Real-Time Dense Wired Sensor Network Based on Traffic Shaping

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

*CISTER Research Centre

CISTER-TR-170602

2017/08/16

Real-Time Dense Wired Sensor Network Based on Traffic Shaping

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

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

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

http://www.cister.isep.ipp.pt

Abstract

XDense is a novel wired 2D-mesh grid sensor network system for application scenarios that benefit from denselydeployed sensing (e.g. thousands of sensors per square meter). Itwas conceived for closed-loop cyber-physical systems (CPS) thatrequire real-time actuation, like active flow control (AFC) onaircraft wing surfaces. XDense communication and distributedprocessing capabilities are designed such that they enable toextract complex features within bounded time and in a responsivemanner. In this paper we tackle the issue of deterministic behaviorof XDense. We present a methodology that uses traffic shapingheuristics to guarantee bounded communication delays and thefulfillment of memory requirements. We evaluate the model forvaried network configurations and workload, and demonstratethe effectiveness of running real-time applications supported onXDense.

Real-Time Dense Wired Sensor Network Based on Traffic Shaping

João Loureiro[†], Raghuraman Rangarajan[†], Borislav Nikolic[†‡], Leandro Indrusiak[§], Eduardo Tovar[†]
[†]CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
{joflo, raghu, borni, emt}@isep.ipp.pt
[‡]Institute of Computer and Network Engineering, Technische Universität Braunschweig, Germany
nikolic@ida.ing.tu-bs.de
[§]University of York, UK
lsi@cs.york.ac.uk

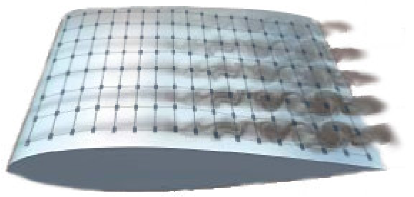
***Abstract*—XDense is a novel wired 2D-mesh grid sensor network system for application scenarios that benefit from densely deployed sensing (e.g. thousands of sensors per square meter). It was conceived for closed-loop cyber-physical systems (CPS) that require real-time actuation, like active flow control (AFC) on aircraft wing surfaces. XDense communication and distributed processing capabilities are designed such that they enable to extract complex features within bounded time and in a responsive manner. In this paper we tackle the issue of deterministic behavior of XDense. We present a methodology that uses traffic shaping heuristics to guarantee bounded communication delays and the fulfillment of memory requirements. We evaluate the model for varied network configurations and workload, and demonstrate the effectiveness of running real-time applications supported on XDense.**

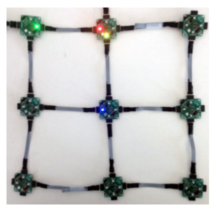
I. Introduction

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

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

To further motivate our approach, let us consider an aerospace application scenario that may benefit from such dense CPS. The drastic increase in demand for air transportation, naturally motivates measures to reduce its environmental impact. The reduction of fuel consumption is obviously important regarding both environmental effects and cost efficiency. It is known from the Breguet range equation [1] 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, what constitutes approximately one half of the total drag for a typical long range aircraft at cruise conditions [2].





(a)

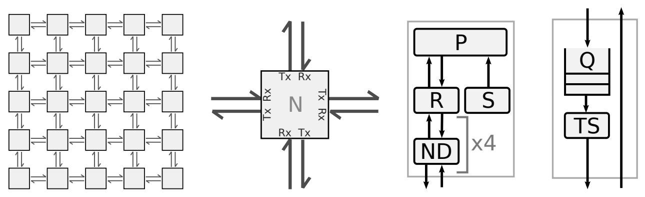
Fig. 1: (a) Conceptual deployment of XDense for active flow control (AFC). (b) 3 × 3 XDense network prototype using COTS.

A significant part of this skin friction is due to turbulent[1] airflow over the wing [4]. Turbulence can be highly undesirable, as it increases drag and noise. Additionally, it causes loss of energy [5], and an important goal is to minimize this loss. Figure 1 exhibits an example in which homogeneous laminar airflow transits to turbulent along the wing.

Several solutions have been proposed already to reduce turbulence. Cattafesta et al. [6] have surveyed the state-of-the-art actuation mechanisms used to reduce turbulent skin friction. The weakness of most approaches is in not relying on sensors to detect and trace turbulent flows, and hence offering 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 AFC implies that physical quantities are tracked through sensors (for example, pressure, temperature and vibration sensors), which are deployed with some high density (eventually a few centimeters apart). Figure 1 shows an envisioned deployment of such sensing/detecting infrastructure on a wing surface to detect the occurrence of turbulent airflows.

[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 [3].



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

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

XDense was developed to deal with the key challenges related to eXtremely Dense deployments of sensors [7]. XDense has a network architecture composed of regular structures (nodes) interconnected in a 2D-mesh network (see Figure 2(a)). This resembles common Network-on-Chip (NoC) architectures [8] and there are also similarities in routing schemes and distributed computing capabilities [9]. XDense exploits low-cost local communication and distributed processing strategies to enable distributed feature detection/extraction. For example, in [7], targeting AFC, we proposed algorithms to enable distributed turbulence detection in airflow computational fluid dynamics (CFD) data.

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. Timeliness is important 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 network, to avoid overload and consequent data loss. These are factors that have great influence on hardware requirements, cost and consequently on the applicability XDense. These two properties are therefore the focus of this paper.

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

The remainder of this paper is organized as follows. Section II introduces the basics of the XDense architecture; Section III formalizes our XDense model for real-time applications; Section IV evaluates the model; Section V discusses the related works and Section VI concludes the paper along with comments on potential future research directions.

II. XDense Architecture and Principles of Operation

A. Architecture and topology of XDense

XDense is a 2-D mesh network whose topology and node architecture are inspired from traditional Network-on-Chip (NoC) designs. Despite its similarities with NoCs, XDense differs in its size and node count. XDense is meant to be deployed on large surfaces (like aircraft wings) and deliver high-precision and very localized 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 [10].

The processor runs the application layer. It interfaces with the sensor and implements high level application-specific protocols for data sharing and processing. The router arbitrates the exchange of data. It can receive and transmit packets in parallel, from/to the processor and networking devices. Networking devices are full-duplex serial communication ports[2]. Each one has a queue (Q) and a traffic shaper (TS) (see Figure 2(d)). Input packets are directly delivered to the router whereas output packets are first queued (in FIFO order) at the target output port before they are dequeued by the traffic shaper to be then transmitted serially over the network. All network transfers are non-preemptive and packet-switched, and all packets have a fixed and equal size.

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

For realizing the above, a custom design integrated circuit (IC) obviously provides the best-fit solution. But, this reduces design flexibility and might become a single application solution. For this reason, we use a microcontroller and other COTS to prototype the XDense node and network, as shown in Figure 1(b). to a limited number of candidate microcontrollers, especially concerning the minimum number of Nodes have five high-speed serial ports, each one equipped with dedicated Direct Memory Access (DMA) channel. More details on developed node hardware prototype using COTS are available in [11].

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

[2]We use serial links as they are widely available in COTS micro-controllers and provide low complexity and low footprint at low cost (compared to parallel links found in NoC[8]). These are pragmatic decisions regarding the envisioned deployment scale.

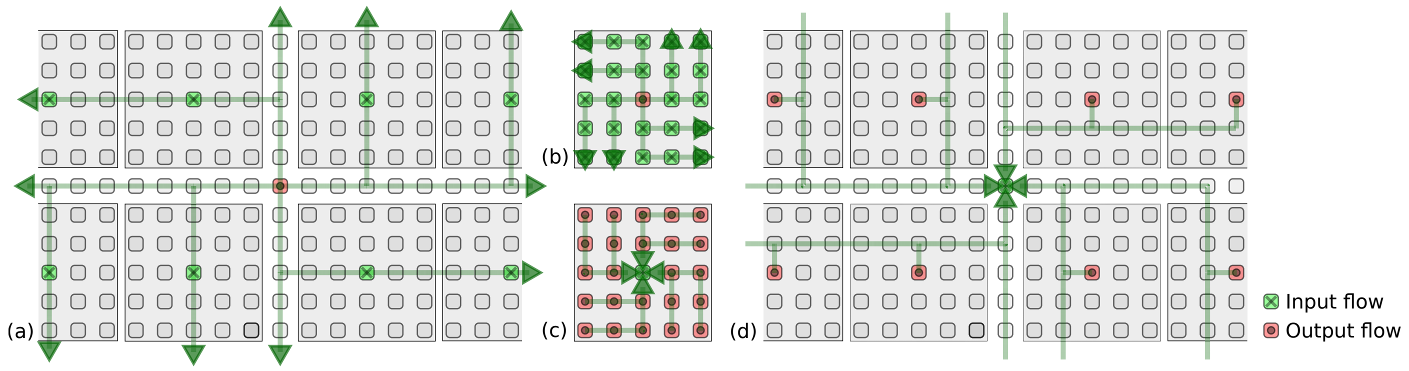


Fig. 3: Example 45×45 network, with a single central sink. In this case, with $n_{\text{radius}} = 2$. Application phases: (a) ϕ_1 – Sink requests data from cluster-heads; (b) ϕ_2 – Cluster heads in turn send a multicast request to nodes in their cluster; (c) ϕ_3 – Nodes send sensor data back to their respective cluster-head; (d) ϕ_4 – Cluster heads process received data and send result to sink.

B. *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 example of utilization is to program the cluster heads to inform the sink *only* upon the occurrence of meaningful events (e.g., airflow changes from laminar to turbulent and conversely). The routing protocols elected should ideally exploit the network topology to avoid congestion. It is also required to define application protocols to allow coordination of clusters by the sink.

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

1) Clustering nodes: The reason for grouping the nodes into clusters is obviously to reduce the load on the network by performing in-cluster data pre-processing at the selected cluster heads. Our tested solution implements non-overlapping "square" clusters – the network topology being a 2D grid of X times Y nodes, all clusters are non-overlapping and of size $n_{\text{size}} \times n_{\text{size}}$, with $n_{\text{size}} \leq X$ and $n_{\text{size}} \leq Y$. n_{size} must be a positive odd number and the cluster head is the node located at the "center" of the square. The cluster size n_{size} is defined through the system parameter n_{radius} that denotes the maximum distance from the cluster head to the farthest node in the cluster (considering rectilinear distance, a.k.a. Manhattan distance). Figure 3 shows a scenario with $n_{\text{radius}} = 2$.

The purpose of local in-cluster processing is to extract high level aerodynamic information of the airflow, which is transmitted in a smaller number of packets (when compared to the number of packets required to transmit the raw data). The pre-processing and compression algorithms to be used are application-specific and are not in the scope of this paper. We have though discussed application specific issues in previous works (see [7] for a discussion on this topic).

2) Executing application in phases: The execution of the application is logically divided in to a set of four consecutive phases ϕ_1, ϕ_2, ϕ_3 and ϕ_4. The first phase starts when the sink requests data from the cluster heads. Every successive phase starts when the previous one ends. Specifically, the four phases are:

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

Note that the clusters may *not* always be in sync with respect to the phase of their execution. The second phase (ϕ_2) for instance, may start in each cluster with a different time offset (this offset being proportional to the distance between the cluster head of each cluster and the sink). Also, note that the sink and cluster heads serve as regular sensing node as well. The sink is the only node to act as the gateway with the outside world and has a backhaul link (for example, a wireless link). Figures 3(a) to 3(d) show the four phases in a chronological order.

3) Spatial isolation through routing schemes: The four phases described above require spatial isolation so that packets do not compete with each other for network resources when traversing it. We use the well-known dimension-order routing algorithms known as X-Y and Y-X routing protocols [8]. In X-Y routing (resp, Y-X), packets are first routed along the X (resp, Y) dimension and then along the Y (resp, X) dimension. These protocols always find the shortest path between the source and destination nodes (again, in terms of the Manhattan distance) and are proven to be deadlock-free [12].

Phases ϕ_1 to ϕ_3 use one of the following two routing algorithms, sometimes called Counterclockwise Dimension Routing (see Figures 3(a)-(c)). The starting dimension (X or Y) depends on the quadrant in which the destination node is, relatively to the origin of the packet. For phase ϕ_4 we propose another routing protocol hereafter referred to as Shifted Clockwise Dimension Routing. This protocol adds an initial change in dimension on the first hop and then uses a regular clockwise routing (see Figure 3(d)).

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

III. 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.

A. Model of computation

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

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

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

Example 1 (3D accelerometer)**.** Consider a 3-axis acceleration sensor whose data has to be transmitted as three separate packets in a single flow (one packet for each acceleration axis). In this case, we want the data for the 3 axis to be transmitted together. Therefore, we set $\beta = 1$ with $\sigma = 3$ for that flow.

Example 2 (Pressure sampling)**.** Consider a use-case in which ten samples of pressure data need to be transmitted, using one packet per sample. We are interested in having periodic sampling, equally distributed in time. By setting the burstiness to $\frac{1}{20}$ for instance, one packet will be sent every 20 TTS. Therefore, for that flow we set $\beta = \frac{1}{20}$ and $\sigma = 10$.

B. Shaping flows and traffic throughout the network

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

Although the flows are shaped at their source node, when multiple flows (say, $f_1^{\text{in}}, f_2^{\text{in}}, \ldots, f_k^{\text{in}}$) traverse the network at the same time, pass through the same router, and compete for the same output port, the resulting output flow f^{out} at that port is a superposition of all these competing flows. As such, f^{out} has an irregular packet-release pattern and a rate that can no longer be modeled using the three parameters (O, σ, β). This can be seen in the example illustrated in Figure 4(b), in which four flows $f_1^{\text{in}}, f_2^{\text{in}}, f_3^{\text{in}}$, and f_4^{in} interfere with each other when competing for a same output port. Each of these input flows f_k^{in} starts at time O_k and has a duration defined as $\ell_k = \frac{\sigma_k}{\beta_k}$. That is, flow f_k sends all its packets during these ℓ_k TTS. As seen in the upper part of the graphic (Figure 4(a)), the number of packets to be sent over time from the output port depends on the starting time and duration of all the competing input flows, and the resulting curve can no longer be modeled using the simple linear model (O, σ, β). It is thus difficult to estimate the delay induced at every output port because of the competing flows and hence to provide timing guarantees on the end-to-end communication time between all the nodes.

To make the network amenable to timing analysis, we shape the traffic *at every output port* of every node and make it fit the linear model (O, σ, β). That is, at every output port of every node in the network, we first identify the set of input flows f_k^{in} (with $k = 1, 2, \ldots$) and based on the respective parameters (O_k, σ_k, β_k) of these flows, we compute the parameters $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ that are used to shape the resulting output flow at that output port. Note that it has been proven in [13] that to calculate optimal shaping parameters in a multihop scenario can be computationally intractable, and thus finding a solution at runtime is not feasible.

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

Assumption 1. In every node, all the packets entering by a given input port are assumed to exit through a single output port.

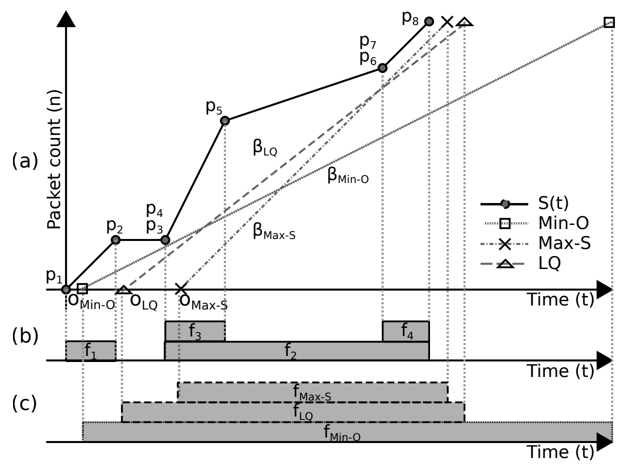


Fig. 4: Traffic shaping heuristics: (a) input, and output flows using the proposed heuristics; time-line showing offset and duration of (b) arriving flows and (c) departure flows.

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

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

C. Shaping output traffic at a single output port

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

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

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

Let $S(t) = \sum_{f_k^{\text{in}} \in F^{\text{in}}} S_k^{\text{in}}(t)$ be the sum of the functions $S_k^{\text{in}}(t)$ of all the input flows f_k^{in}. This function $S(t)$ is depicted in Figure 4(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 4. With these new notations, we can introduce our three heuristics for the computation of the parameters $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ of the shaper used at the analyzed output port.

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

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

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

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

*1) Minimum offset (*Min-O*):* This first heuristic aims at avoiding bursty traffic while coping as much as possible with the bandwidth demand of the input flows. This traffic shaper forwards the first packet as soon as it can, i.e. one TTS after the packet has arrived, at time $O^{\text{out}} = t_1 + 1$ TTS, and forwards all the subsequent packets at the highest admissible rate; that is, with the highest burstiness β^{out} such that the number of packets sent at any time $t \ge O^{\text{out}}$ never exceeds $S(t)$. This burstiness corresponds to the highest slope among the slopes of all the lines passing through the point $(t_1+1, 0)$ such that, for every $t_j \in \mathcal{T}$, the point of x-coordinate t_j in the line has an y-coordinate $\le S(t)$ – In simple terms, the line is "below" the function $S(t)$, $\forall t \ge 0$. 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 4(a) shows Min-O departure curve with β^{out} as $\beta_{\text{Min-O}}$.

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

$$\beta^{\text{out}} = \max_{t_j \in \mathcal{T}} \left(\frac{S(t_m) - S(t_j)}{t_m - t_j} \right) \tag{2}$$

Figure 4(a) shows Max-S departure curve with β^{out} as $\beta_{\text{Max-S}}$. The offset O^{out} in Max-S is simply set to the X-intercept of the line of slope β^{out} and passing through the anchor point $(t_m, S(t_m))$ to which we add 1 TTS, to make sure that packets are not forwarded before the first packet arrives (like we did in Min-O), i.e.

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

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

Figure 4(a) shows the departure line of Max-S, initially calculated with a slope > 1 as a result of Equation 2. That slope is then adjusted to $\beta^{\text{out}} = 1$ as depicted on that Figure. As seen, after adjusting its slope, the line corresponding to the parameters of the Max-S traffic shaper does not intersect with the function $S(t)$ – It seems to be "too much shifted to the right". An easy patch to reduce this gap between $S(t)$ and the shaper is to set its offset to the *minimum* offset such that the line remains below all the points of $S(t)$. That is,

$$O^{\text{out}} = \min_{t \geq 0} \left\{ t \text{ such that } \beta^{\text{out}} \leq \min_{\substack{t_j \in \mathcal{T} \\ t_j > t}} \left(\frac{S(t) - S(t_j)}{t - t_j} \right) \right\} \tag{3}$$

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

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

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

where

$$\begin{aligned} \bar{t} &= \frac{1}{m} \sum_{t_j \in \mathcal{T}} t_j \\ \bar{S} &= \frac{1}{m} \sum_{t_j \in \mathcal{T}} S(t_j) \end{aligned}$$

and r is the correlation coefficient computed as

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

Once we have computed the slope, we choose the smallest offset O^{out} such that the line of slope β^{out} and passing through $(O^{\text{out}}, 0)$ is never above any point $(t, S(t))$, $\forall t \geq 0$. This is done using Equation 3.

D. Worst-case per-hop delays and maximum queue sizes

Having stated the heuristics, we can now apply them to all the phases of the application. We perform this in a hop-by-hop strategy, starting from the output ports of the nodes for which the parameters $(O_k^{\text{in}}, \sigma_k^{\text{in}}, \beta_k^{\text{in}})$ of all the interfering flows f_k^{in} are known. For each such output port, the resulting output flow f^{out} is shaped using the same model $(O^{\text{out}}, \sigma^{\text{out}}, \beta^{\text{out}})$ that is then propagated as the input flow in the next hop. The process continues until the parameters of the shaper of every output port of all the nodes of the network are defined (the output ports that no flows ever traverse and that are thus unused are naturally ignored). As mentioned earlier, we assume that there are no cyclic dependencies between the flows at any output port, which implies that the process eventually terminates.

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

$$L^{\text{out}}(t) = \beta^{\text{out}} t - \beta^{\text{out}} O^{\text{out}} \tag{4}$$

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 $\mathrm{dv}_j^{\mathrm{out}}$ between the point $(t, S(t))$ and the point $(t, L^{\mathrm{out}}(t))$ on the line L^{out}. This vertical distance is simply equal to:

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

and thus the maximum number $\mathrm{MaxQueue}$ of packets buffered at that output port is given by:

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

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

$$\mathrm{MaxQueue} = \max_{t_j \in \mathcal{T}} \left(S(t_j) - L^{\mathrm{out}}(t_j) \right) \tag{5}$$

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

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

Similarly, the transmission delay at any time-instant t is given by the *horizontal* distance $\mathrm{dh}_j^{\mathrm{out}}$ between the point $(t, S(t))$ and the point of y-coordinate $S(t)$ on the line L^{out}. According to Equation 4, that point of y-coordinate $S(t) \in L^{\mathrm{out}}$ has an x-coordinate x such that $S(t) = \beta^{\mathrm{out}} x - \beta^{\mathrm{out}} O^{\mathrm{out}}$ and thus $x = \frac{S(t)}{\beta^{\mathrm{out}}} + O^{\mathrm{out}}$. The horizontal distance is then simply given by:

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

and thus the maximum delay $\mathrm{MaxDelay}$ at that output port is:

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

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

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

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

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

$$\mathrm{ExecTime} = \max_{\mathrm{card} \in [\uparrow, \downarrow, \rightarrow, \leftarrow]} \left(O^{\mathrm{in}} + \frac{\sigma^{\mathrm{in}}}{\beta^{\mathrm{in}}} \right) \tag{6}$$

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

IV. Evaluation of traffic shaping heuristics

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

Network setup: The network is organized as a square grid of $45 \times 45 = 2025$ nodes with an unique sink located at the center of the grid. Figure 3 depicts a closeup on the sink. In that figure we can also see 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, the overall cluster organization, and the routes taken by the flows in the different phases. Based on an integer parameter n_{radius} that we vary in our experiments, we define every cluster as a square grid of $(2n_{\mathrm{radius}} + 1)^2$ nodes with the cluster head at the center of the grid. As such, n_{radius} defines both the cluster size and the number of clusters (the smaller the clusters, the more clusters in the network, and reversely). Because of our routing algorithms and network symmetry assumptions (position of sink in the center of the network and cluster-head in the center of their cluster), the workload observed in each quadrant around the sink or cluster-heads will be identical. This makes it sufficient to analyze a single quadrant of the network or cluster.

Shaping heuristics: We evaluate the performance of the three proposed heuristics Min-O, Max-S, and LQ against the performance of a cycle-accurate network simulator that we call BE.[3] The simulator does not implement any traffic shaper and thus it delivers the best effort (BE) performance overall. However, the congestion scenarios at the bottlenecks of the network (mostly at the sink and cluster heads) are so complex that they are hardly analyzable and hinder the development of an analytical framework to provide real-time guarantees.

Evaluation criteria and methodology: For each of the three heuristics Min-O, Max-S, LQ, we evaluate the maximum queue sizes and the end-to-end execution time of the phases ϕ_3 and ϕ_4. For BE, maximum queue sizes and the end-to-end execution time are measured in the simulator. We do so for different cluster sizes and flow burstiness. Specifically, we compute this metrics for in a network with 45×45 nodes divided in clusters with size n_{radius} set to $1, 2, 3, 4$ and 5. For each of these cluster sizes, we analyze the variation of this metrics when we vary the burstiness β of all the flows at their source node from 0 to 1 by step of 0.02.

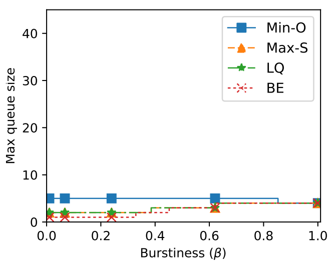
The message size σ differs for each phase. For phases ϕ_1 and ϕ_2, a single packet is generated at the sink and cluster head, whereas $\sigma = 1$. At phase ϕ_3, each sensing node outputs a flow with message size $\sigma = 4$. At the end of ϕ_3, the cluster head receives in total four packets per each node on its cluster plus 4 of its own sensed data. In turn, each cluster head outputs a flow with message size σ, as the sum of all these packets times $\lceil 1 - \text{CR} \rceil$. The term CR aims at reproducing the effect of data aggregation by the cluster head. For this work we define this as a fixed value equal to $\text{CR} = 80\%$, which was shown in previous work [7] to be a reasonable ratio, depending on the data of interest.

A. *Maximum queue size*

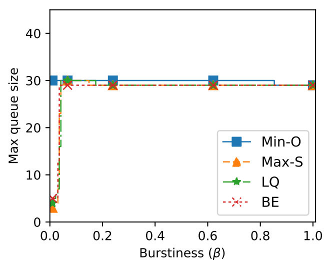
For each of the three heuristics Min-O, Max-S and LQ, we first derive the parameters (O, σ, β) of all the traffic shapers in the network. Then we use Equation 5 on every shaper to compute the maximum queue size of the corresponding node and finally, we retain the maximum queue size of all the nodes in the network. As we can see in Figure 5(a) and (b), in phase ϕ_3 the queues are smaller for smaller clusters (n_{radius}). This is expected since smaller clusters contain less nodes and therefore there are less packets exchanged within each cluster, and thus less congestion.

The opposite scenario would be expected for phase ϕ_4 since using smaller clusters means more clusters in the network, and thus more cluster heads transmitting packets to the sink. Yet, this is not observed in Figure 5(c) and (d). That is, smaller clusters *do not imply longer queues*. The reason for this counter-intuitive result can be unveiled by looking at the *utilization* of the four input links of the sink. We define the utilization of a link as the number of packets sent to that link in a given phase (here, phase ϕ_4) divided by the time (number of TTS) it takes for all those packets to traverse it. An utilization of 1 means that the link is never idle during the considered phase whereas an utilization of 0 means that the link is not used. As seen in Figure 6(c) and (d), smaller clusters yield a better utilization of the input links of the sink. This is because the number of packets sent to the sink does not depend on the cluster size and with more (and smaller) clusters, the input links of the sink 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

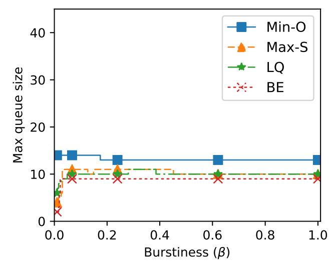
[3]More details on the implementation of our simulator can be found in [24]



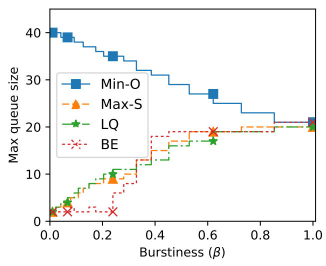
(a) $\phi_3, n_{\text{radius}} = 1$



(b) $\phi_3, n_{\text{radius}} = 5$

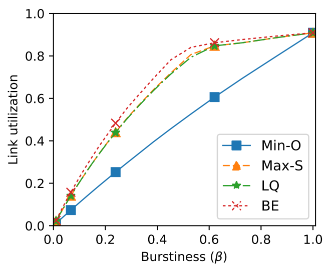


(c) $\phi_4, n_{\text{radius}} = 1$

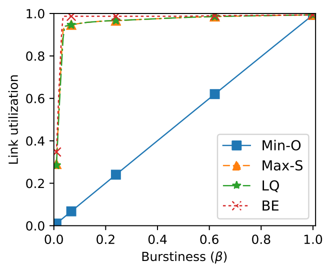


(d) $\phi_4, n_{\text{radius}} = 5$

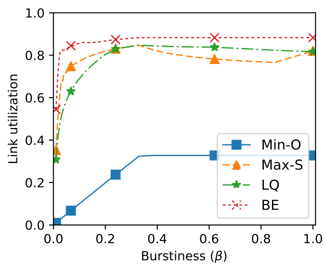
Fig. 5: Maximum queue size computed for our three traffic shaping heuristics against the maximum queue size measured during simulation (we display here only the results for the phases ϕ_3 and ϕ_4 and for n_{radius} set to 1 and 5).



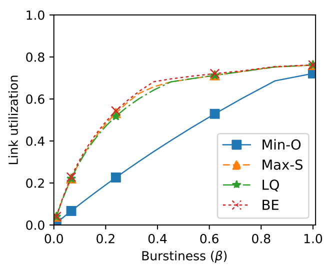
(a) $\phi_3, n_{\text{radius}} = 1$



(b) $\phi_3, n_{\text{radius}} = 5$



(c) $\phi_4, n_{\text{radius}} = 1$



(d) $\phi_4, n_{\text{radius}} = 5$

Fig. 6: Link utilization computed for our three traffic shaping heuristics against the Link utilization measured during simulation (we display here only the results for the phases ϕ_3 and ϕ_4 and for n_{radius} set to 1 and 5).

traverse the intermediate hops. Greater utilization for a same number of packets means that there is less congestion in the network and thus smaller queue at the individual hops.

It is worth noticing that in some scenarios, the maximum queue size obtained when using traffic shaping is smaller than the maximum queue size without traffic shaping. This is evidenced in Figure 5(d), for $\beta \in [0.4, 0.6]$, where the maximum queue size of $\mathrm{Max\text{-}S}$ and LQ is smaller than that of BE. This result is due to the offset O of the traffic shapers in the initial hops. The offsets have for effect to distribute in time the load on the network and thus decrease the overall congestion.

Another interesting observation is for the method $\mathrm{Min\text{-}O}$, the maximum queue size gets smaller for increased burstiness. This is very counter-intuitive since we would expect that by injecting more traffic in the network, the congestion would increase. However, this phenomena can be easily explained mathematically: it is due to the way the method $\mathrm{Min\text{-}O}$ is defined. Looking at Figure 4, the flows duration defined as $\frac{\sigma}{\beta}$ are longer for lower burstiness β and thus for low values of β, the first points $\in \mathcal{T}$ (depicted by p_1, p_2, etc.) are farther from the origin. Since $\mathrm{Min\text{-}O}$ selects a point close to the origin as "anchor" point, its slope must be small so that the line remains below the function $S(t)$. With a low slope, it is likely that the vertical distance between the function $S(t)$ and the line will be high (in particular if $S(t)$ increases quickly). These phenomena can already be observed, to a limited extent, in Figure 4.

To conclude, the heuristics $\mathrm{Max\text{-}S}$ and LQ have an average performance close to that of BE that does not use traffic shaping. This means that our heuristics are able to provide accurate timing estimates and yet suffer very little loss of performance as compared to a best-effort solution.

B. *Phase execution time*

We compare the execution time of the phases ϕ_3 and ϕ_4 in Figure 7, again for the two cluster sizes defined by $n_{\mathrm{radius}} = 1$ and 5 and varying the burstiness of the initial flows from 0 to 1 by step of 0.03. The execution times are computed by using Equation 6. As seen in all graphics of Figure 7, increasing the burstiness considerably reduces the execution time of the phases (note that the plots are in logarithmic scale) up to a point it remains constant. This point is reached only for high burstiness in Figure 7(a) whereas it is reached almost immediately in Figure 7(b). This threshold beyond which the execution time cannot be further reduced can be explained by looking at the utilization of input link of cluster heads (for phase ϕ_3) and the sink (for phase ϕ_4). Those thresholds correspond to specific values of the burstiness for which the links saturate and therefore, any further increase in burstiness only results in longer queues but not in reduced execution time.

V. Related work

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

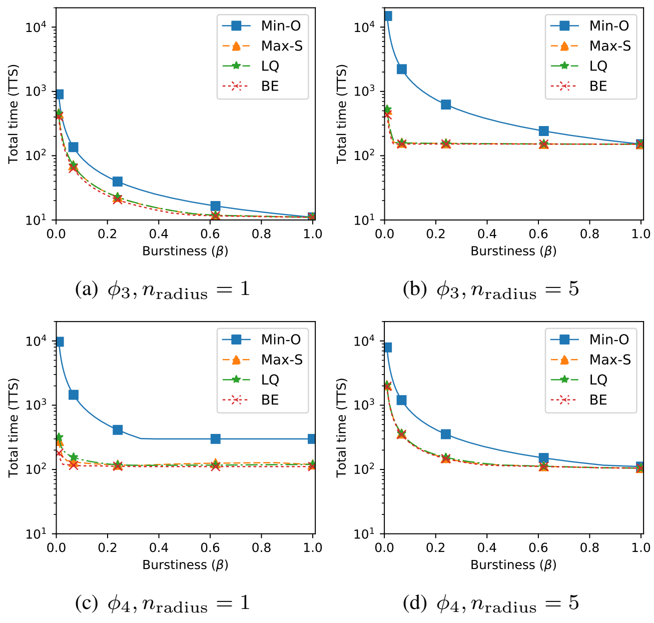


Fig. 7: Execution time of phases ϕ_3 and ϕ_4 computed for our three traffic shaping heuristics against their execution times recorded during simulation.

A multi-modal sensor network was proposed in [14], as a scalable sensor network with up to hundreds of nodes per square meter. However, due to the wireless nature of the links (infrared), contentions and collisions substantially increase the cost of communication. Their research leans more towards wireless sensor networks whose performance does not suit the application scenarios we focus on. Instead of wireless links, in [15], the authors use wired links to deploy few sensors in a grid network, to act as an electronic skin. Nodes are interconnected using shared buses, approach that differs from ours.

More recently, the authors of [16] presented a modular and dense sensor network in a form factor of a tape, tailored to wearables. In these cases though, master-slave buses are used to interconnect nodes (through SPI or I2C), regardless of the shape of the network. Not only shared buses drastically decrease the opportunity for distributed processing due to their finite bandwidth that do not scale along with the number of nodes, but they also constraint the number of nodes due to limited address space and related electrical limitations. They are therefore not a scalable solution.

XDense uses a 2D mesh network architecture that resembles common NoC architectures [8], [17]. Numerous works to achieve real-time guarantees have been proposed for NoCs. For instance, the authors of [18] proposed a worst-case analysis technique for priority-preemptive, wormhole switched NoCs. This approach was later extended in [19], in which an end-to-end schedulability analysis was proposed for many-core systems. However, wormholing is tailored to parallel links, where very high bandwidth is required, and for large amount of data transfer between cores. Because XDense relies on non-prioritized packet switched serial communication, this approach does not apply to our network architecture.

More in line with our approach, real-time communication for multi-hop networks addresses the issue of guaranteeing the delivery of messages with time constraints, in networks with multiple nodes and point-to-point interconnection. This solution is based on a linear bounded arrival process, initially proposed by Cruz [20]. Later in [21], the authors propose a channel establishment scheme for computing worst-case response times. This problem has also been addressed by Cidon et al. [22] where FIFO scheduling is used to obtain an upper-bound on the network delay in each node. They also limit buffer sizes, aiming at maximizing throughput while keeping a low probability of packet loss.

Traffic shaping to achieve tighter and deterministic bounds has also been investigated already. For example, in [13] the authors use rate controlled Earliest Deadline First scheduling in conjunction with per-hop traffic shaping to provide deterministic end-to-end delay guarantees. They identify the shaping parameters that result in maximal network utilization. This has also been studied for NoCs in [23] for worst-case response guarantees and buffer space optimization.

VI. Conclusions and future work

The proposed traffic shaping heuristics enable us to endow XDense networks with real-time capabilities. We showed that on average, the performance of XDense is very similar with and without traffic shaping. This means that the proposed traffic shaping techniques allow for accurate timing and memory requirement estimates to be derived while imposing minor performance overheads.

Although it has not been discussed in the paper, the analysis framework that we propose can also serve as a basis to reason on the dimensioning of the system (choosing the network size, the cluster configuration, etc.).

Improvements to the model can be made along many dimensions. One would be to incorporate applications with heterogeneous flows sources, in which case our heuristics may perform differently. Another would be to bring in computational fluid dynamics data to analyze the performance of the model vs synthetic data. The model can also be improved with more accurate portrayal of hardware (for example, 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 [24].

Acknowledgments

This work was supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the Portugal2020 Program, within the CISTER Research Unit (CEC/04234).

References

[1] S. Anders, W. Sellers, and A. Washburn, "Active flow control activities at nasa langley," in *2nd AIAA Flow Control Conference*, 2004, p. 2623.

[2] T. Washburn, "Airframe drag/weight reduction technologies," *Green Aviation Summit-Fuel Burn Reduction, NASA Ames Research Centre*, 2010.

[3] S. K. Robinson, "Coherent motions in the turbulent boundary layer," *Annual Review of Fluid Mechanics*, vol. 23, no. 1, pp. 601–639, 1991.

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

[5] W. K. Blake, *Mechanics of Flow-Induced Sound and Vibration V2: Complex Flow-Structure Interactions*. Elsevier, 2012, vol. 2.

[6] L. N. Cattafesta and M. Sheplak, "Actuators for active flow control," *Annual Review of Fluid Mechanics*, vol. 43, pp. 247–272, 2011.

[7] J. Loureiro, R. Rangarajan, and E. Tovar, "Distributed sensing of fluid dynamic phenomena with the xdense sensor grid network," in *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2015 IEEE 3rd International Conference on*, Aug 2015, pp. 54–59.

[8] N. K. Kavaldjiev and G. J. M. Smit, "A survey of efficient on-chip communications for soc," in *4th PROGRESS Symposium on Embedded Systems, Nieuwegein, The Netherlands*. Utrecht, The Netherlands: Technology Foundation STW, October 2003, pp. 129–140.

[9] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*. IEEE, 2002, pp. 105–112.

[10] N. Kasagi, Y. Suzuki, and K. Fukagata, "Microelectromechanical systems-based feedback control of turbulence for skin friction reduction," *Annual review of fluid mechanics*, vol. 41, pp. 231–251, 2009.

[11] J. Loureiro, R. Rangarajan, and E. Tovar, "Demo abstract: Towards the development of xdense, a sensor network for dense sensing," *12th European Conference on Wireless Sensor Networks (EWSN)*, p. 23.

[12] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based noc architectures under performance constraints," in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '03. New York, NY, USA: ACM, 2003, pp. 233–239.

[13] V. Sivaraman, F. M. Chiussi, and M. Gerla, "Deterministic end-to-end delay guarantees with rate controlled {EDF} scheduling," *Performance Evaluation*, vol. 63, no. 45, pp. 509 – 519, 2006.

[14] J. Lifton, M. Broxton, and J. A. Paradiso, "Experiences and directions in pushpin computing," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005.

[15] B. F. Mistree and J. A. Paradiso, "Chainmail: A configurable multimodal lining to enable sensate surfaces and interactive objects," in *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, ser. TEI '10. New York, NY, USA: ACM, 2010, pp. 65–72.

[16] A. Dementyev, H.-L. C. Kao, and J. A. Paradiso, "Sensortape: Modular and programmable 3d-aware dense sensor network on a tape," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, ser. UIST '15. New York, NY, USA: ACM, 2015, pp. 649–658.

[17] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (noc) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[18] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 161–170.

[19] L. S. Indrusiak, "End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration," *Journal of Systems Architecture*, vol. 60, no. 7, pp. 553 – 561, 2014.

[20] R. L. Cruz, "A calculus for network delay and a note on topologies of interconnection networks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1987.

[21] D. D. Kandhlur, K. G. Shin, and D. Ferrari, "Real-time communication in multihop networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, Oct 1994.

[22] I. Cidon and I. S. Gopal, "Paris: An approach to integrated high-speed private networks," *International Journal of Digital & Analog Cabled Systems*, vol. 1, no. 2, pp. 77–85, 1988.

[23] S. Manolache, P. Eles, and Z. Peng, "Buffer space optimisation with communication synthesis and traffic shaping for nocs," in *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, ser. DATE '06. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 718–723.

[24] J. Loureiro, P. Santos, R. Rangarajan, and E. Tovar, "Simulation module and tools for xdense sensor network," in *Proceedings of the Workshop on Ns-3*, ser. WNS3 '17. New York, NY, USA: ACM, 2017, pp. 110–117.