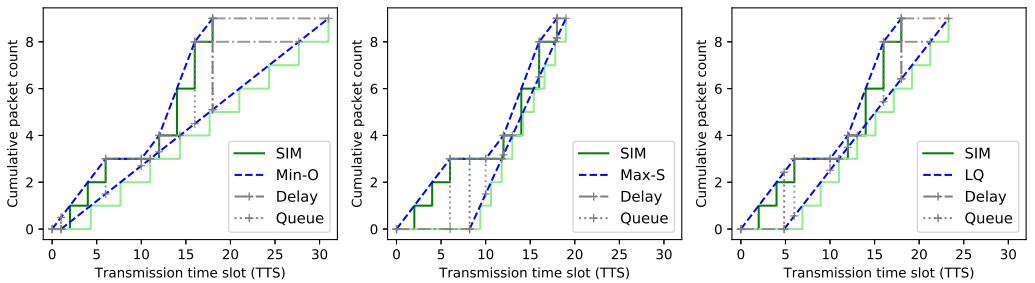


Fig. 8. Cumulative arrival/departure curves for a single node, using (a) Min-O, (b) Max-S and (c) LQ heuristics.

For the same reasons as those mentioned for MaxQueue, the maximum delay MaxDelay can be computed by looking only at the points $t_j \in \mathcal{T}$, i.e.,

$$\text{MaxDelay} = \max_{t_j \in \mathcal{T}} \left(\frac{S(t_j)}{\beta^{\text{out}}} + O^{\text{out}} - t_j \right) \tag{8}$$

Note that the transmission delay is an interesting parameter to analyze the end-to-end delay or per-hop delays of individual packets. However, in this paper we rather focus on estimating upper-bounds on the execution time of the phases and thus of the overall real-time application.

To compute the execution time of a given phase, we must know exactly when the phase start and when it ends. However, phases may overlap in time and happen simultaneously. For instance, for the application scenario considered in this paper, a cluster head located close to the sink may enter phase ϕ_2 long before a cluster head that is far from the sink (since it receives the request from phase ϕ_1 sooner). For simplicity, we assume in this work that a phase ends when a given node has received all the packets sent to it. For example, the time at which all the cluster heads have received their requested data marks the end of phase ϕ_3 and the time at which the sink has received all the processed data marks the end of phase ϕ_4. As such, we compute the execution time of a phase as the relative time-instant at which all the four input flows of that given node – a cluster head for phase ϕ_3 and the sink for phase ϕ_4 – terminate, i.e. the four flows coming from the north, south, east, and west input ports of that node. The execution time of a phase is thus given by

$$\text{ExecTime} = \max_{\text{card} \in [\uparrow, \downarrow, \rightarrow, \leftarrow]} \left(O^{\text{in}} + \frac{\sigma^{\text{in}}}{\beta^{\text{in}}} \right) \tag{9}$$

where for each cardinal direction $\uparrow$, $\downarrow$, $\rightarrow$, and $\leftarrow$ (north, south, east, and west), the flow f^{in} characterized by $(O^{\text{in}}, \sigma^{\text{in}}, \beta^{\text{in}})$ is the input flow coming from that cardinal direction.

3.5 Validation example

To validate our theoretical model we compare it with simulation. For that, we define the following scenario: a single node receives an arbitrary number of known input flows, which are shaped into an output flow (using any of the proposed heuristics) We compare the arrival/departure curves calculated using our model, against the curves obtained in simulation.

Figure 8(a-c) shows the cumulative arrival/departure curves due to three input flows, and the resulting output flow obtained using each of the three heuristics proposed. The input flows characteristics were chosen in order to emphasize the effect of each shaping heuristic on the output flow.

We use the same input flows in all three scenarios, which are $F_x = [f_1^{in}, f_2^{in}, f_3^{in}]$, where $f_1^{in} = \{O = 0, \sigma = 3, \beta = 0.5\}$, $f_2^{in} = \{O = 10, \sigma = 3, \beta = 0.5\}$ and $f_3^{in} = \{O = 12, \sigma = 3, \beta = 0.5\}$. The resulting output flows are different for each heuristic, which are $f_{Min-O}^{out} = \{O = 1, \sigma = 9, \beta = 0.3\}$, $f_{Max-S}^{out} = \{O = 8.2, \sigma = 9, \beta = 0.83\}$ and $f_{LQ}^{out} = \{O = 4.8, \sigma = 9, \beta = 0.49\}$.

The arrival curves obtained through simulation (common to the three cases) is a stair function that presents the superposition of all the arriving flows. Because the output flow is shaped, the corresponding departure curve is a homogeneous stair function. As expected, the arrival/departure curves calculated using our model precede the simulated ones in every point. It also shows the queue size and delay calculated at every point in which the arrival and/or departure curves start, finish or change its slope.

For the given F_x, from Figure 8(a), Min-O performs badly compared to the other heuristics, as it adds large delays between the arrivals and departures, which leads to equally large queues and long execution time. We notice from Figure 8(b) that the departure curve obtained with Max-S approaches maximally the arrival curve at its tip (1 TTS far), leading to the optimal execution time with the cost of larger queues from 0 to 10 TTS. On the other hand, LQ heuristic leads to smaller queues, with a slightly longer execution time.

Although these results provide an intuition on the trade-offs between the heuristics proposed, they do not depict the results of multi-hop communication, in which case the effects may differ. Therefore, a more complete evaluation is provided in the next section to understand how each heuristic performs through multiple hops.

4 EVALUATION OF TRAFFIC SHAPING HEURISTICS

Application use-case: To evaluate the proposed heuristics, we consider the application scenario introduced in Section 2. Remember that the execution of this application is divided logically in four consecutive phases ϕ_1, ϕ_2, ϕ_3 and ϕ_4. In the first phase ϕ_1, the unique sink node requests all the cluster heads to send their data; in phase ϕ_2, the cluster heads performs another request to all the nodes of their respective cluster; in phase ϕ_3, the nodes reply to the cluster heads by sending them the sensed data; and in phase ϕ_4, the cluster heads process the data received and transmit the result back to the sink. Since there is no network congestion in phases ϕ_1 and ϕ_2 – because all the packets sent from the sink to the cluster heads and then from the cluster heads to the sensing nodes have their own private route to their destination – these two phases are neither affected by a modification of the cluster size, nor by changing the number of clusters, nor by altering the burstiness of the flows generated during phases ϕ_3 and ϕ_4. We shall therefore focus *only* on phases ϕ_3 and ϕ_4 in which network congestion does occur and for which a modification of the aforementioned parameters has an impact on the performance.

Network setup: The network is organized as a square grid of $45 \times 45 = 2025$ nodes with an unique sink located at the center of the grid. Figure 5 depicts a closeup on the sink. In that figure we can also see the overall cluster organization, the routes taken by the flows in the different phases and the central row and central column of nodes in the middle that are dedicated only to the communication between the cluster heads and the sink. Based on an integer parameter n_{radius} that we vary in our experiments, we define every cluster as a square grid of $(2n_{\text{radius}} + 1)^2$ nodes with the cluster head at the center of the grid. As such, n_{radius} defines both the cluster size and the number of clusters (the smaller the clusters, the more clusters in the network, and reversely). Because of our routing algorithms and network symmetry assumptions (position of sink in the center of the network and cluster-head in the center of their cluster), the workload observed in each quadrant around the sink or cluster-heads will be identical. This makes it sufficient to analyze a single quadrant of the network or cluster.

Shaping heuristics: We evaluate the performance of the three proposed heuristics Min-O, Max-S, and LQ against the performance of a cycle-accurate network simulator that we call BE. The simulator does not implement any traffic shaper and thus it delivers the best effort (BE) performance overall.

Simulator: The simulator consists of a module for XDense on top of Network-Simulator-3 (NS-3). It is scalable to simulate very large network deployment scenarios with low computational cost. This is because we use packets, routing algorithms and addressing schemes with low overhead, tailored to this kind of network.

The general nature of the design and implementation of configurable links, packets, communication ports, router and applications, also makes our module suitable to other 2D mesh network architectures as well (NoCs for example). This is because the different abstraction levels can be implemented independently, as a set of intermediate models, each one with its specific objectives. For example, the traffic shapers theorized in this work were actually implemented in our simulator as an independent application layer, so we could study its practical viability, and for debugging purposes. [4]

Evaluation criteria and methodology: For each of the three heuristics Min-O, Max-S, LQ, we evaluate the maximum queue sizes and the end-to-end execution time of the phases ϕ_3 and ϕ_4. For BE, maximum queue sizes and phases execution time are measured in the simulator. We do so for different cluster sizes, flow burstiness and network load distribution. Because there is no source of nondeterminism in our simulation model, each runs gives the exact same results for the same input parameters. Thus, we are only required to run our experiments once for each scenario, for as long as all four phases last.

To understand the impact of varying the network load, we analyze both homogeneous and heterogeneous flow scenarios in phases ϕ_3 and ϕ_4 (that is, the phases when the actual data transmission happens). We define a homogeneous flow scenario as one in which all nodes generate flows with equal burstiness and message size. A scenario with random message sizes and burstiness is defined as a heterogeneous flow scenario.

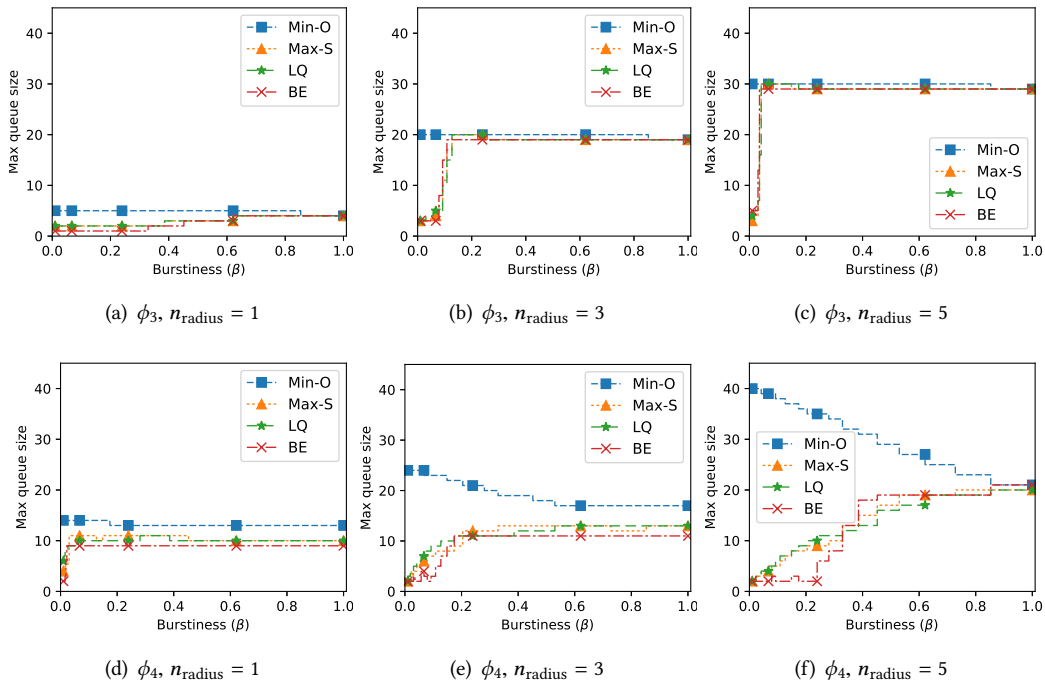
We analyze the homogeneous scenario by varying the burstiness β of flows from phases ϕ_3 and ϕ_4 from 0.02 to 1 by step of 0.02, for different cluster sizes.

The message size σ differs for each phase. For phases ϕ_1 and ϕ_2, a single packet is generated at the sink and cluster head ($\sigma = 1$). In phase ϕ_3, each node outputs a flow with message size $\sigma = 4$. At the end of ϕ_3, the cluster head receives in total four packets per each node on its cluster. Subsequently, each cluster head outputs a flow with message size σ, as the sum of all these packets plus 4 packets of its own sensed data, times $\lceil 1 - \mathrm{CR} \rceil$. The term CR aims at reproducing the effect of data compression by the cluster head. For this work we define this as a fixed value equal to CR = 80%, which was shown in previous work [20] to be a reasonable ratio in some air flow scenarios. The number of packets originated by each cluster vary with cluster size, whereas the number of clusters is inversely proportional to the cluster size. This trade off has a compensatory effect on the overall number of packets transmitted to the sink.

In the heterogeneous flow scenario, flows generated at phases ϕ_3 and ϕ_4 have random message sizes. We use a uniform distribution function with $\sigma = rand(0, 10)$ and burstiness $\beta = rand(0.02, 1)$. A message size of zero signifies that a node does not have an output flow.

In our results, we compare the performance of both heterogeneous and homogeneous scenarios. In order to do this fairly, we guarantee that for both homogeneous (HO) and heterogeneous (HE) network load distribution, the sum of burstiness of all flows, as well as the sum of all message sizes,

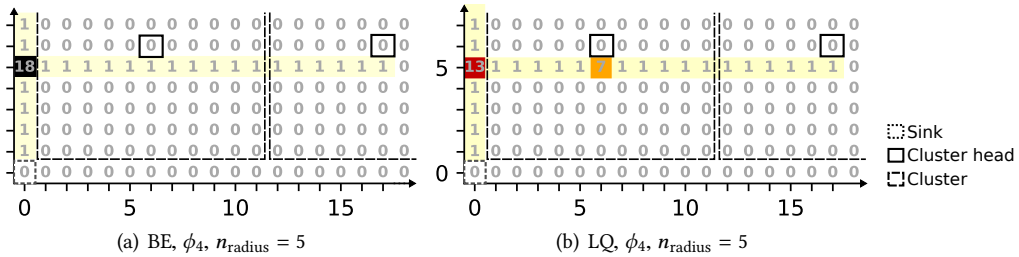
[4]The simulator source code used in the simulation this paper for the simulator, pre and post processing tools, example scenarios and implementation of the traffic shaping heuristics are available at: https://bitbucket.org/joaofl/noc



(a) ϕ_3, $n_{\text{radius}} = 1$ (b) ϕ_3, $n_{\text{radius}} = 3$ (c) ϕ_3, $n_{\text{radius}} = 5$

(d) ϕ_4, $n_{\text{radius}} = 1$ (e) ϕ_4, $n_{\text{radius}} = 3$ (f) ϕ_4, $n_{\text{radius}} = 5$

Fig. 9. Homogeneous flow scenario: Maximum queue size for traffic shaping heuristics against simulation. Results are for phases ϕ_3 and ϕ_4 and n_{radius} set to 1, 3 and 5.



(a) BE, ϕ_4, $n_{\text{radius}} = 5$ (b) LQ, ϕ_4, $n_{\text{radius}} = 5$

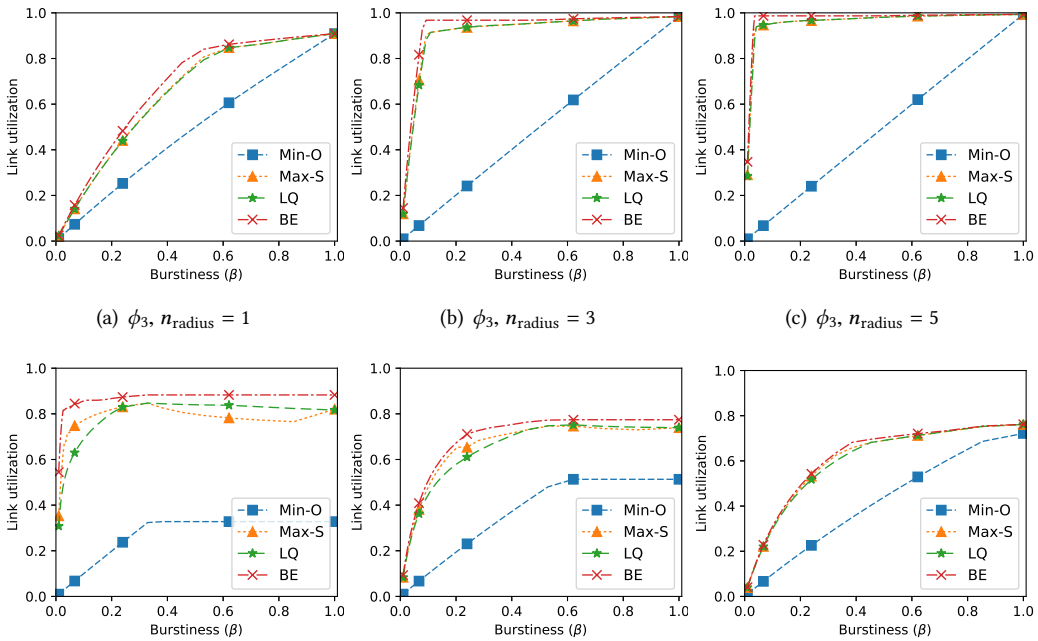
Fig. 10. Queue size density map of the top-right quadrant of the network (17×7 nodes), for heuristics (a) LQ and (b) BE. X and Y axis are nodes coordinates relative to the sink.

are equal. That is, $\sum \beta^{HO} = \sum \beta^{HE}$ and $\sum \sigma^{HO} = \sum \sigma^{HE}$. This guarantees that the total network load remains the same for both scenarios, even if individual load distribution varies.

For both scenarios, the offset remains the same; and equal to their distance from the sink/cluster head (since it is meant to model the minimum time required for a node to reply to a request).

4.1 Maximum queue size with homogeneous load distribution

For each of the three heuristics Min-O, Max-S and LQ, we first derive the parameters (O, σ, β) of all the traffic shapers in the network. Then we use Equation 7 on every shaper to compute the



(a) ϕ_3, $n_{\text{radius}} = 1$ (b) ϕ_3, $n_{\text{radius}} = 3$ (c) ϕ_3, $n_{\text{radius}} = 5$

(d) ϕ_4, $n_{\text{radius}} = 1$ (e) ϕ_4, $n_{\text{radius}} = 3$ (f) ϕ_4, $n_{\text{radius}} = 5$

Fig. 11. Homogeneous flow scenario: Link utilization for traffic shaping heuristics against simulation. Results are for phases ϕ_3 and ϕ_4 and n_{radius} set to 1, 3 and 5.

maximum queue size of the corresponding node and finally, we retain the maximum queue size of all the nodes in the network.

As we can see in Figures 9(a) and 9(b), in phase ϕ_3 the queues are smaller for smaller clusters (n_{radius}). This is expected since smaller clusters contain less nodes and therefore there are less packets exchanged within each cluster, and thus less congestion. The opposite scenario would be expected for phase ϕ_4 since using smaller clusters means more clusters in the network, and thus more cluster heads transmitting packets to the sink. Yet, this is not observed in Figure 9(d) and 9(e). That is, smaller clusters *do not imply longer queues*. The reason for this counter-intuitive result can be unveiled by looking at the *utilization* of the four input links of the sink.

We define the link utilization as the average utilization of a given link of a node during a given phase. It is calculated as the number of packets sent on that link in a given phase (here, phase ϕ_4) divided by the time (number of TTS) it takes for all those packets to traverse it. An utilization of 1 means that the link is never idle during the considered phase whereas an utilization of 0 means that the link is not used. As seen in Figure 11(d), smaller clusters yield a better utilization of the input links of the sink. This is because the sum of packets sent to the sink does not depend *only* on the cluster size. With more (and smaller) clusters, there will be more clusters and more cluster-heads transmitting to the sink from shorter distances. Thus, its input links will spent less time idle waiting for the packets to arrive from longer distances. In other words, with fewer (but bigger) clusters, cluster heads are farther from the sink and thus its input links spend more time idle waiting for the packets to traverse the intermediate hops. Greater utilization, for the same number of packets

received by the sink, shows that there is less congestion in the network and thus smaller queue at individual hops.

It is worth noticing that in some scenarios, the maximum queue size obtained when using traffic shaping is smaller than the maximum queue size without traffic shaping. This is experienced for example in ϕ_4 for $n_{radius} = 3$ and 5, shown in Figure 9(e) and 9(f), for $\beta \in [0.4, 0.6]$. In this window, the maximum queue size of Max-S and LQ are smaller than that of BE. This result is due to the offset O imposed by the traffic shapers in the initial hops. In these cases, the offsets act on distributing in time the load on the network, and thus decreasing the maximum congestion.

However, in cases with lower link utilization and burstiness, BE yields shorter queue sizes. In these cases the network is underutilized, such that BE still does not lead to excessive load on the network. On the other hand, performing traffic shaping imputes on unnecessary buffering by nodes, and consequently greater queues.

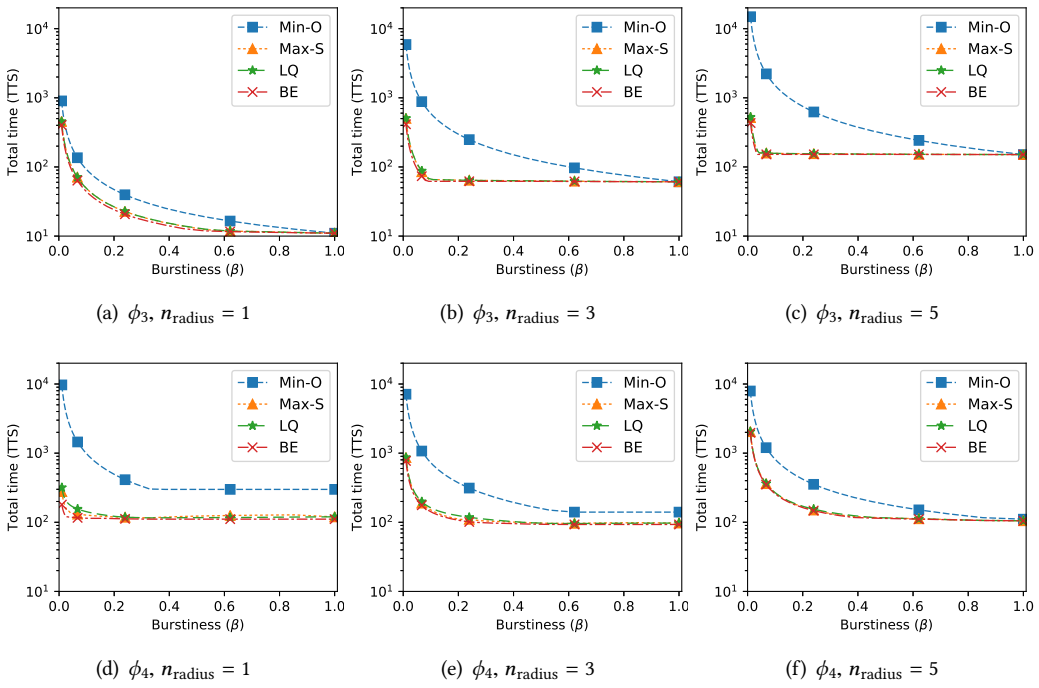
The aforementioned effect is shown in Figure 10. It shows the queue size density map of the top-right quadrant of the network, with $n_{radius} = 5$. The sink is located at the bottom-left corner. Flows are routed using shifted clockwise routing as previously shown in Figure 5; hence, right to left in this map. The left most nodes in the network are usually where the bottleneck happens. Using BE for example, in Figure 10(a), the node located at coordinates $(x, y) = (0, 5)$ gets up to 18 packets queued, since it is located at a conjunction of flows coming from cluster heads on its right and top. Because of the offset O calculated using LQ, we can see from Figure 10(b) that by shaping and queuing the flows originating at the right side of the network (by the node located at $(6, 5)$) has the effect of delaying the reception of those packets by the left most node located at $(0, 5)$; Thus, reducing the maximum queue size at the nodes aligned with the sink. Comparatively, from 18 packets using BE, to 13 using LQ.

Another interesting observation occurs during ϕ_4 for the method Min-O. The maximum queue size gets smaller with increased burstiness. This is very counter-intuitive since we would expect that by injecting more traffic in the network, the congestion would increase. However, this phenomena can be easily explained mathematically: it is due to the way the method Min-O is defined. Looking at Figure 7, the flows duration defined as $\frac{\sigma}{\beta}$ are longer for lower burstiness β and thus for low values of β, the first points $\in \mathcal{T}$ (depicted by p_1, p_2, etc.) are farther from the origin. Since Min-O selects a point close to the origin as "anchor" point, its slope must be small so that the line remains below the function $S(t)$. With a low slope, it is likely that the vertical distance between the function $S(t)$ and the line will be high (in particular if $S(t)$ increases quickly). These phenomena can be observed, to a limited extent, in Figure 7.

4.2 Phase execution time for homogeneous load distribution

We compare the execution time of the phases ϕ_3 and ϕ_4 in Figure 12, again for the cluster sizes defined by $n_{\text{radius}} = 1$, 3 and 5 and varying the burstiness of the initial flows from 0.02 to 1 by step of 0.02. The execution times are computed by using Equation 9. As seen in all graphics of Figure 12, increasing the burstiness considerably reduces the execution time of the phases (note that the plots are in logarithmic scale) which remains constant after a point. This point is reached only for high burstiness in Figure 12(a) whereas it is reached almost immediately in Figure 12(b). This threshold beyond which the execution time cannot be further reduced can be explained by looking at the utilization of input link of cluster heads (for phase ϕ_3) and the sink (for phase ϕ_4). Those thresholds correspond to specific values of the burstiness for which the links saturate and therefore, any further increase in burstiness only results in longer queues but not in reduced execution time.

From the above results, we observe that the LQ heuristic performs better overall. We vary n_{radius} from 1 to 5, with β varying as before, for both ϕ_3 and ϕ_4. The results are shown in Figure 13.



(a) ϕ_3, $n_{\text{radius}} = 1$ (b) ϕ_3, $n_{\text{radius}} = 3$ (c) ϕ_3, $n_{\text{radius}} = 5$

(d) ϕ_4, $n_{\text{radius}} = 1$ (e) ϕ_4, $n_{\text{radius}} = 3$ (f) ϕ_4, $n_{\text{radius}} = 5$

Fig. 12. Homogeneous flow scenario: Execution time of phases ϕ_3 and ϕ_4 for traffic shaping heuristics against simulation.

Figures 13(b) and 13(e) show the inverse relationship with n_{radius}. In ϕ_3, smaller the n_{radius}, the smaller the clusters, and hence reduced traffic. In ϕ_4, there are more clusters transmitting to the sink, and consequently more traffic and link utilization.

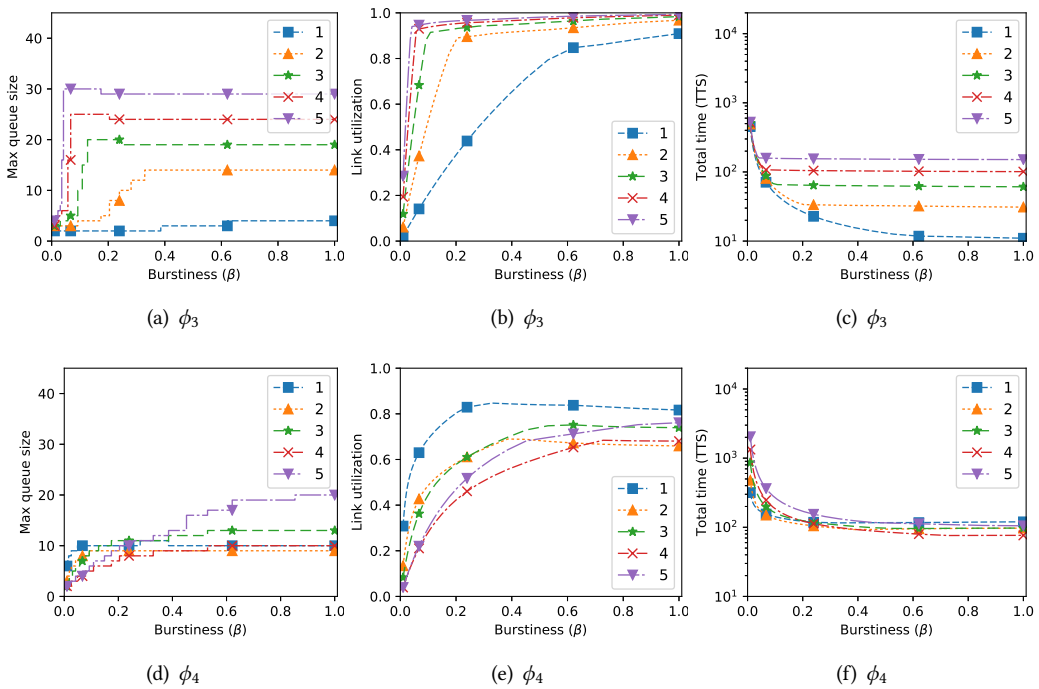
It is also worth noticing that the increase/decrease on the link utilization changes non-linearly with n_{radius}. This is because the number of nodes in each cluster grows with the square of the n_{radius}.

By looking at Figures 13(a) and 13(c) we can observe a property of the *LQ* heuristic (this also occurs for the other heuristics which are not shown for brevity). For all values of n_{radius}, both maximum queue size and total execution time remain constant (from $\beta > 0.4$). So even when link utilization is saturated, an increase in burstiness at flows' sources does not lead to worst queues and total time. We explain this phenomenon in Section 4.1. The same behavior can mostly be observed also during ϕ_4 in Figures 13(d) and 13(f).

4.3 Maximum queue size with heterogeneous load distribution

The results for maximum queue sizes for heterogeneous loads (see Figure 14) while tending to show the same trends as for homogeneous loads, present more chaotic behavior. BE performance is degraded. This implies that in more scenarios, applying traffic shaping is enough to reduce queue sizes compared to the case with homogeneous loads.

Also, in Figure 15, one can see that the link utilization due to heterogeneous load distribution, presents similar overall behavior when compared to the homogeneous scenario. This was expected, since the network load was intentionally designed to be approximately the same. But apart from that, we can see that the Max-S heuristic performs better than before, specially during phase ϕ_4



(a) ϕ_3 (b) ϕ_3 (c) ϕ_3

(d) ϕ_4 (e) ϕ_4 (f) ϕ_4

Fig. 13. LQ heuristic - (a) maximum queue size and, (b) link utilization and (c) total execution time, with varying burstiness and $n_{\text{radius}} = [1, 2, 3, 4, 5]$.

(Figures 15(d) to 15(f)), in which it provides higher link utilization when compared to LQ (in most cases for $\beta < 0.6$), while keeping queue size and total execution time approximately the same.

4.4 Phase execution time for heterogeneous load distribution

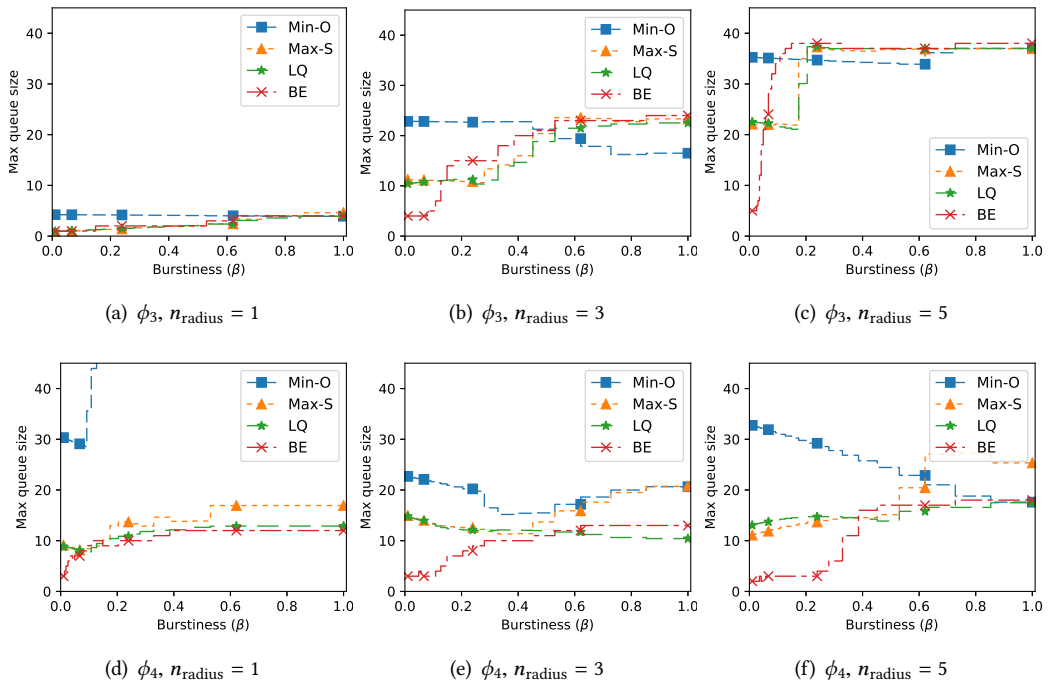
Total execution time again present the same results, since the total number of packets and average burstiness among nodes is the same for both scenarios. This is shown in Figure 16.

Once again, we take a closer look at the LQ heuristic alone, to understand the impact of n_{radius}. These results are show in Figure 17 for phases ϕ_3 and ϕ_4. There is a clear drop in performance in all metrics for this specific heuristic. The same drop is not observed for Max-S heuristic, that betters BE performance for heterogeneous flow sources.

To summarize, the heuristics Max-S and LQ, in both homogeneous and heterogeneous flow sources, perform close to that of BE that does not use traffic shaping. This means that by applying our heuristics for traffic shaping we are able to provide timing and resource usage determinism, and yet impose very little loss in terms of performance as compared to a best-effort solution.

5 RELATED WORK

In this section, we briefly discuss the differences between XDense and other sensor networks tailored to dense sensing and we go through a few systems that use similar mesh-grid network architectures. Then, we discuss some seminal works on real-time communication in multi-hop networks and traffic shaping to provide communication bounds for real-time applications.



(a) ϕ_3, $n_{\text{radius}} = 1$ (b) ϕ_3, $n_{\text{radius}} = 3$ (c) ϕ_3, $n_{\text{radius}} = 5$

(d) ϕ_4, $n_{\text{radius}} = 1$ (e) ϕ_4, $n_{\text{radius}} = 3$ (f) ϕ_4, $n_{\text{radius}} = 5$

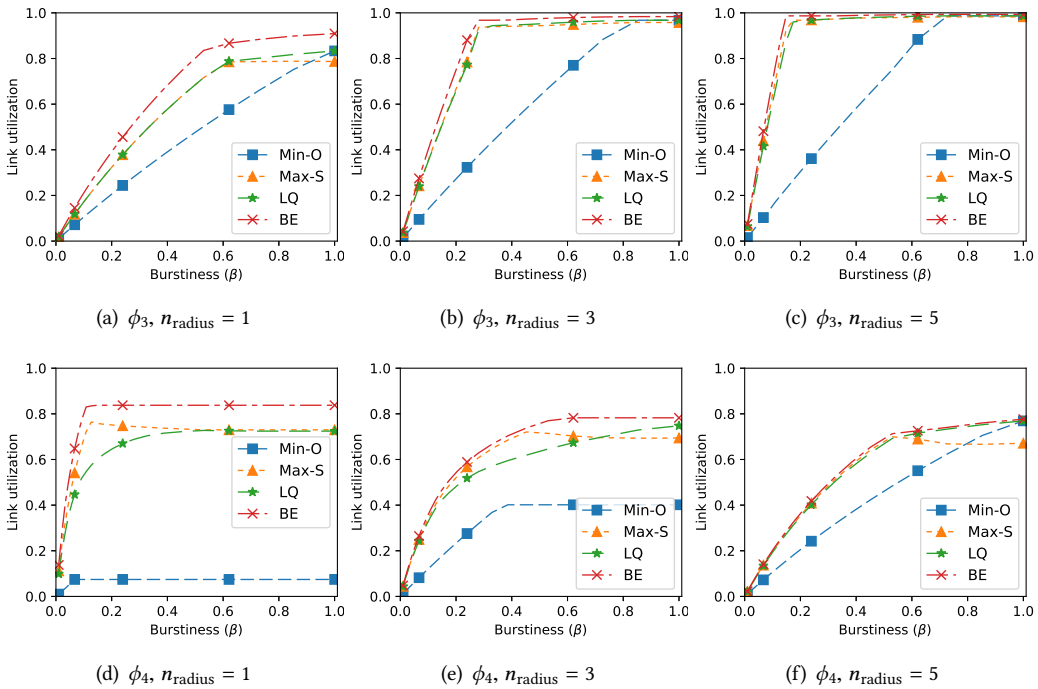
Fig. 14. Heterogeneous flow scenario: Maximum queue size for traffic shaping heuristics against simulation. Results are for phases ϕ_3 and ϕ_4 and n_{radius} set to 1, 3 and 5.

A multi-modal sensor network was proposed in [17], as a scalable sensor network with up to hundreds of nodes per square meter. However, due to the wireless nature of the links (infrared), contentions and collisions substantially increase the cost of communication. Their research leans more towards wireless sensor networks whose performance does not suit the application scenarios we focus on. Instead of wireless links, in [23], the authors use wired links to deploy few sensors in a grid network, to act as an electronic skin. However, nodes are interconnected using shared buses, approach that differs from ours.

More recently, the authors of [7] presented a modular and dense sensor network in a form factor of a tape, tailored to wearables. In these cases though, master-slave buses are used to interconnect nodes (through SPI or I2C), regardless of the shape of the network. Not only shared buses drastically decrease the opportunity for distributed processing due to their finite bandwidth that do not scale along with the number of nodes, but they also constraint the number of nodes due to limited address space and related electrical limitations. They are therefore not a scalable solution.

Related to the challenges of dense sensing on aircraft wings, in [15] the authors demonstrate a design integration and experimental assessment of a stretchable sensor network that is embedded inside a wing. The network consists of a passive and static structure, in which nodes are individually read from the exterior. The authors provide a good technological solution to integration related challenges, but do not address network communication issues.

XDense uses a 2D mesh network architecture that resembles common NoC architectures [1, 14]. Numerous works to achieve real-time guarantees have been proposed for NoCs. For instance, the authors of [34] proposed a worst-case analysis technique for priority-preemptive, wormhole



(a) ϕ_3, $n_{\text{radius}} = 1$ (b) ϕ_3, $n_{\text{radius}} = 3$ (c) ϕ_3, $n_{\text{radius}} = 5$

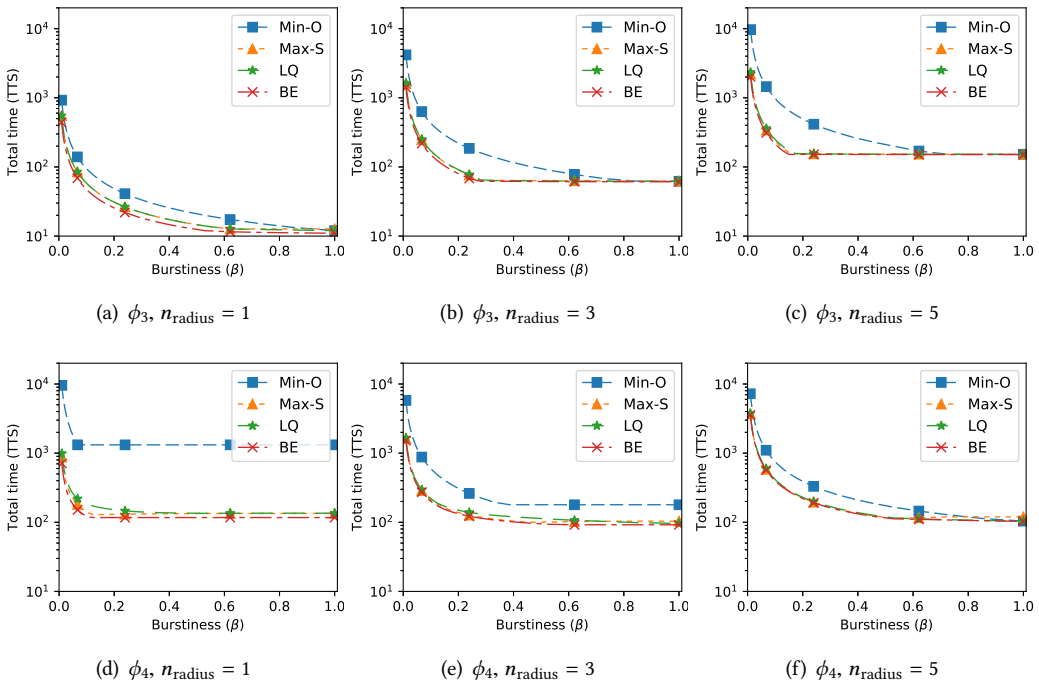
(d) ϕ_4, $n_{\text{radius}} = 1$ (e) ϕ_4, $n_{\text{radius}} = 3$ (f) ϕ_4, $n_{\text{radius}} = 5$

Fig. 15. Heterogeneous flow scenario: Link utilization for traffic shaping heuristics against simulation. Results are for phases ϕ_3 and ϕ_4 and n_{radius} set to 1, 2 and 5).

switched NoCs. This approach was later extended in [12], in which an end-to-end schedulability analysis was proposed for many-core systems. Additionally, in [27] authors provide methods to efficiently calculate the worst case bandwidth and latency bounds for real-time traffic streams on wormhole-switched NoCs with arbitrary topology. However, wormholing is tailored to parallel links, where very high bandwidth is required, and for large amount of data transfer between cores. Because XDense relies on non-prioritized packet switched serial communication, this approach does not apply to our network architecture.

More in line with our approach, initially proposed by Cruz [6], network calculus enables real-time communication for packet switched multi-hop point-to-point networks, addressing the issue of guaranteeing the delivery of messages with time constraints. This problem has been extensively researched since then; In [10], the authors survey the state-of-the-art of deterministic and probabilistic network calculus, by providing a review of service curve models of common schedulers along with different types of networks and methods for identification of a system's service curve representation. With a slightly different concept, the authors in [9] propose a feasibility analysis of periodic hard real-time traffic in packet-switched networks using first come first served queuing, without traffic shaping. Their framework suite real-time analysis of switched Ethernet, and can provide better results compared to network-calculus in some cases.

Traffic shaping to achieve tighter and deterministic bounds has also been investigated already. For example, in [35] the authors use rate controlled Earliest Deadline First scheduling in conjunction with per-hop traffic shaping to provide deterministic end-to-end delay guarantees. They identify the shaping parameters that result in maximal network utilization. This has also been studied for NoCs



(a) ϕ_3, $n_{\text{radius}} = 1$ (b) ϕ_3, $n_{\text{radius}} = 3$ (c) ϕ_3, $n_{\text{radius}} = 5$

(d) ϕ_4, $n_{\text{radius}} = 1$ (e) ϕ_4, $n_{\text{radius}} = 3$ (f) ϕ_4, $n_{\text{radius}} = 5$

Fig. 16. Heterogeneous flow scenario: Execution time of phases ϕ_3 and ϕ_4 computed for traffic shaping heuristics against simulation.

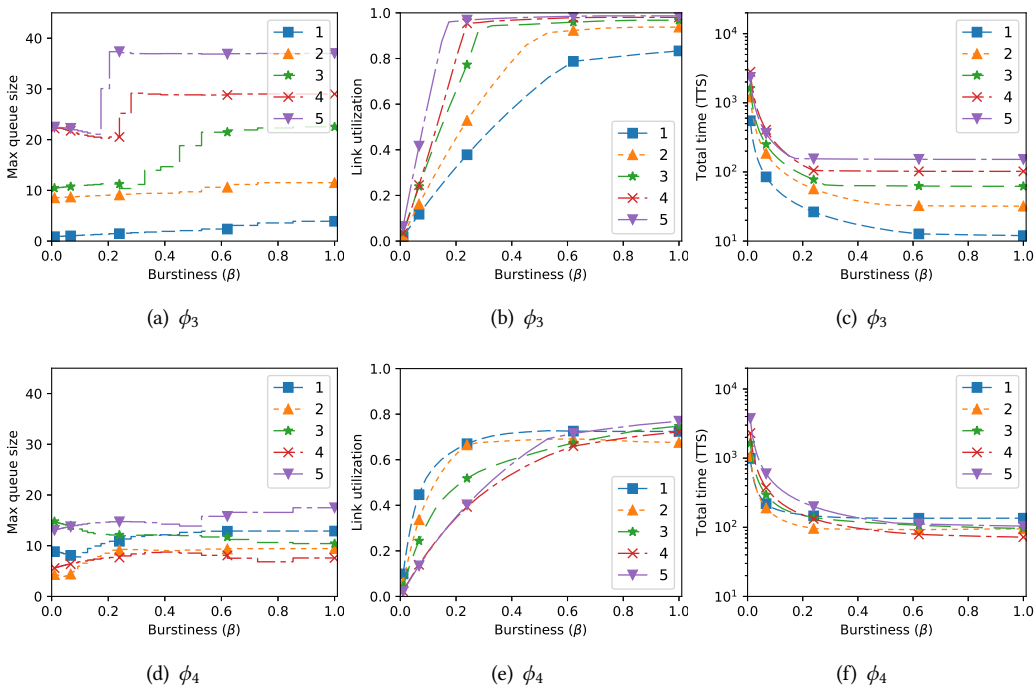
in [22] for worst-case response guarantees and buffer space optimization. Traffic shaping is also used on Ethernet networks to provide real-time guarantees by shaping the packet sources [36]. The authors provide an asynchronous traffic scheduling algorithm, which gives low delay guarantees, while keeping low implementation complexity.

6 CONCLUSIONS AND FUTURE WORK

The proposed traffic shaping heuristics enable us to endow XDense networks with real-time capabilities. We showed that the performance of XDense is approximately the same, with and without traffic shaping. This means that the proposed traffic shaping techniques allow determining timing and memory requirements while imposing minor performance overheads. The performance of XDense, in both homogeneous and heterogeneous scenarios also showcases its stable performance. However, further experiments exploiting distributed processing algorithms on CFD input data for AFC need to be performed (as discussed in Section 2.3).

Even though it has not been discussed in the paper, the analysis framework that we propose can also serve as a basis to reason on the dimensioning of the system; in the choice of network size, the clusters size, and other configuration that may impact on the performance. The framework was designed to allow modeling different clustering, event detection and actuation algorithms, to meet the demands of different application scenarios.

We also believe the proposed framework can be used in other kinds of synchronous multi-hop networks. The requirements are that the flow sources (the nodes from which the packets are generated) are known, and can be modeled according to our flow model. This should allow us to



(a) ϕ_3 (b) ϕ_3 (c) ϕ_3

(d) ϕ_4 (e) ϕ_4 (f) ϕ_4

Fig. 17. Link utilization (a), maximum queue size (b) and total execution time (c) with varying burstiness, for the LQ heuristic only, for $n_{\text{radius}} = [1, 2, 3, 4, 5]$.

shape every output/input flows on the network, and provide real-time guarantees based on the calculations. We think that there are opportunities on designing new traffic shaping heuristics that would provide reduced queue sizes and delays compared to the actual ones.

Improvements to the model can be made along many dimensions. One would be to bring in computational fluid dynamics data to analyze the performance of the model vs synthetic data. The model can also be improved with more accurate portrayal of hardware (for example, to consider the internal delays). One way to do this is to measure delays on real hardware and incorporate it. We have already made some progress along these lines [21].

ACKNOWLEDGMENTS

This work was supported by National Funds through FCT (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (CEC/04234).

REFERENCES

[1] Ankur Agarwal, Cyril Iskander, and Ravi Shankar. 2009. Survey of network on chip (noc) architectures & contributions. *Journal of engineering, Computing and Architecture* 3, 1 (2009), 21–27.

[2] Scott Anders, William Sellers, and Anthony Washburn. 2004. Active flow control activities at NASA Langley. In *2nd AIAA Flow Control Conference*. 2623.

[3] Richard Barry et al. 2008. FreeRTOS. *Internet, Oct* (2008).

[4] William K Blake. 2012. *Mechanics of Flow-Induced Sound and Vibration V2: Complex Flow-Structure Interactions*. Vol. 2. Elsevier.

[5] Louis N Cattafesta and Mark Sheplak. 2011. Actuators for active flow control. *Annual Review of Fluid Mechanics* 43 (2011), 247–272.

[6] R. L. Cruz. 1991. A calculus for network delay. I. Network elements in isolation. *IEEE Transactions on Information Theory* 37, 1 (Jan 1991), 114–131. https://doi.org/10.1109/18.61109

[7] Artem Dementyev, Hsin-Liu (Cindy) Kao, and Joseph A. Paradiso. 2015. SensorTape: Modular and Programmable 3D-Aware Dense Sensor Network on a Tape. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 649–658. https://doi.org/10.1145/2807442.2807507

[8] Rostislav Reuven Dobkin, Arkadiy Morgenshtein, Avinoam Kolodny, and Ran Ginosar. 2008. Parallel vs. Serial On-chip Communication. In *Proceedings of the 2008 International Workshop on System Level Interconnect Prediction (SLIP '08)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/1353610.1353620

[9] Xing Fan, Magnus Jonsson, and Jan Jonsson. 2009. Guaranteed real-time communication in packet-switched networks with FCFS queuing. *Computer Networks* 53, 3 (2009), 400 – 417. https://doi.org/10.1016/j.comnet.2008.10.018

[10] M. Fidler. 2010. Survey of deterministic and stochastic service curve models in the network calculus. *IEEE Communications Surveys Tutorials* 12, 1 (First 2010), 59–86. https://doi.org/10.1109/SURV.2010.020110.00019

[11] Jingcao Hu and Radu Marculescu. 2003. Energy-aware Mapping for Tile-based NoC Architectures Under Performance Constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference (ASP-DAC '03)*. ACM, New York, NY, USA, 233–239. https://doi.org/10.1145/1119772.1119818

[12] Leandro Soares Indrusiak. 2014. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture* 60, 7 (2014), 553 – 561. https://doi.org/10.1016/j.sysarc.2014.05.002

[13] Nobuhide Kasagi, Yuji Suzuki, and Koji Fukagata. 2009. Microelectromechanical systems-based feedback control of turbulence for skin friction reduction. *Annual review of fluid mechanics* 41 (2009), 231–251.

[14] 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.

[15] Fotis Kopsaftopoulos, Raphael Nardari, Yu-Hung Li, and FK Chang. 2015. Experimental identification of structural dynamics and aeroelastic properties of a self-sensing smart composite wing. In *Proceedings of the 10th International Workshop on Structural Health Monitoring, Stanford, CA*.

[16] Shashi Kumar, Axel Jantsch, J-P Soininen, Martti Forsell, Mikael Millberg, Johny Oberg, Kari Tiensyrja, and Ahmed Hemani. 2002. A network on chip architecture and design methodology. In *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*. IEEE, 105–112.

[17] Joshua Lifton, Michael Broxton, and Joseph A. Paradiso. 2005. Experiences and Directions in Pushpin Computing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*. IEEE Press, Piscataway, NJ, USA, Article 57. http://dl.acm.org/citation.cfm?id=1147685.1147753

[18] J. Loureiro, R. Rangarajan, B. Nikolic, L. Indrusiak, and E. Tovar. 2017. Real-time dense wired sensor network based on traffic shaping. In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 1–10. https://doi.org/10.1109/RTCSA.2017.8046307

[19] João Loureiro, Raghuraman Rangarajan, and Eduardo Tovar. [n. d.]. Demo Abstract: Towards the Development of XDense, A Sensor Network for Dense Sensing. *12th European Conference on Wireless Sensor Networks (EWSN)* ([n. d.]), 23. www.cister.isep.ipp.pt/ewsn2015/EWSN15PosterDemoProc_WEB.pdf#page=24

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

[21] João Loureiro, Pedro Santos, Raghuraman Rangarajan, and Eduardo Tovar. 2017. Simulation Module and Tools for XDense Sensor Network. In *Proceedings of the Workshop on Ns-3 (WNS3 '17)*. ACM, New York, NY, USA, 110–117. https://doi.org/10.1145/3067665.3067680

[22] Sorin Manolache, Petru Eles, and Zebo Peng. 2006. Buffer Space Optimisation with Communication Synthesis and Traffic Shaping for NoCs. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings (DATE '06)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 718–723. http://dl.acm.org/citation.cfm?id=1131481.1131683

[23] Behram F.T. Mistree and Joseph A. Paradiso. 2010. ChainMail: A Configurable Multimodal Lining to Enable Sensate Surfaces and Interactive Objects. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '10)*. ACM, New York, NY, USA, 65–72. https://doi.org/10.1145/1709886.1709899

[24] F. Palacios, J. Alonso, K. Duraisamy, M. Colonno, J. Hicken, A. Aranake, A. Campos, S. Copeland, T. Economon, A. Lonkar, et al. 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.

[25] Alexandros Pantelopoulos and Nikolaos G Bourbakis. 2010. A survey on wearable sensor-based systems for health monitoring and prognosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40, 1

(2010), 1–12.

[26] Ruiyi Que and Rong Zhu. 2012. Aircraft Aerodynamic Parameter Detection Using Micro Hot-Film Flow Sensor Array and BP Neural Network Identification. *Sensors* 12, 8 (2012), 10920–10929. https://doi.org/10.3390/s120810920

[27] D. Rahmati, S. Murali, L. Benini, F. Angiolini, G. De Micheli, and H. Sarbazi-Azad. 2013. Computing Accurate Performance Bounds for Best Effort Networks-on-Chip. *IEEE Trans. Comput.* 62, 3 (March 2013), 452–467. https://doi.org/10.1109/TC.2011.240

[28] J Reneaux et al. 2004. Overview on drag reduction technologies for civil transport aircraft. *ONERA: Tire a Part* 153 (2004), 1–18.

[29] Stephen K Robinson. 1991. Coherent motions in the turbulent boundary layer. *Annual Review of Fluid Mechanics* 23, 1 (1991), 601–639.

[30] Carlos Rodrigues, Carlos FÃĺlix, Armindo Lage, and Joaquim Figueiras. 2010. Development of a long-term monitoring system based on FBG sensors applied to concrete bridges. *Engineering Structures* 32, 8 (2010), 1993 – 2002. https://doi.org/10.1016/j.engstruct.2010.02.033

[31] Artur Saudabayev and Huseyin Atakan Varol. 2015. Sensors for robotic hands: A survey of state of the art. *IEEE Access* 3 (2015), 1765–1782.

[32] V Schmitt and F Charpin. 1979. Pressure distributions on the ONERA-M6-Wing at transonic Mach numbers. *Experimental data base for computer program assessment* 4 (1979).

[33] JH Seo, F Cadieux, R Mittal, E Deem, and L Cattafesta. 2018. Effect of synthetic jet modulation schemes on the reduction of a laminar separation bubble. *Physical Review Fluids* 3, 3 (2018), 033901.

[34] Zheng Shi and Alan Burns. 2008. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS '08)*. IEEE Computer Society, Washington, DC, USA, 161–170. http://dl.acm.org/citation.cfm?id=1397757.1397996

[35] Vijay Sivaraman, Fabio M. Chiussi, and Mario Gerla. 2006. Deterministic end-to-end delay guarantees with rate controlled {EDF} scheduling. *Performance Evaluation* 63, 4âĂŞ5 (2006), 509 – 519. https://doi.org/10.1016/j.peva.2005.04.002

[36] J. Specht and S. Samii. 2016. Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. 75–85. https://doi.org/10.1109/ECRTS.2016.27

[37] Lau Troy M, Gwin Joseph T, and Ferris Daniel P. 2012. How many electrodes are really needed for EEG-based mobile brain imaging? *Journal of Behavioral and Brain Science* 2012 (2012).

[38] Tony Washburn. 2010. Airframe Drag/Weight Reduction Technologies. *Green Aviation Summit-Fuel Burn Reduction, NASA Ames Research Centre* (2010).

[39] M Watson, AJ Jaworski, and NJ Wood. 2007. Application of synthetic jet actuators for the modification of the characteristics of separated shear layers on slender wings. *The Aeronautical Journal* 111, 1122 (2007), 519–529.