CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

# Journal Paper

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

**João Loureiro***
**Raghu R.**
**Borislav Nikolic**
**Leandro Soares Indrusiak**
**Eduardo Tovar***

*CISTER Research Centre

CISTER-TR-190710

2019/08/01

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

João Loureiro*, Raghu R., Borislav Nikolic, Leandro Soares Indrusiak, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: joflo@isep.ipp.pt, raghu@isep.ipp.pt, borni@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

## Abstract

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 that require real-time sensing and actuation, like active flow control 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 article, 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 link utilization, queue size, and execution time. With the proposed traffic-shaping heuristics, we endow XDense with the capabilities required for real-time applications.

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

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

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 that require real-time sensing and actuation, like active flow control 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 article, 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 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. 2019. Extensive Analysis of a Real-Time Dense Wired Sensor Network Based on Traffic Shaping. *ACM Trans. Cyber-Phys. Syst.* 3, 3, Article 27 (August 2019), 27 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 tend to be minimally priced. This makes it economically feasible

This work was supported by National Funds through FCT (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (CEC/04234).
Authors' addresses: J. Loureiro, R. Rangarajan, and E. Tovar, CISTER Research Centre, Rua Alfredo Allen, 535, 4200-135 PORTO, Portugal; emails: {joflo, raghu, emt}@isep.ipp.pt; B. Nikolic, Institut fur Datentechnik und Kommunikationsnetze, Technische Universitat Braunschweig, Universitatsplatz 2, 38106 Braunschweig, Germany / CISTER Research Centre, Rua Alfredo Allen, 535, 4200-135 PORTO, Portugal; email: borni@isep.ipp.pt; L. S. Indrusiak, Computer Science Department, University of York, Heslington, York YO10 5DD, United Kingdom; email: leandro.indrusiak@york.ac.uk.

2378-962X/2019/08-ART27 $15.00
https://doi.org/10.1145/3230872

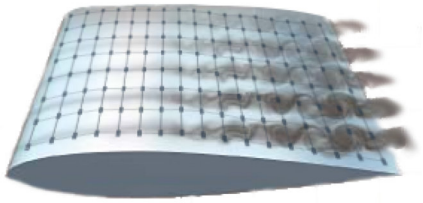


Fig. 1. Conceptual deployment of XDense for AFC.

to densely deploy sensor networks with very large quantities of computing nodes. Accordingly, it is possible to take a 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.

Such densely instrumented systems, however, pose 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.

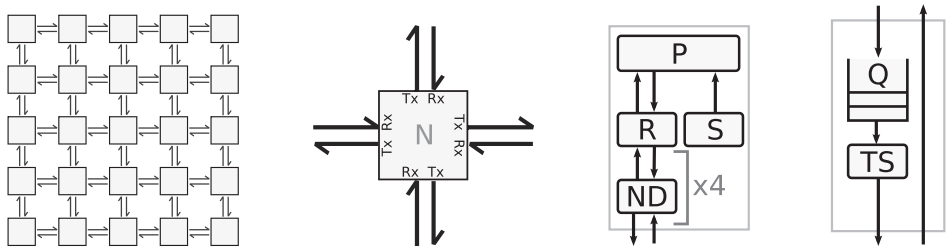
To further motivate our approach, let us consider an aerospace application scenario that may benefit from such dense cyber-physical systems (CPS). The drastic increase in demand for air transportation naturally motivates measures to reduce its environmental impact. The reduction of fuel consumption is important with regard to 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 and Sheplak [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 (e.g., pressure, temperature, and vibration sensors), which are deployed with some high

[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].




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

Fig. 2. (a) The XDense 2D 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), networking device (ND), and sensor (S). (d) ND architecture. The output port includes a queue (Q) and a traffic shaper (TS).

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 (Figure 2(a)). This resembles common Network-on-Chip (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 that XDense can be useful for other applications that require fast response times and dense networks of sensorssuch as (i) aerodynamic tests [26], (ii) structural monitoring [30], (iii) biomedical devices for the electroencephalogram [37], (iv) robotic e-skins [31], and (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 is 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 make decisions on the density of the deployment and minimum sampling rate requirements, and consequently network load. For example, in Loureiro et al. [20], targeting AFC, we proposed algorithms to enable distributed turbulent airflow 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 of XDense. These two properties are therefore the focus of this article.

### 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, this work makes the following two contributions:

(i) proposes three heuristics to shape the traffic in the network and (ii) develops a mathematical framework to model and analyze the application and network and provides upper bounds on communication delays, application execution time, and maximum buffer requirements.

As an extension to the work presented in Loureiro et al. [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 article 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 article and offers comments on potential future research directions.

## 2 XDENSE ARCHITECTURE AND PRINCIPLES OF OPERATION

### 2.1 Architecture and Topology of XDense

XDense is a 2D mesh network whose topology and node architecture are inspired from traditional NoC designs. Despite its similarities with NoCs, XDense differs in its size and node count. XDense is meant to be deployed on large surfaces (e.g., 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).

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 NDs. NDs are full-duplex serial communication ports. We use serial links because they are widely available in COTS microcontrollers and provide low complexity and low footprint at low cost (compared to parallel links found in NoC[14]). For example, Dobkin et al. [8] show that the utilization of parallel links beyond 2mm 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 then be transmitted serially over the network. All network transfers are nonpreemptive 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 twofold: 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. In addition, all temporal units are normalized and

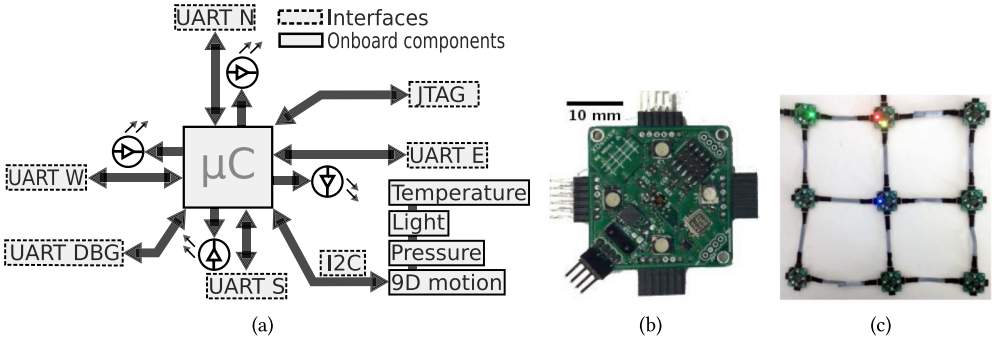



Fig. 3. (a) Node schematic showing each major component of the system. (b) Node prototype. (c) A 3 × 3 network.

quantified in terms of Transmission Time Slots (TTS), which is the time required to transmit a single packet.

### 2.2 Hardware Implementation

For realizing the preceding design of XDense, a custom-designed Integrated Circuit (IC) provides the best-fit solution. However, this reduces design flexibility and might become a single application solution. For this reason, we use a microcontroller ($\mu$C) and other COTS to prototype the XDense node and network. The rest of this section provides context to this discussion with an overview of the prototype that we have developed, shown in Figure 3.

To implement XDense, we chose the Atmel ATSAM4N8A $\mu$C. It is based on the 32-bit ARM Cortex-M4 RISC processor, which is a mid-range general-purpose $\mu$C that runs at up to 100MHz 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 allow 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 multitasking.

The schematics, prototype node, and a 3 × 3 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 Loureiro et al. [19].

### 2.3 AFC Application Scenario

We now revisit the AFC application discussed in Section 1. Although not used as input in the analysis ahead, with the focus being to prove the real-time 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. 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 3D, 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.

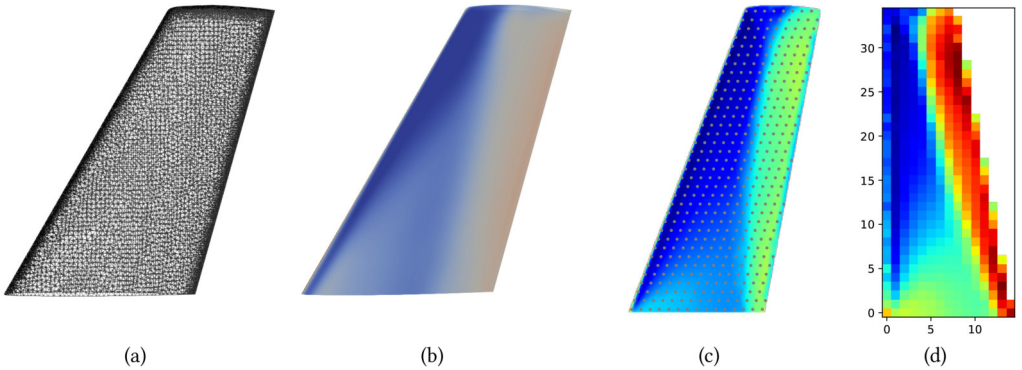



Fig. 4. (a) Pressure distribution over a wing's surface. (b) Data of a single time frame, from CFD simulation, as input for XDense. (c) Sensors displacement. (d) Normalized data, as seen by each sensor.

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 airflow, and temporal pressure and temperature data of the surface of the wing are produced (see 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. Figure 4(c) and (d) illustrate sensor deployment and sensor data, respectively.

More specifically, the sensor data are generated and imported using the SU2 simulator for multiphysics [24]. SU2 is a reliable open source CFD simulator that 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 and therefore not influencing in any way the results presented in this article.

### 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 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

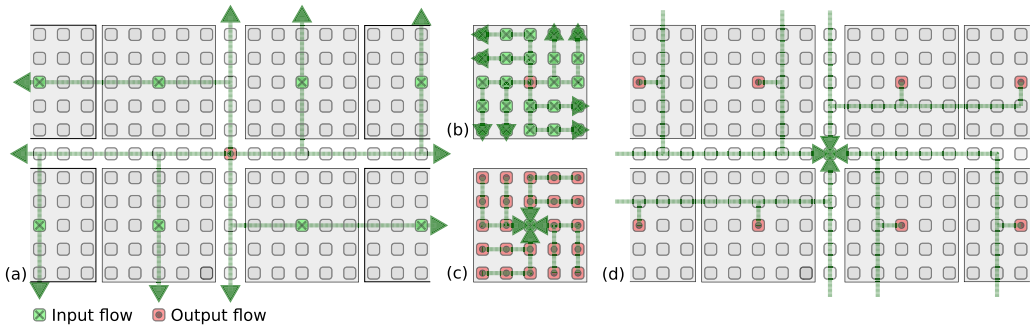



Fig. 5. Example 45 × 45 network, with a single central sink, in this case with $n_{\text{radius}} = 2$. Application phases: $\phi_1$, sink requests data from cluster heads (a); $\phi_2$, cluster heads in turn send a multicast request to nodes in their cluster; $\phi_3$, nodes send sensor data back to their respective cluster-head; and (d) $\phi_4$, cluster heads process received data and send result to the sink.

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: (i) the nodes are clustered and one node in each cluster (cluster head) is in charge of aggregating and prprocessing the data; (ii) the execution of the application is divided logically in subsequent phases; and (iii) the network implements routing schemes that guarantee 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 preprocessing at the selected cluster heads. Our tested solution implements nonoverlapping "square" clusters—the network topology being a 2D grid of $X$ times $Y$ nodes, and all clusters are nonoverlapping and of size $n_{\text{size}} \times n_{\text{size}}$, with $n_{\text{size}} \leq X$ and $n_{\text{size}} \leq Y$. $n_{\text{size}}$ must be a positive odd number, and the cluster head is the node located at the "center" of the square. The cluster size $n_{\text{size}}$ is defined through the system parameter $n_{\text{radius}}$ 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 $n_{\text{radius}}$ given by $(2 \times n_{\text{radius}} + 1)^2$.

Nodes arbitrate their role on the network at runtime (to act either as a cluster head or a normal node). They do this on reception of a packet from the sink containing the packet origin and the $n_{\text{radius}}$ 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 preprocessing and compression algorithms to be used are application specific and are outside the scope of this article. We have, however, discussed application-specific data processing issues in previous works (see Loureiro et al. [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: $\phi_1, \phi_2, \phi_3$, and $\phi_4$. The first phase is started by the sink, when it requests data from the cluster heads. Specifically, the four phases are as follows:

- Phase $\phi_1$. The sink requests the cluster heads of all clusters to send the processed data.
- Phase $\phi_2$. On receiving the request from the sink, the cluster heads in turn request the nodes of their respective clusters to send their data.
- Phase $\phi_3$. Every node of each cluster transmits its sensed data to its cluster head.
- Phase $\phi_4$. The cluster heads process the received data and transmit the result back to the sink.

Note that clusters in the network may not be in sync with one another with respect to the phase of their execution. The second phase ($\phi_2$) for instance, start in each cluster with a different time offset; Offset that is proportional to the distance between the cluster head of each cluster and the sink. We assume that all nodes of a cluster 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, as nodes have could rely on a time synchronization protocol for that (what we do not consider in this work).

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 (e.g., a wireless link). Figure 5 shows the four phases in chronological order.

*2.4.3 Spatial Isolation through Routing Schemes.* The four phases described earlier 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 (respectively, Y-X), packets are first routed along the X (respectively, Y) dimension and then along the Y (respectively, 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 $\phi_1$ through $\phi_3$ use one of the following two routing algorithms, sometimes called *counterclockwise dimension routing* (see Figure 5(a)–(c)). The starting dimension (X or Y) depends on the quadrant in which the destination node is, relative to the origin of the packet. For phase $\phi_4$, 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 $\phi_4$, sent by the cluster head to the sink. This routing scheme results in flows from phase $\phi_4$ to travel orthogonal to the flows from phases $\phi_1$, $\phi_2$, and $\phi_3$, 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, 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

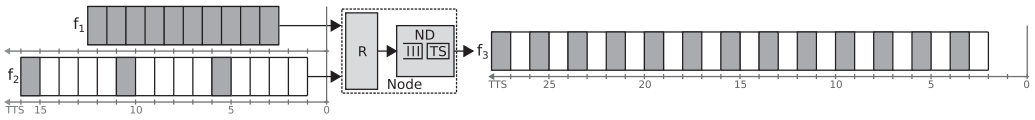



Fig. 6. Traffic shaper example scenario: two input flows shaped by an intermediate node as an output flow. Parameters for the input flows are $f_1 = \{O = 2.5, \beta = 1, \sigma = 10\}$ and $f_2 = \{O = 1, \beta = \frac{1}{5}, \sigma = 3\}$. The resulting flow is $f_3 = \{O = 2, \beta = \frac{1}{2},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

exchanged between the nodes involved in phase $\phi_i$. 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 follows:

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

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 $\sigma$ and $\beta$ 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 next.

*Example 3.1 (9 Degrees of Freedom Motion Sensor).* Consider a 9 degrees of freedom 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 10 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 earlier, the sending of all packets by the source node of the corresponding flow $f$ is done according to its parameters $(O, \sigma, \beta)$; these three parameters allow for a precise timing and sending rate at the source node of $f$. Note that for simplicity, hereafter we shall use 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 $f^{\text{out}}$ at that port is a superposition of all of these competing flows. As such, $f^{\text{out}}$ may present an irregular packet transmission pattern and a rate that can no longer be modeled using the three parameters $(O, \sigma, \beta)$.

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 $f_k^{\text{in}}$ starts at time $O_k$ and has a duration defined as $\ell_k = \frac{\sigma_k}{\beta_k}$. In other words, flow $f_k$ sends all of its packets after $\ell_k$ 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 $f_3^{\text{out}}$ that is not a superposition of the two input flows but rather is preset deterministic patterns that can be modeled using the three parameters $(O, \sigma, \beta)$.

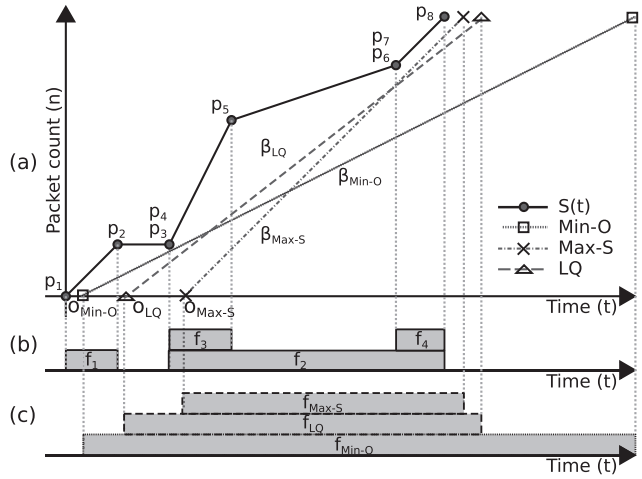



Fig. 7. Traffic-shaping heuristicsInput and output flows using the proposed heuristics (a). A time line showing offset and duration of arriving flows (b) and departure flows (c).

In other words, 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 $(O, \sigma, \beta)$. For that, we first identify the set of input flows $f_k^{\text{in}}$ (with $k = 1, 2, \ldots$) at every output port of every node in the network, and based on the respective parameters $(O_k, \sigma_k, \beta_k)$ 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 corresponds to the incoming flows that define the number of packets to be sent over time from the output port, which depends on the starting time and duration of all competing input flows. Three possible resulting flows $f^{\text{out}}$ 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 of the output ports.[3] We make two important assumptions regarding the flows and their routing.

**Assumption 1.** During phases $\phi_3$ and $\phi_4$, in every node, all of the packets entering by a given input port are assumed to exit through a single output port.

**Assumption 2.** There is 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 $(O, \sigma, \beta)$. If any of these input flows, say $f_k^{\text{in}}$, 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. This requirement must be satisfied for all input flows competing for $p_2$, and interactively it must be satisfied as well for all output ports of the upstream routers, until the traffic shaper at the source nodes of all 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

[3]Note that it has been proven in Sivaraman et al. [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.

$f_1$ 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.

### 3.3 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 $F^{\text{in}}$ 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 $f_k^{\text{in}}$ at a given time $t$ (TTS). When $t$ is earlier than the starting instant $O_k^{\text{in}}$ 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 $\sigma_k^{\text{in}}$ of packets sent by $f_k^{\text{in}}$, with $\ell_k$ being the duration of the flow. Between the two bounds $O_k^{\text{in}}$ and $O_k^{\text{in}} + \ell_k$, the function increases steadily from 0 to $\sigma_k^{\text{in}}$ with a constant slope of $\beta_k^{\text{in}}$.

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 input flows $f_k^{\text{in}}$. 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 $L^{\text{out}}$ of slope $\beta^{\text{out}}$ and passing through the point $(O^{\text{out}}, 0)$, the *vertical* distance $\text{dv}_j^{\text{out}}$ between a point $(t_j, S(t_j)) \in S(t)$, $\forall t_j \in \mathcal{T}$, and the line $L^{\text{out}}$ represents the number of packets being buffered at time $t$ at that output port. The *horizontal* distance $\text{dh}_j^{\text{out}}$ between a point $(t_j, S(t_j)) \in S(t)$, $\forall t_j \in \mathcal{T}$ and $L^{\text{out}}$ represents the delay (induced by the shaper) that all 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 $\sigma^{\text{out}}$ that is the same for all heuristics proposed. Considering that the shaper is not allowed to drop any packet, it is naturally the sum of the size of all input flows $f_k^{\text{in}}$—that is,

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

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

*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 subsequent packets at the highest admissible rate—that is, with the highest burstiness $\beta^{\text{out}}$ 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 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 a 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 the Min-O departure curve with $\beta^{\text{out}}$ 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 sends all 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 $\beta^{\text{out}}$ such that the line with slope $\beta^{\text{out}}$ 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 the Max-S departure curve with $\beta^{\text{out}}$ as $\beta_{\text{Max-S}}$. The offset $O^{\text{out}}$ in Max-S is simply set to the X-intercept of the line of slope $\beta^{\text{out}}$ 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)—that is,

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

After computing the offset $O^{\text{out}}$, 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 readjustment must be performed after computing $O^{\text{out}}$, because doing it before would in some cases allow a packet to be forwarded before it even arrives—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 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 $O^{\text{out}}$ can be computed easily by positioning the line of slope $\beta^{\text{out}}$ on every point $(t_j, S(t_j))$, $\forall t_j \in \mathcal{T}$, and retaining the maximum X-intercept of all of 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 $L^{\text{out}}$ that minimizes the distance between every point $(t_j, S(t_j)) \in S(t)$, $\forall t_j \in \mathcal{T}$ and $L^{\text{out}}$. 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 $O^{\text{out}}$ such that the line of slope $\beta^{\text{out}}$ 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 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 interfering flows $f_k^{\text{in}}$ are known. For each such output port, the resulting output flow $f^{\text{out}}$ 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 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 $\beta^{\text{out}}$ that intersects with the x-axis at the point $(O^{\text{out}}, 0)$. We denote this line as $L^{\text{out}}$ 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 $\text{dv}_j^{\text{out}}$ between the point $(t, S(t))$ and the point $(t, L^{\text{out}}(t))$ on the line $L^{\text{out}}$. 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 subfunction is linear, it can easily be shown 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 $t \geq 0$—that is,

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

This holds true because every subfunction of $S(t)$ is a segment that is either of the following:

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

Similarly, the transmission delay at any time instant $t$ is given by the *horizontal* distance $\text{dh}_j^{\text{out}}$ between the point $(t, S(t))$ and the point of y-coordinate $S(t)$ on the line $L^{\text{out}}$. 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

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

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

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

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 article, 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 starts and when it ends. However, phases may overlap in time and happen simultaneously. For instance, for the application scenario considered in this article, a cluster head located close to the sink may enter phase $\phi_2$ long before a cluster head that is far from the sink (since it receives the request from phase $\phi_1$ sooner). For simplicity, we assume in this work that a phase ends when a given node has received all packets sent to it. For example, the time at which all cluster heads have received their requested data marks the end of phase $\phi_3$, and the time at which the sink has received all processed data marks the end of phase $\phi_4$. As such, we compute the execution time of a phase as the relative time instant at which all four input flows of that given node—a cluster head for phase $\phi_3$ and the

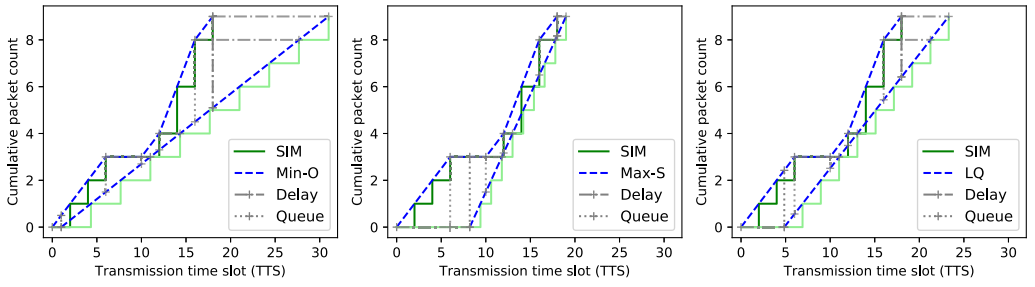



Fig. 8. Cumulative arrival/departure curves for a single node using Min-O (a), Max-S (b), and LQ (c) heuristics.

sink for phase $\phi_4$—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 $f^{\text{in}}$ 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 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 flow characteristics were chosen 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) are a stair function that presents the superposition of all 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. Figures 8(a-c) also show 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. However, the LQ heuristic leads to smaller queues, with a slightly longer execution time.

Although these results provide an intuition on the tradeoffs between the heuristics proposed, they do not depict the results of multihop 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: $\phi_1$, $\phi_2$, $\phi_3$, and $\phi_4$. In the first phase, $\phi_1$, the unique sink node requests all cluster heads to send their data; in phase $\phi_2$, the cluster heads perform another request to all nodes of their respective cluster; in phase $\phi_3$, the nodes reply to the cluster heads by sending them the sensed data; and in phase $\phi_4$, the cluster heads process the data received and transmit the result back to the sink. Considering that there is no network congestion in phases $\phi_1$ and $\phi_2$—because all 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 $\phi_3$ and $\phi_4$. We shall therefore focus *only* on phases $\phi_3$ and $\phi_4$, 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 $45 \times 45 = 2{,}025$ nodes with a unique sink located at the center of the grid. Figure 5 depicts a close-up 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 $n_{\text{radius}}$ 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, $n_{\text{radius}}$ 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 refer to as 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 (e.g., NoCs). 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]

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

**Evaluation criteria and methodology:** For each of the three heuristics Min-O, Max-S, and LQ, we evaluate the maximum queue sizes and the end-to-end execution time of the phases $\phi_3$ and $\phi_4$. 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 run 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 $\phi_3$ and $\phi_4$ (i.e., 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 $\beta$ of flows from phases $\phi_3$ and $\phi_4$ from 0.02 to 1 by step of 0.02, for different cluster sizes.

The message size $\sigma$ differs for each phase. For phases $\phi_1$ and $\phi_2$, a single packet is generated at the sink and cluster head ($\sigma = 1$). In phase $\phi_3$, each node outputs a flow with message size $\sigma = 4$. At the end of $\phi_3$, the cluster head receives a total of four packets per each node on its cluster. Subsequently, each cluster head outputs a flow with message size $\sigma$, as the sum of all of these packets plus four 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 airflow 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 tradeoff has a compensatory effect on the overall number of packets transmitted to the sink.

In the heterogeneous flow scenario, flows generated at phases $\phi_3$ and $\phi_4$ 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.

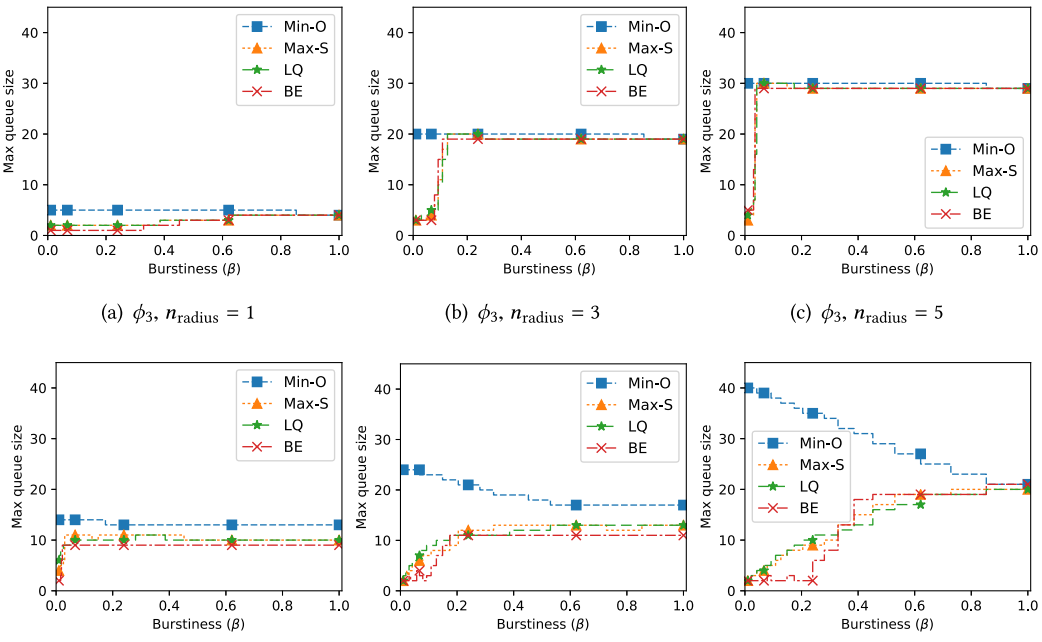
In our results, we compare the performance of both heterogeneous and homogeneous scenarios. To do this fairly, we guarantee that for both homogeneous (HO) and heterogeneous (HE) network load distribution, the sum of burstiness of all flows, as well as the sum of all message sizes, are equal. In other words, $\sum \beta^{HO} = \sum \beta^{HE}$ 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 $(O, \sigma, \beta)$ of all traffic shapers in the network. Then we use Equation (7) on every shaper to compute the maximum queue size of the corresponding node, and, finally, we retain the maximum queue size of all nodes in the network.

As we can see in Figure 9(a) and (b), in phase $\phi_3$ the queues are smaller for smaller clusters ($n_{\text{radius}}$). This is expected because 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 $\phi_4$, because 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 (e). In other words, smaller clusters do not imply longer queues. The reason for this counterintuitive result can be unveiled by looking at the *utilization* of the four input links of the sink.




(a) $\phi_3$, $n_{\text{radius}} = 1$ (b) $\phi_3$, $n_{\text{radius}} = 3$ (c) $\phi_3$, $n_{\text{radius}} = 5$

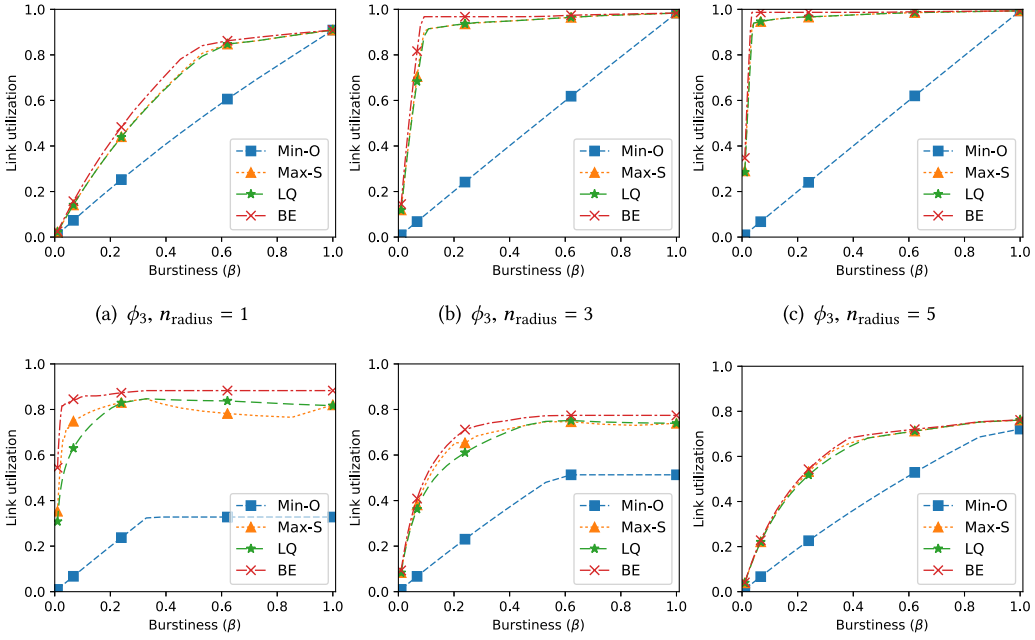
(d) $\phi_4$, $n_{\text{radius}} = 1$ (e) $\phi_4$, $n_{\text{radius}} = 3$ (f) $\phi_4$, $n_{\text{radius}} = 5$

Fig. 9. Homogeneous flow scenario: Maximum queue size for traffic shaping heuristics against simulation. Results are for phases $\phi_3$ (a-c) and $\phi_4$ (d-f) with $n_{\text{radius}}$ set to 1, 3 and 5.

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 $\phi_4$) divided by the time (number of TTS) it takes for all of 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 10(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 noting 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 $\phi_4$ for $n_{radius} = 3$ and 5, shown in Figure 9(e) and (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.

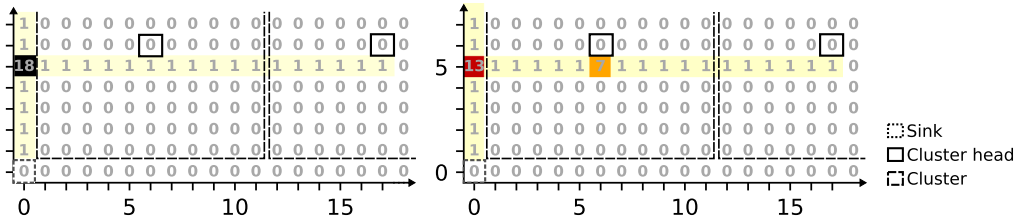
However, in cases with lower link utilization and burstiness, BE yields shorter queue sizes. In these cases, the network is underutilized such that BE still does not lead to excessive load on the network. However, performing traffic shaping imputes on unnecessary buffering by nodes and consequently greater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




(a) $\phi_3$, $n_{\text{radius}} = 1$ (b) $\phi_3$, $n_{\text{radius}} = 3$ (c) $\phi_3$, $n_{\text{radius}} = 5$

(d) $\phi_4$, $n_{\text{radius}} = 1$ (e) $\phi_4$, $n_{\text{radius}} = 3$ (f) $\phi_4$, $n_{\text{radius}} = 5$

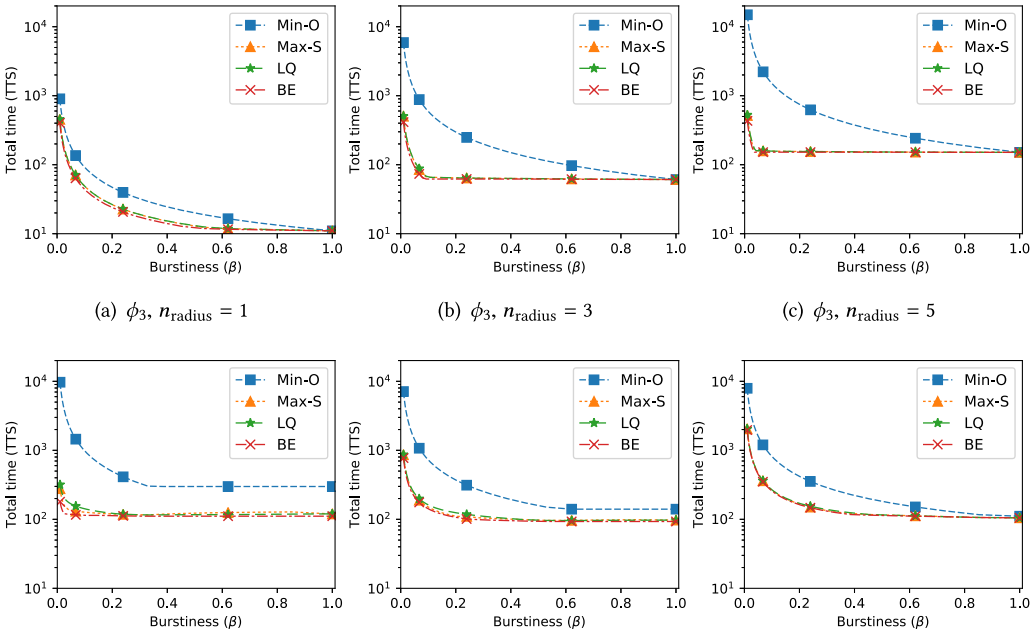
Fig. 10. Homogeneous flow scenario: Link utilization for traffic shaping heuristics against simulation. Results are for phases $\phi_3$ (a-c) and $\phi_4$ (d-f) with $n_{\text{radius}}$ set to 1, 3 and 5.




(a) BE, $\phi_4$, $n_{\text{radius}} = 5$ (b) LQ, $\phi_4$, $n_{\text{radius}} = 5$

Fig. 11. Queue size density map of the top-right quadrant of the network (17 × 7 nodes), for heuristics LQ (a) and BE (b). The X and Y axis are node coordinates relative to the sink.

The aforementioned effect is shown in Figure 11. 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 leftmost nodes in the network are usually where the bottleneck happens. Using BE, for example, in Figure 11(a), the node located at coordinates $(x, y) = (0, 5)$ gets up to 18 packets queued, as 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 11(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 leftmost node located at $(0, 5)$, thus reducing the maximum queue size at the nodes aligned with the sink and comparatively, from 18 packets using BE to 13 using LQ.




(a) $\phi_3$, $n_{\text{radius}} = 1$ (b) $\phi_3$, $n_{\text{radius}} = 3$ (c) $\phi_3$, $n_{\text{radius}} = 5$

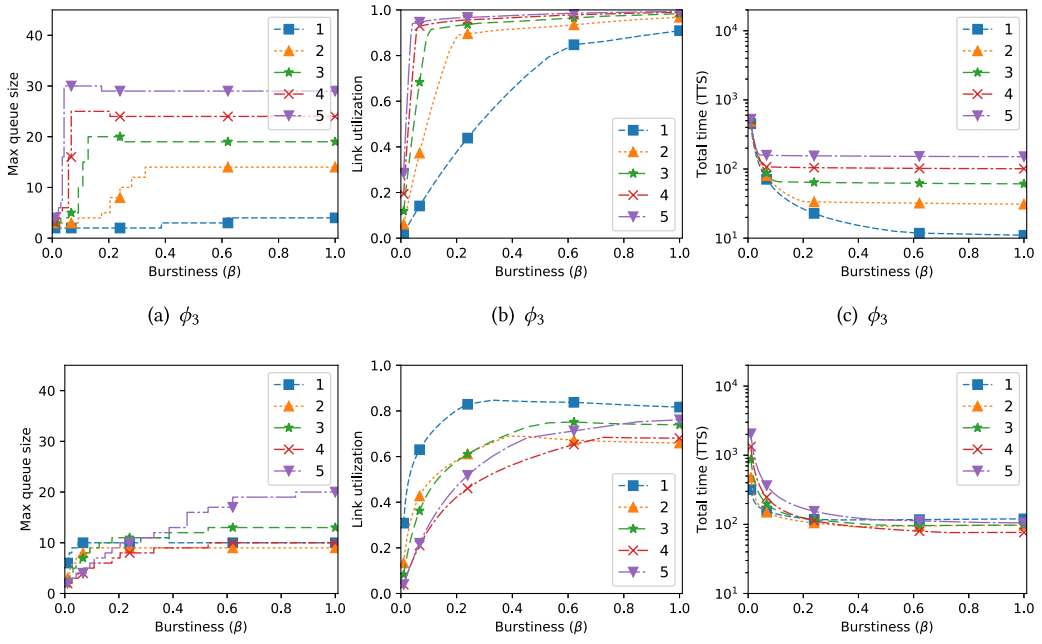
(d) $\phi_4$, $n_{\text{radius}} = 1$ (e) $\phi_4$, $n_{\text{radius}} = 3$ (f) $\phi_4$, $n_{\text{radius}} = 5$

Fig. 12. Homogeneous flow scenario: Execution time of phases $\phi_3$ (a-c) and $\phi_4$ (d-f) for traffic shaping heuristics against simulation.

Another interesting observation occurs during $\phi_4$ for the method Min-O. The maximum queue size gets smaller with increased burstiness. This is very counterintuitive because 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 $\beta$, and thus for low values of $\beta$, the first points $\in \mathcal{T}$ (depicted by $p_1$, $p_2$, etc.) are farther from the origin. Considering that Min-O selects a point close to the origin as the "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 (particularly 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 $\phi_3$ and $\phi_4$ 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 a 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 $\phi_3$) and the sink (for phase $\phi_4$). 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.




(a) $\phi_3$ (b) $\phi_3$ (c) $\phi_3$

(d) $\phi_4$ (e) $\phi_4$ (f) $\phi_4$

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

From the preceding results, we observe that the LQ heuristic performs better overall. We vary $n_{\text{radius}}$ from 1 to 5, with $\beta$ varying as before, for both $\phi_3$ and $\phi_4$. The results are shown in Figure 13. Figure 13(b) and (e) show the inverse relationship with $n_{\text{radius}}$. In $\phi_3$, the smaller the $n_{\text{radius}}$, the smaller the clusters, and hence reduced traffic. In $\phi_4$, there are more clusters transmitting to the sink, and consequently more traffic and link utilization.

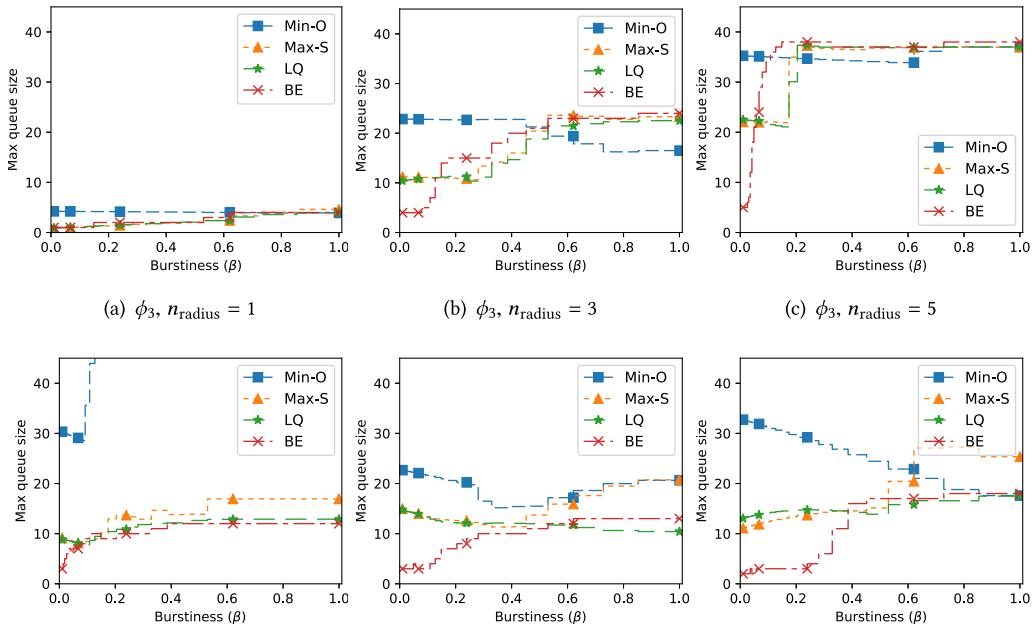
It is also worth noting that the increase/decrease on the link utilization changes nonlinearly with $n_{\text{radius}}$. This is because the number of nodes in each cluster grows with the square of the $n_{\text{radius}}$.

By looking at Figure 13(a) and (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 $n_{\text{radius}}$, both maximum queue size and total execution time remain constant (from $\beta > 0.4$). Thus, 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 $\phi_4$ in Figure 13(d) and (f).

### 4.3 Maximum Queue Size with Heterogeneous Load Distribution

The results for maximum queue sizes for heterogeneous loads (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.

In addition, 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, as the network load was intentionally designed to be approximately the same. But apart




(a) $\phi_3$, $n_{\text{radius}} = 1$ (b) $\phi_3$, $n_{\text{radius}} = 3$ (c) $\phi_3$, $n_{\text{radius}} = 5$

(d) $\phi_4$, $n_{\text{radius}} = 1$ (e) $\phi_4$, $n_{\text{radius}} = 3$ (f) $\phi_4$, $n_{\text{radius}} = 5$

Fig. 14. Heterogeneous flow scenario: maximum queue size for traffic-shaping heuristics against simulation. Results are for phases $\phi_3$ (a-c) and $\phi_4$ (d-f) and $n_{\text{radius}}$ set to 1, 3 and 5.

from that, we can see that the Max-S heuristic performs better than before, especially during phase $\phi_4$ (Figure 15(d)–(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 presents the same results, as the total number of packets and average burstiness among nodes is the same for both scenarios. This is shown in Figure 16.

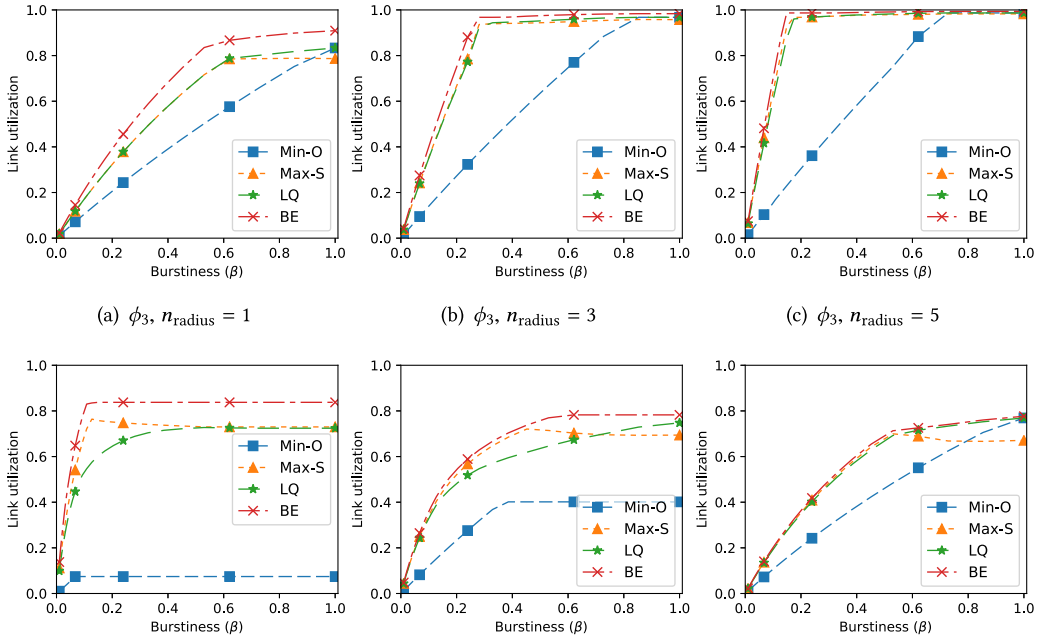
Once again, we take a closer look at the LQ heuristic alone to understand the impact of $n_{\text{radius}}$. These results are show in Figure 17 for phases $\phi_3$ and $\phi_4$. There is a clear drop in performance in all metrics for this specific heuristic. The same drop is not observed for the Max-S heuristic, that improves 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, which 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 multihop networks and traffic shaping to provide communication bounds for real-time applications.

A multimodal sensor network was proposed in Lifton et al. [17] as a scalable sensor network with up to hundreds of nodes per square meter. However, due to the wireless nature of the links




(a) $\phi_3$, $n_{\text{radius}} = 1$ (b) $\phi_3$, $n_{\text{radius}} = 3$ (c) $\phi_3$, $n_{\text{radius}} = 5$

(d) $\phi_4$, $n_{\text{radius}} = 1$ (e) $\phi_4$, $n_{\text{radius}} = 3$ (f) $\phi_4$, $n_{\text{radius}} = 5$

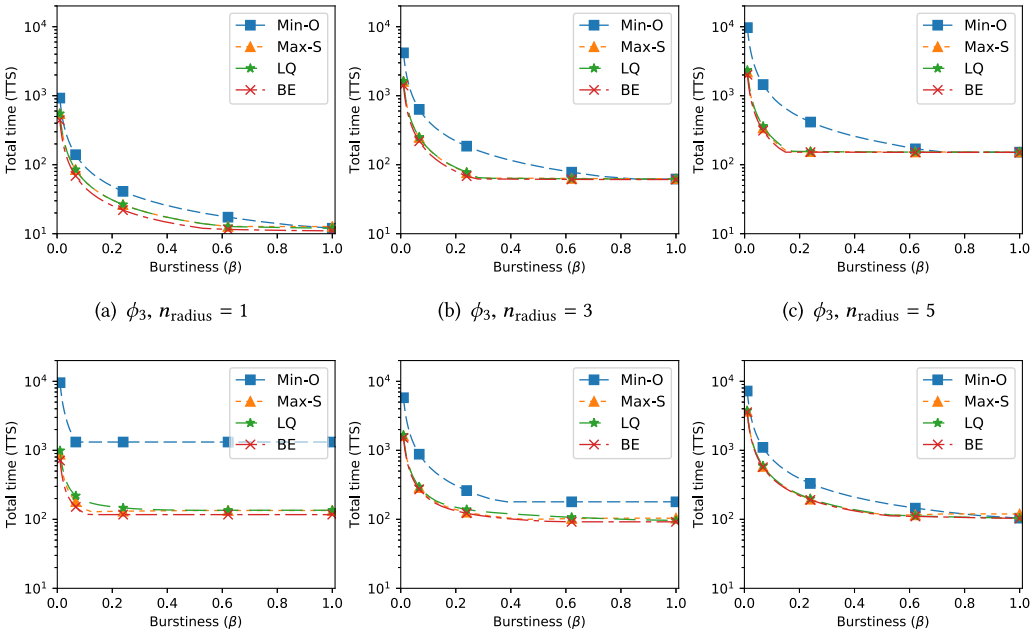
Fig. 15. Heterogeneous flow scenario: link utilization for traffic-shaping heuristics against simulation. Results are for phases $\phi_3$ (a-c) and $\phi_4$ (d-f) and $n_{\text{radius}}$ set to 1, 2 and 5.

(infrared), contentions and collisions substantially increase the cost of communication. Their research leans more toward wireless sensor networks whose performance does not suit the application scenarios on which we focus. Instead of wireless links, Mistree and Paradiso [23] use wired links to deploy few sensors in a grid network, to act as an electronic skin. However, nodes are interconnected using shared buses, an approach that differs from ours.

More recently, Dementyev et al. [7] presented a modular and dense sensor network in a form factor of a tape, tailored to wearables. In their work, however, master-slave buses are used to interconnect nodes (through SPI or I2C), regardless of the shape of the network. Not only do 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, Kopsaftopoulos et al. [15] 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 they 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, Shi and Burns [34] proposed a worst-case analysis technique for priority-preemptive, wormhole-switched NoCs. This approach was later extended in the work of Indrusiak [12], in which an end-to-end schedulability analysis was proposed for many-core systems. Additionally, Rahmati et al. [27] provide methods to efficiently calculate the worst-case bandwidth and latency bounds for




(a) $\phi_3$, $n_{\text{radius}} = 1$ (b) $\phi_3$, $n_{\text{radius}} = 3$ (c) $\phi_3$, $n_{\text{radius}} = 5$

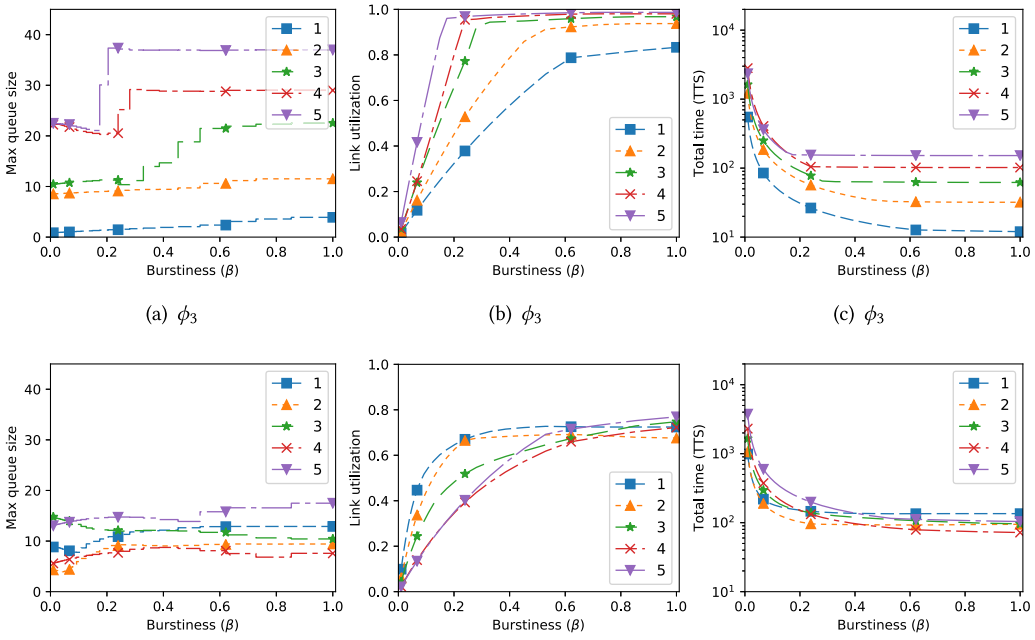
(d) $\phi_4$, $n_{\text{radius}} = 1$ (e) $\phi_4$, $n_{\text{radius}} = 3$ (f) $\phi_4$, $n_{\text{radius}} = 5$

Fig. 16. Heterogeneous flow scenario: Execution time of phases $\phi_3$ (a-c) and $\phi_4$ (d-f) computed for traffic shaping heuristics against simulation.

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 a large amount of data transfer between cores. Because XDense relies on nonprioritized 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 multihop point-to-point networks, addressing the issue of guaranteeing the delivery of messages with time constraints. This problem has been extensively researched since then. Fidler [10] surveys 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, Fan et al. [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 can provide better results compared to network calculus in some cases.

Traffic shaping to achieve tighter and deterministic bounds has also been investigated. For example, Sivaraman et al. [35] 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 in the work of Manolache et al. [22] for worst-case response guarantees and buffer space optimization. In the work of Specht and Samii [36], traffic shaping is also used on Ethernet networks to provide real-time guarantees by shaping the packet sources. The authors provide an asynchronous traffic scheduling algorithm, which gives low delay guarantees while keeping low implementation complexity.




(a) $\phi_3$ (b) $\phi_3$ (c) $\phi_3$

(d) $\phi_4$ (e) $\phi_4$ (f) $\phi_4$

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

## 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 article, 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 cluster size, and other configurations that may impact 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 that the proposed framework can be used in other kinds of synchronous multihop 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 shape each of the output/input flows on the network and provide real-time guarantees based on the calculations. We believe 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 CFD data to analyze the performance of the model versus synthetic data. The model can also be improved with more accurate portrayal of hardware (e.g., 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].

## 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 *Proceedings of the 2nd AIAA Flow Control Conference*. 2623.

[3] Richard Barry. 2016. Mastering the FreeRTOS real time kernel. *Real Time Engineers Ltd.* https://freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf.

[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. DOI: 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 and Technology (UIST'15)*. ACM, New York, NY, 649–658. DOI: 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, 43–50. DOI: 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. DOI: 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 (2010), 59–86. DOI: 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, 233–239. DOI: 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. DOI: 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. In *Proceedings of the 4th PROGRESS Symposium on Embedded Systems*. 129–140.

[15] F. Kopsaftopoulos, R. Nardari, Y.-H. Li, and F. K. 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*.

[16] 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 *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*. IEEE, Los Alamitos, CA, 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, Los Alamitos, CA, 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 *Proceedings of the 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'17)*. 1–10. DOI: https://doi.org/10.1109/RTCSA.2017.8046307

[19] João Loureiro, Raghuraman Rangarajan, and Eduardo Tovar. 2015. Demo abstract: Towards the development of XDense, a sensor network for dense sensing. In *Proceedings of the 12th European Conference on Wireless Sensor Networks (EWSN'15)*. 23. http://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 *Proceedings of the 2015 IEEE 3rd International Conference on Cyberr-Physical Systems, Networks, and Application (CPSNA'15)*. 54–59. DOI: 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, 110–117. DOI: 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 (DATE'06)*. 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 4th International Conference on Tangible, Embedded, and Embodied Interaction (TEI'10)*. ACM, New York, NY, 65–72. DOI: https://doi.org/10.1145/1709886.1709899

[24] F. Palacios, J. Alonso, K. Duraisamy, M. Colonno, J. Hicken, A. Aranake, A. Campos, et al. 2013. Stanford University unstructured (SU 2): An open-source integrated computational environment for multi-physics simulation and design. In *Proceedings of the 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. DOI: 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 Transactions on Computers* 62, 3 (March 2013), 452–467. DOI: https://doi.org/10.1109/TC.2011.240

[28] J. Reneaux2004. 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 Felix, 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. DOI: 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), AR–138.

[33] J. H. 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 2nd ACM/IEEE International Symposium on Networks-on-Chip (NOCS'08)*. IEEE, Los Alamitos, CA, 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, 4 (2006), 509–519. DOI: 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 *Proceedings of the 2016 28th Euromicro Conference on Real-Time Systems (ECRTS'16)*. 75–85. DOI: https://doi.org/10.1109/ECRTS.2016.27

[37] Troy M. Lau, Joseph T. Gwin, and Daniel P. Ferris. 2012. How many electrodes are really needed for EEG-based mobile brain imaging? *Journal of Behavioral and Brain Science* 2, 3 (2012), 387.

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

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

Received September 2017; revised April 2018; accepted June 2018