Distributed Load balancing for heterogeneous Fog Computing infrastructures in Smart Cities

Roberto Beraldi[a], Claudia Canali[b], Riccardo Lancellotti[b], Gabriele Proietti Mattia[a]

[a]*Department of Computer, Control and Management Engineering "Antonio Ruberti", Sapienza University of Rome*
[b]*Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia*

Abstract

Smart cities represent an archetypal example of infrastructures where the fog computing paradigm can express its potential: we have a large set of sensors deployed over a large geographic area where data should be pre-processed (e.g., to extract relevant information or to filter and aggregate data) before sending the result to a collector that may be a cloud data center, where relevant data are further processed and stored.

However, during its lifetime the infrastructure may change, e.g., due to the additional sensors or fog nodes deploy, while the load can grow, e.g., for additional services based on the collected data. Since nodes are typically deployed in multiple time stages, they may have different computation capacity due to technology improvements. In addition, an uneven distribution of the workload intensity can arise, e.g., due to hot spot for occasional public events or to rush hours and users' behavior. In simple words, resources and load can vary over time and space.

Under the resource management point of view, this scenario is clearly challenging. Due to the large scale and variable nature of the resources, classical centralized solutions should in fact be avoided, since they do not scale well and require to transfer all data from sensors to a central hub, distorting the very nature of in-situ data processing.

In this paper, we address the problem of resources management by proposing two distributed load balancing algorithms, tailored to deal with heterogeneity. We evaluate the performance of such algorithms using both a simplified environment where we perform several sensitivity analysis with respect to the factors responsible for the infrastructure heterogeneity and exploiting a realistic scenario of a smart city. Furthermore, in our study we combine theoretical models and simulation. Our experiments demonstrate the effectiveness of the algorithms under a wide range of heterogeneity, overall providing a remarkable improvement compared to the case of not cooperating nodes.

Keywords: Smart cities, Fog computing, Queuing model, Simulation

Email addresses: `beraldi@diag.uniroma1.it` (Roberto Beraldi), `claudia.canali@unimore.it` (Claudia Canali), `riccardo.lancellotti@unimore.it` (Riccardo Lancellotti), `proiettimattia@diag.uniroma1.it` (Gabriele Proietti Mattia)

1. Introduction

The fog computing paradigm shifts the deployment model of distributed applications from a centralized approach, where every component of the application runs in a cloud data center, towards a multi-layer architecture, where processing functions can be distributed along the path from the network edge to the centralized cloud core [1]. This innovative model aims to cope with the demand for low latency (in the order of 10ms) and high throughput (in the order of 10Gbps), combined with security and privacy demands [2]. A similar approach lies behind the terms Edge Computing, Multi-access Edge Computing (MEC), or Cloudlet, which differ for the physical front line where managed resources are placed.

The reference model for our paper is provided in Fig. 1, with fog nodes densely distributed in a given geographic area that provide a service of data processing for a plethora of sensors, e.g., IoT sensors or image processing in a smart city application, using a 5G Fog Radio Access Network (F-RAN) architecture [3]. See [4] and [5] for a more in-depth description of computer vision-based services and smart cities infrastructures. It is worth to note that, even if in this paper the reference scenario is focused on a smart city application, the main findings of our research have general validity and can be applied to other application fields with similar characteristics.

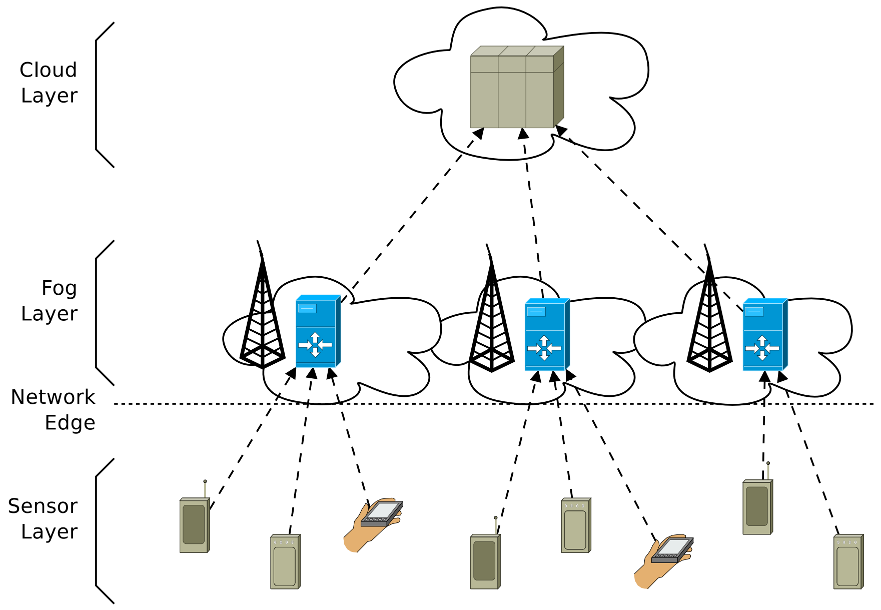


Figure 1: A typical fog computing deploy model.

To understand how important the role of fog computing infrastructures could get in the near future it is sufficient to consider the businesses opportunities created by the combination of dense populations and complex infrastructures: up to 80% of cities in the United States, indeed, are expected to adopt connected technologies within the year 2025 and create about \$1.5 trillion dollars in economic value[1].

[1]https://www.forbes.com/sites/blakemorgan/2019/11/01/top-80-stats-about-a-future-customer-

In the case of a smart city, the fog computing infrastructure has to support IoT-based applications that often present critical requirements in terms of low and predictable latency and high computational load. Typical examples of such applications are gaming and augmented reality, control of autonomous vehicles, vehicular traffic monitoring, environmental sensing (e.g., air quality control), and public surveillance [6, 7]. These applications usually involve high amounts of data (e.g., video frames) produced at the sensor layer that should be processed by the fog nodes involving computationally expensive tasks. Another application key requirement could be an almost real-time responsiveness to efficiently react to situation changes. Finally, we must consider that smart city infrastructures are seldom deployed in the space of a single project, but the deployment is typically divided into multiple projects over time. As a consequence, the resulting fog infrastructure is typically highly heterogeneous, with nodes characterized by highly different hardware with different communication and processing capabilities.

Among the several challenges introduced by the fog computing (see [8] for a general discussion), in order to address the above upcoming scenario, we focus on a specific research topic currently under investigation: the design of an algorithm for resource sharing among uncoordinated and heterogeneous fog nodes in order to improve the response time of smart cities applications. Although resource sharing is a classical and well-studied topic in the computer science community, this model of fog computing does not fit all the assumptions of the studies available in the literature. In particular, the following elements are new to the fog deployment: (i) the heterogeneity among the elements of the infrastructure; (ii) the execution time of a job that is comparable to the time required to transfer the job from the node of origin to another node; (iii) the absence of a centralized entity that acts as a load balance.

Our proposal consists of two load balancing algorithms, namely *sequential forwarding* and *adaptive forwarding*, designed to take these peculiarities into account. In particular, in this paper we place a major emphasis on the heterogeneity aspect of the problem, considering that fog nodes are characterized by different computing power and may receive different workload intensities. The workload consists of jobs that are continuously generated from end devices (on-line load balancing). The basic idea of the proposed algorithms is the following. We assume that the fog computing layer provides an elementary service to end-users, e.g., consisting in object detection inside a video frame [5]. As jobs reach the fog nodes, the nodes estimate the expected waiting time (based on the number of jobs already being processed by the node). If the waiting time exceeds a threshold Θ (that may change depending on the fog node and may be self-tuning), the fog node forwards *blindly* at random the job to another fog node, that executes the same decision algorithm. By this, we mean that the node doesn't keep or probe any information about the current state of the other nodes, but rather picks one of the nodes it is aware of, uniformly at random. Decisions are memory-less, except for the number of forwarding (steps) already done, which is carried in the message. The steps are upper bounded by a parameter M. At the M-th forwarding, the receiving fog node will process the job without further attempts, unless its processing

experience-shaped-by-technology/

queue is full, in which case the job will be dropped.

Any of the two algorithms fits the three challenges of fog computing because (i) it takes explicitly into account uneven load distributions and heterogeneous node characteristics and configurations parameters, e.g., lightly loaded nodes do not forward their jobs often (where the load is normalized to the actual node execution speed) (ii) it places a significant effort in limiting the number of (potentially expensive) transmissions between nodes by adapting Θ, (iii) it is completely distributed since forwarding decisions are local, uncorrelated and autonomous.

The contribution of the paper can be summarized as follows:

- definition of a lightweight randomized on-line distributed load balancing algorithm suitable for scenarios characterized by *independent* providers and *heterogeneous* load conditions, along with a variant based on a *self-tuning* mechanism;
- mathematical analysis and experimental evaluation of the algorithms on a realistic scenario, showing evidence of significant improvements compared to unbalanced nodes. Throughout our sensitivity analysis we found that the loss rate of the proposed algorithms is in most cases 15 to 19 times lower than the case where no cooperation is used. In a similar way, the response time is reduced by 19% to 11% in most scenarios. In the realistic setup our algorithms provide an even more impressive performance gain with a reduction in the loss rate from 13% to 0.2% and a response time nearly halved;
- through our tests we point out how a self-tuning mechanism can provide robust performance requiring limited tuning of the algorithms' parameters and we provide some insight on how the characteristics of a heterogeneous infrastructure impact on the algorithms' parameters.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 describes the proposed algorithms while Section 4 describes a mathematical model of the protocol and provides some numerical results on the performance of the algorithms. In Section 5 a simulation-based study is used to carry out sensitivity analyses of the proposed algorithms to the main scenario parameters and to evaluate the performance of our proposals in a realistic scenario of a smart city application deployment. Finally, some concluding remarks are provided in Section 6.

2. Related Work

The interest towards the management of fog computing infrastructures has been addressed in literature in multiple ways.

On one hand, a corpus of literature aims to address the problem of connecting end devices (e.g., sensors) to fog nodes and fog nodes to cloud data centers. For example [9] proposes an optimization model based on energy consumption to map processing tasks over fog nodes and cloud data centers. A different approach is introduced in [10] where the focus is more oriented towards providing a good mapping between fog nodes and sensors in a heterogeneous environment. However, both these papers follow an approach where the incoming data flows and computing tasks are statically assigned over the infrastructure. Our algorithms, instead are completely dynamic and can easily adapt to workload conditions that change over time. Our experiments show that with, limited tuning, it is relatively easy to adapt to a wide range of application scenarios.

Our research is more focused on the proposal of load balancing algorithms. In this field, a significant example is an algorithm called Multi-tenant Load Distribution Algorithm for Fog Environments (MtLDF) [11], that has been proposed to optimize load balancing in Fogs environments considering specific multi-tenancy requirements. However, the proposed load balancing scheme assumes the presence of a centralized fog management layer that collects all the state information concerning the fog nodes, while our solution is fully distributed.

Another interesting approach where the tasks for the fog noses are classified according to their computational nature and are subsequently allocated to the appropriate host is proposed in [12]. Even if the communication is based on distributed Pub/Sub messaging pattern based on brokering system for IoT applications, the system is still based on a centralized workload balancer, unlike our fully distributed approach.

Yousefpour *et al.* [13] propose a system similar to our Sequential Forwarding algorithm. However, the proposed solution still requires a centralized repository to store the load state of each fog node. A variant of the proposed system relies on a specific communication pattern similar to a gossip protocol to send updates on the load state of each node. Our approach, based on a blind forwarding provides good performance without complex coordination structures.

A different approach is followed in [14] where, instead of flows of requests, the focus is on a batch system where incoming tasks are collected and periodically distributed to the In [14] an approach is presented to periodically distribute the incoming tasks in the edge computing network. This approach is highly effective to guarantee the respect of quality-of-service (QoS) requirements. However, such approach is not viable unless the application model handles only batch of tasks that needs to be dispatched across a network that is not subject to changes in the processing time or overload conditions. On the other hand, our proposal adopts a more general vision of online task processing suitable for a more reactive approach to the problem.

Finally, the idea of randomly selecting nodes to offload a task is used in the class of power-of-choices algorithms, adapted in [15, 16] to work in the fog deploy model. The key difference with the algorithm proposed in this paper is that tasks are forwarded without making any selection among alternatives and that self-adaption is absent.

As a final remark, it is worth to note that almost all the proposed studies do not explicitly take into account the heterogeneous nature of a fog computing infrastructure, while in our analysis we explicitly focus on this aspect.

3. Sequential forwarding algorithms

We now discuss our two proposed algorithms for load balancing, which are variants of the same central idea. This key idea is to allow fog nodes to make autonomous and uncoordinated decisions about serving or offloading a job. This decision is promptly taken on a per-job basis without the need of any coordination, e.g., a centralized entity or information about the state of other nodes. This is a distinctive feature of our design that in our opinion fits a general fog computing model, where fog nodes can be heterogeneous and can undergo different local management rules. In order to explain the load balancing algorithms, we refer to the architecture in Fig. 1, where a set of sensors send jobs to a layer of fog nodes. Each sensor communicates with one fog node. The sensor to fog node mapping can be based on geographic distance as in [9] or can exploit a more complex algorithm [10]. The workload of each fog node is typically heterogeneous, for the twofold reasons of time-dependent fluctuations in the workload patterns or due to uneven distribution of sensors among the fog nodes [17]. A similar heterogeneity can occur also in the characteristics of the fog nodes as the deployment of the infrastructure starts with a small prototype implementation that is then expanded over time with more modern fog nodes. Furthermore, fog nodes may belong to different providers that may adopt their own management rules or technology, e.g., containerization implementations based on Docker Swarm or Kubernetes from one hand, or based on VMs on the other hand. For the sake of our proposal we assume that the fog nodes expose some standard high-level interface through which they may forward jobs, e.g., HTTP endpoints, masking the actual network layer solution[2]. All nodes know the communication endpoints of the other nodes.

Our proposal consists of two algorithms aiming to define *when* a job should be forwarded to a neighbor, and *to which* neighbor the job should be forwarded. We start our presentation with a *Sequential Forwarding* algorithm; next we describe an evolution of this algorithm, namely *Adaptive Forwarding* algorithm. Finally, we discuss a baseline algorithm, namely *No LB*, that is the case where no load balancing occurs among the fog nodes.

3.1. Sequential Forwarding algorithm

The *Sequential Forwarding* algorithm uses a threshold Θ_n for each fog node n to decide if an incoming job should be forwarded to a random neighbor or not. The threshold operates on the system load, which is the number of jobs queued in the fog node (or being executed). The system load represents an estimate of the waiting time for the incoming job. An additional parameter of the algorithm is the maximum number of steps M to guarantee a limit on the delay associated with the load balancing phase.

Algorithm 1 presents the proposed load balancing mechanism. When a job arrives, if the job has not yet reached the M-th step, the system load (that is the number of jobs already scheduled for processing in the fog node) is considered. If the value does not exceed the threshold Θ_n, the job is accepted and scheduled for local processing. Otherwise, it is forwarded to a randomly-selected neighbor. We point out two main features of the proposed algorithm that are (i) the *blind* and *memoryless* nature of the algorithm so that no probing for the neighbor status nor reservation (to make sure that the job finds the resource available [18]) is required; and (ii) the ability of the algorithm to adapt to heterogeneous scenarios and to operate in a completely distributed way thanks to the per-node threshold

[2]For example, the X2 interface allows direct communications among 5G nodes.

Algorithm 1 Sequential Forwarding Algorithm

```
Require: $M$, $\Theta_n$, Job
  if Job.$\mathit{Steps}()$ $< M$ then
    if System.$\mathit{Load}()$ $\leq \Theta_n$ then
      $\mathit{ProcessLocally}$(Job)
    else
      Neigh $\leftarrow$ $\mathit{Random}$(System.$\mathit{Neighbors}()$)
      Job.$\mathit{IncrementSteps}()$
      $\mathit{Forward}$(Job, Neigh)
    end if
  else
    $\mathit{ProcessLocally}$(Job)
  end if
```

Θ_n. If the job has already been forwarded M times, it is scheduled for local processing. This makes the algorithm extremely simple to implement and facilitates its adoption among different providers.

We also detail the local processing of the job, as this task is also responsible for the drop of the job if the queue is full, as pointed out in Algorithm 2. It is worth to note that dropping a job in the case the queue becomes too long may be an extreme measure undesirable for some applications. In this analysis we prefer to focus on a simplified scenario that is easy to model and to implement rather than considering multiple queuing and dropping behavior depending on the application. Such evolution of the algorithms can be an additional extension of the present research that is left as future work.

Algorithm 2 Local processing: $\mathit{ProcessLocally}()$

```
Require: Job
  if System.$\mathit{Queue}()$ $<$ System.$\mathit{MaxQueue}()$ then
    $\mathit{Enqueue}$(Job)
  else
    $\mathit{Drop}$(Job)
  end if
```

3.2. Adaptive Sequential Forwarding algorithm

The Sequential Forwarding proposed in Sec. 3.1 has two separate parameters, Θ_n and M, that show an inherent inter-dependence: indeed, if Θ_n is low, we may have a high number of forwarding, so M may play a pivotal role. This makes the algorithm tuning complex, especially in heterogeneous scenarios, where the thresholds may be different across the infrastructure. To address this problem we introduce an adaptive version of the algorithm.

The Adaptive Sequential Forwarding algorithm (*Adaptive Forwarding* for short), is an evolution of the Sequential Forwarding proposed in Sec. 3.1 that introduces some self-tuning ability.

Algorithm 3 Adaptive Forwarding Algorithm

```
Require: $M$, Job
  $Q \leftarrow \text{System}.\mathit{MaxQueue}()$
  $\Theta \leftarrow \lceil \text{Job}.\mathit{Steps}() * Q/M \rceil$
  $\mathit{SequentialForwarding}(M, \Theta, \text{Job})$
```

Algorithm 3 shows the behavior of the Adaptive Forwarding Algorithm. The threshold Θ_n is computed in the same way for every node and grows linearly with the number of steps. If a job has never been forwarded (or has been forwarded just a few times), the job is not processed locally unless the local load is very low. On the other hand, if a job has already been forwarded several times we assume a more relaxed attitude towards the search of a fog node with a low load.

The algorithm starts requesting the maximum queue length of the fog node; next the algorithm computes the threshold Θ_n that grows linearly with the number of times the job has been forwarded. In particular, we tune the growth of the threshold in such a way that, after M steps, the threshold Θ_n is equal to the maximum queue length of a fog node, thus guaranteeing that the job will be accepted unless this last visited node has no room in its queue.

3.3. Baseline algorithm

In the performance evaluation, we consider also the *No LB* algorithm, that is the case where no load balancing occurs, as a baseline.

Considering the previously described algorithms, and given Q as the maximum length of queue (as in Algorithm 3), the behavior corresponds to the case where $\Theta_n > Q$ or to the case where $M < 1$. We expect this algorithm to suffer from a high loss rate (that is jobs dropped because the queue is full), unbalanced load (especially in scenarios with heterogeneous load distribution among the fog nodes) and, generally, poor performance.

4. Model

We start studying the algorithm under an ideal deployment composed by an infinite number of nodes and for $M = 1$. The case $M > 1$ can be incorporated in the proposed framework, but for the sake of clarity, this extension is left as future work. To capture heterogeneity, we assume that two types of fog nodes exist, type A and type B.

We define as α (β) the probability that a class A node (class B node) forwards a job to a node of the same class. Any fog node is abstracted as a FIFO queue with the same bounded number of places Q_A (Q_B), included the server. Class A (B) nodes get a nominal Poisson flow of jobs at rate λ_A (λ_B) jobs/s from directly connected users, while the service time of a job is exponentially distributed with an average processing rate of μ_A (μ_B).

The system under investigation is composed of two tagged nodes $a \in A, b \in B$ plus a set of N_A class A nodes and a set of $N_B = N_A = N$ class B nodes. We derive the main performance metric of the load balancer in the limit of $N \to \infty$. The reason for this limiting study is that it assumes independence among nodes, a desirable property that holds for simple homogeneous FIFO queues [19]. Let then assume that all the nodes of the system are independent from each other. The effect of the N nodes on a and b is equal to the probability of node a (b) being in a given state, since it represents the fraction of the N_A (N_B) nodes in the same state.

Without loss of generality, we now focus on node a. Due to the symmetry, the results for node b are the same, except for the fact of swapping labels A with B and α with β. In the following derivations, we assume that a node is in a state i when it has i tasks in its queue.

Node a may receive jobs forwarded by other nodes of the same class or from the other class with rate:

$$\lambda_{F_A} = \alpha\lambda_A\tilde{\pi}_{A\Theta_A} + (1-\beta)\lambda_B\tilde{\pi}_{B\Theta_B} \tag{1}$$

where $\tilde{\pi}_{Ai} = \sum_{j=i}^{Q_A} \pi_{Ai}$ ($\tilde{\pi}_{Bi} = \sum_{j=i}^{Q_B} \pi_{Bj}$) is the probability that a node of class A (B) is in a state higher or equal than i, indeed π_{Ai} (π_{Bi}) is the steady state probability of a node of class A (B) to be in state i. In fact, due to independence among states, $N_A\lambda_A\tilde{\pi}_{A\Theta_A}$ is the rate at which N_A class A nodes forwards job. And one of this job hits a with probability $\frac{\alpha}{N_A+1}$: the job selects nodes of the same class with probability α and picks exactly a with probability $\frac{1}{N_A+1}$. For this reason, the first term in the above expression represents the flow of jobs seen by a and coming from nodes of the same class, in the limit of $N \to \infty$. The second term has a similar interpretation.

The transition rate from the state i to $i+1$ is:

$$\lambda_{Ai} = \begin{cases} \lambda_A + \lambda_{F_A} & i \le \Theta_A \\ \lambda_{F_A} & i > \Theta_A \end{cases} \tag{2}$$

The steady state probability distribution satisfies the following standard linear set of equations:

$$Q_A\pi_A = [0, \ldots, 1]^T \tag{3}$$

where:

$$Q_A = \begin{pmatrix} -\lambda_{A_0} & \mu_A & 0 & \ldots & & 0 \\ \lambda_{A_0} & -(\lambda_{A_1} + \mu_A) & \mu_A & \ldots & & 0 \\ & \ddots & & & & \\ 0 & \lambda_{A_{i-1}} & -(\lambda_{A_i} + \mu_A) & \mu_A & \ldots & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{4}$$

The solution is found numerically as follows. The linear system is first solved using the matrix Q_A^0 with $\lambda_{Ai} = \lambda_A$. An analogous system of equations is solved using Q_B^0, where $\lambda_{Bi} = \lambda_B$. These solutions exist because they define two independent $M/M/1/Q$ Markov Chains. From these solutions, Eq. (1) (and the analogues for B) are used to compute the traffic flows for two new pair of matrix, say Q_A^1 and Q_B^1. The algorithm continues until $max\{||Q_A^n - Q_A^{n-1}||\} < \epsilon$ and $max\{||Q_B^n - Q_B^{n-1}||\} < \epsilon$.

4.1. Metrics

In this section, we derive the main performance indicator of the algorithm. Due to the symmetry we keep the node a point of view. Any metric concerning b is the a's one where B, A and α, β are swapped.

4.1.1. Blocking probability

The first metric of interest is the probability that a job is blocked p_B, namely the fraction of jobs that cannot be served by any fog node. As these jobs are dropped by the fog nodes, this metric is also referred to as loss rate or drop rate. The value of this metric is given by:

$$p_B = \frac{p_{B_A} + p_{B_B}}{2} \tag{5}$$

where the two contributions are the probability that a job received by a class A (class B) node is blocked, which are given by:

$$p_{B_A} = \tilde{\pi}_A[\alpha \pi_{A_{Q_A}} + (1-\beta)\pi_{B_{Q_B}}] \tag{6}$$

4.1.2. Response Time

The second relevant metric is the average response time, T, namely the time elapsed from when a job is received by a node until the serving nod ends to execute the job. We do not consider the delay due to a possible reply to the originator of the job. This quantity is due to the queueing waiting time (W), execution time ($\frac{1}{\mu}$), and job forwarding delay. Concerning the fist contribution, the average queue length of the node a is:

$$\overline{Q}_A = \sum_{k=1}^{Q_A} k\pi_{A_k} \tag{7}$$

The net flow of jobs entering in a is the sum of jobs from:

- users connected to a, at rate $\lambda_A(1-\tilde{\pi}_{A\Theta_A})$
- class A nodes with state above Θ_A selecting a, at rate $\tilde{\pi}_{A\Theta_A}\lambda_A\alpha(1-\pi_{Q_A})$
- class B nodes with state above Θ_B selecting a, at rate $\tilde{\pi}_{B\Theta_B}\lambda_B(1-\beta)(1-\pi_{Q_A})$.

Hence, applying the Little's result the queue's waiting time at class A nodes is:

$$W_A = \frac{\overline{Q}_A}{(1-\pi_{Q_A})(\lambda_A(1-\tilde{\pi}_{A\Theta_A}) + \lambda_A\tilde{\pi}_{A\Theta_A}\alpha + \lambda_B\tilde{\pi}_{B\Theta_B}(1-\beta))} \tag{8}$$

Since a not blocked job arriving to a class A node is served either by a class A node (hence experiencing S_A) or a class B node (hence experiencing S_B), the average response time of jobs received by class A nodes is a weighed average of the two service delays plus the average time spent to forward a job:

$$T_A = \frac{P_{S_{AA}}\left(\frac{1}{\mu_A} + W_A\right) + P_{S_{AB}}\left(\frac{1}{\mu_B} + W_B\right)}{P_{S_{AA}} + P_{S_{AB}}} + \tilde{\pi}_{A\Theta_A}\delta \tag{9}$$

where $P_{S_{AA}}$ ($P_{S_{AB}}$) is the probability that the job is served by a class A (class B) node, $\tilde{\pi}_{A\Theta_A}$ the probability a job is forwarded, and δ the average forwarding time. The probability $P_{S_{AA}}$ takes into account that the fact that the job can be forwarded and served by another class A node or directly served by a:

$$P_{S_{AA}} = \tilde{\pi}_{A\Theta_A}\alpha(1 - \pi_{A_{Q_A}}) + (1 - \tilde{\pi}_{A\Theta_A}) \tag{10}$$

while $P_{S_{AB}}$ is:

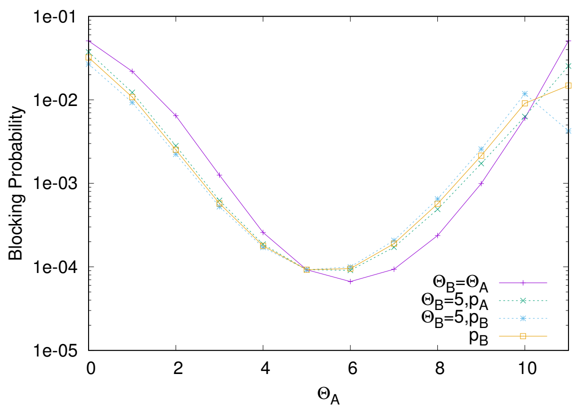
$$P_{S_{AB}} = \tilde{\pi}_{A\Theta_A}(1 - \alpha)(1 - \pi_{B_{Q_B}}) \tag{11}$$

Finally:

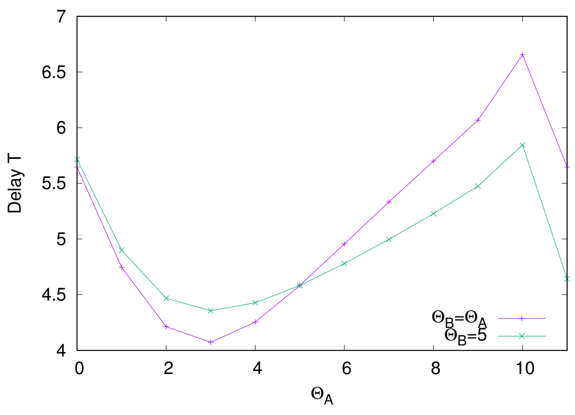
$$T = \frac{T_A + T_B}{2} \tag{12}$$

4.2. Result

In this section we report some representative results obtained from our model, when $\alpha = \beta = 0.5$. Fig. 2 reports the blocking probability and the response time as a function of Θ_A for the homogeneous case, i.e., when nodes have the same speed. The optimal threshold that minimizes the blocking probability is when $\Theta_A = \Theta_B = 6$ (as shown in Fig. 2a), i.e., roughly half of the total queue length, which is also an intuitive result: if the current length is too small, there is also a small chance for the job of landing on a less loaded queue, whereas if the length is too high load balancing doesn't arise since the job cannot return to the original node. The lowest response time is however obtained for the different threshold value $\Theta_A = \Theta_B = 3$, as shown in Fig. 2b. The reason is that nodes now drop more jobs and hence the queue length is shorter. Reducing the threshold further will also progressively eliminate the load balancing effect and hence the average queue length increases again. Both figures also show a representative case of thresholds tuned differently, in particular when $\Theta_B = 5$ while Θ_A is free to vary.



(a) Blocking probability

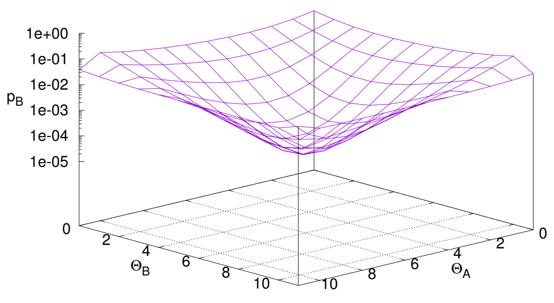


(b) Response time

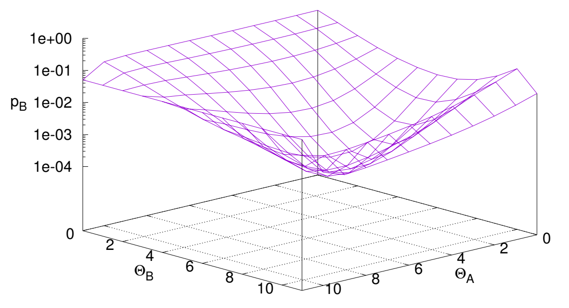
Figure 2: Performance metrics vs Θ_A, same traffic and service times.

Figure 3a shows the blocking probability for the full combination of thresholds for the homogeneous case and Figure 3b for when the speed of class B node is twice the A's one.

We can see how there is still an advantage despite the heterogeneity among nodes, and that the best threshold combination is now slightly different.



(a) $\mu_B = \mu_A$



(b) $\mu_B = 2\mu_A$

Figure 3: Loss rate vs. thresholds, traffic intensity $\rho = 0.9$

λ_A	λ_B	μ_A	μ_B	ρ_A	ρ_B	Θ_A^*	Θ_B^*	p_{B_0}	p_B^*	T_0	T^*
0.85	0.85	1.00	1.00	0.85	0.85	6	6	3.55e-02	5.12e-06	5.22	4.52
0.90	0.90	1.00	1.00	0.90	0.90	6	6	5.08e-02	6.67e-05	5.65	4.96
0.95	0.95	1.00	1.00	0.95	0.95	6	6	6.94e-02	1.18e-03	6.08	5.65
0.99	0.99	1.00	1.00	0.99	0.99	7	7	8.64e-02	1.06e-02	6.42	7.02
0.85	0.90	1.00	1.00	0.85	0.90	6	6	4.31e-02	1.85e-05	5.43	4.72
0.95	0.90	1.00	1.00	0.95	0.90	6	6	6.01e-02	2.71e-04	5.86	5.25
0.99	0.90	1.00	1.00	0.99	0.90	6	6	6.86e-02	9.05e-04	6.03	5.56
0.90	0.90	**0.90**	1.00	1.00	0.90	6	7	7.09e-02	1.12e-03	6.43	6.13
0.90	0.90	**1.10**	1.00	0.82	0.90	6	6	3.91e-02	8.66e-06	5.07	4.38
0.90	0.90	**1.20**	1.00	0.75	0.90	6	5	3.28e-02	1.74e-06	4.66	3.81
1.80	0.90	**2.00**	1.00	0.90	0.90	7	5	5.08e-02	1.12e-04	4.23	3.65
2.70	0.90	**3.00**	1.00	0.90	0.90	8	5	5.08e-02	2.61e-04	3.76	3.33
3.60	0.90	**4.00**	1.00	0.90	0.90	8	4	5.08e-02	4.97e-04	3.53	2.96

Table 1: Optimal thresholds that minimize the blocking probability.

To better explore the range of applicability of the proposed protocol, Tab. 1 reports the highest possible reduction of the blocking probability under a variety of conditions, obtained by setting the thresholds to the values Θ_A^*, Θ_B^* that minimize p_B. The results are divided into four groups. In the first one, the offered traffic changes for all nodes in the same way and nodes are also equal in terms of execution speed. i.e., the system is homogeneous. The result confirms that the best threshold is almost half of the queue length. Load balancing can reduce up to four orders of magnitude the blocking probability for moderate traffic with respect to the case in which there is no cooperation, reported in column p_{B_0}, and it allows reducing at the same time the average delay (see T^* and T_0) for all cases but the high load case. As the load increases to 0.99 in fact, while there is still an improvement in terms of p_B the average response time is higher compared to no cooperation just because more jobs are queued. Moreover, the table reports the result of three heterogeneous cases, where nodes are different because type A nodes: (i) get a different traffic flow, (ii) have a different execution time, (iii) have different traffic and execution time but their traffic intensity $\rho = \frac{\lambda}{\mu}$ is constant. The changed parameter is given in bold. For all scenarios, the algorithm is able to reduce the blocking probability of at most two orders of magnitude. The response time T^* is also always lower than the no cooperative case.

Once the analysis provided a generally positive assessment concerning the expected behavior of the protocol, we are now ready to consider more realistic cases, with a finite number of nodes and higher M.

5. Simulation Results

In the present section, we evaluate the performance of the proposed algorithms relying on a simulation approach. Throughout these tests we will consider the Sequential Forwarding algorithm (*Seq Fwd* described in Sec. 3.1), the Adaptive Forwarding alternative (*Adapt Fwd*, Sec. 3.2) and the case where no load balancing occurs (*No LB*). We start providing a summary of the tests carried out, presenting the reference experimental scenarios. Next, we consider a simplified scenario where we have two classes of fog nodes, namely A and B, characterized by different configurations and, possibly, by different computing power. In this scenario we first validate the simulation results against the numerical model presented in Sec. 4 and, next, we discuss the main findings of the simulation-based performance evaluation. After this preliminary study, we carry out a thorough sensitivity analysis with respect to the main parameters that may determine a scenario heterogeneity, regarding different computational power of the fog nodes and different distribution of A and B fog nodes populations. Finally, we focus on a realistic geographic setup for a smart city and we evaluate in detail both the Sequential Forwarding and the Adaptive Forwarding algorithms.

5.1. Scenarios definition

The first scenario used in our experiments is named *simplified scenario*. We model the fog nodes as M/M/1/Q queuing systems, with an exponential distribution of both incoming jobs from the sensors and job service in the fog nodes. We consider two populations of fog nodes, namely A and B characterized by a different processing power such that $\mu_A \geq \mu_B$, with $\mu_B = 1.0$ jobs/sec. Let N_A be the number of nodes of class A and N_B be the number of node of type B. Concerning the workload, we consider that every fog node receives the same workload intensity $\lambda_A = \lambda_B = \lambda$. To make sure that the global load on the fog infrastructure is $\rho = 0.9$, we consider that $\rho = (N_A + N_B)\lambda/(N_A\mu_A + N_B\mu_B)$ and we derive λ from this formula. For each fog node, maximum the queue length is set to $Q = 10$. Additional parameters related to adaptive queue size or the possibility to drop only some classes of jobs are not considered in the experiments. Indeed, taking into account all these options would result in combinatorial explosion of the parameter space. This would make the analysis hard to perform and to present in the space of a single research paper. Every sensor sends jobs to just one fog node, and the sensor-to-fog mapping is statically assigned. The delay experienced every time a job is forwarded corresponds to the network latency between each pair of fog nodes that is $\delta = 0.9$s (that is the network delayδ is comparable with the average service time $1/\mu$). In this scenario the simulation considers a set of 50 fog nodes, that should be enough to capture the main characteristics of the algorithm performance. Each simulation is repeated 5 times and the results are averaged over the runs.

The second scenario, called *realistic scenario*, is based on a smart city case study based in Modena, a city in northern Italy with roughly 180'000 inhabitants. The fog infrastructure aims to support a smart city application that provides environmental sensing and vehicular traffic monitoring. Sensors located along the main city streets collect data on air quality (e.g., atmospheric pollutants such as suspended particulate) and vehicular traffic (e.g., number and speed of vehicles). The goal is to provide in real time a detailed model on urban traffic and air pollution. The fog layer is composed of fog nodes placed in facilities belonging to the municipality that exchange information with the sensors and among themselves using long-range wireless links (such as IEEE 802.11ah/802.11af [20]). Each sensor communicates with the nearest fog node, as in [9], and we assume the delay among fog nodes to be proportional to the distance between them. Concerning the processing capability of the fog nodes, we developed a prototype software that counts the number of vehicles in a frame taken from a camera connected to the sensor. Based on these experiments, we modeled the processing time using a Gaussian probability distribution with an average $1/\mu = 10$ms (and with a standard deviation of 1ms). The network delay is proportional to the distance between nodes, but, throughout the infrastructure, is normalized to have an average delay of $\delta = 10$ms, that is comparable with the processing time. As in the previous scenario, we also introduce a population of A nodes that are faster and are characterized by a processing rate that is double compared to the standard B-class nodes ($\mu_A = 2\mu_B$). We select 10% of the nodes as being of A-class, and the selected nodes are the ones receiving the highest amount of jobs from the sensors. The process of producing images from the sensors is modeled using an exponential distribution. The topology is generated from real geographic data using the PAFFI framework [17], with 100 sensors and 20 fog nodes. The geographic placement of sensors, results in heterogeneous workload distributions among the nodes, ranging from 250 jobs/sec to some fog nodes that are almost idle. The average load over the whole infrastructure is such that the average utilization $\overline{\rho} = \overline{\lambda}/\overline{\mu} = 0.9$.

We summarize the main parameters of the realistic scenario in Fig. 4. Each fog node is represented by means of two circles: the thin circle represents its incoming load, while the thick circle is the processing power. If the thin circle is outside the thick one, the node is at risk of overload. Otherwise, if the thin circle is inside the thick one, no overload should occur. Moreover, in this figure we represent A and B fog nodes classes using two different colors (purple and green, respectively).

From a software tools point of view, the simulation is based on the Omnet++ frame-

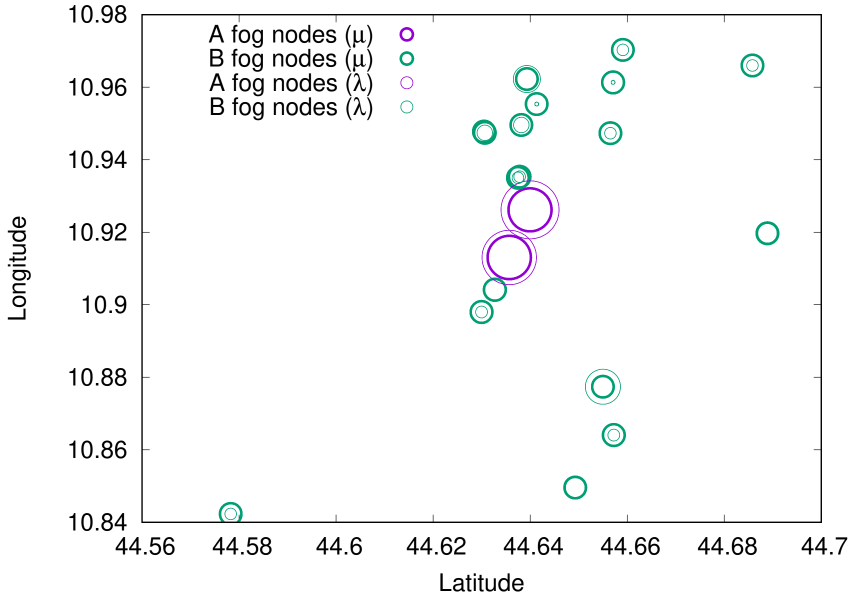


Figure 4: Realistic scenario - map representation

work[3], with additional modules developed *ad-hoc* to support the two proposed load balancing algorithms.

Throughout the performance evaluation, the main considered performance metrics are:

- *Loss rate*, that is the probability of a job being dropped because the queue of the selected fog node is full. This condition is described for the model in Sec. 4.1.1.
- *Response time*, that is the time occurring between the moment the job is received from the first fog node, to the moment the processing ends on the final fog node. The response time model is described in Sec. 4.1.2. In our experiments, we provide a breakdown of the response time components: that are *service time* (T_{Srv}, the time spent being processed), *balancer time* (T_{Bal}, the time spent being forwarded among the fog nodes), and *queuing time* (T_{Queue}, the time spent in the fog node ready queue waiting to be processed).

As a reference for the reader we summarize symbols and metrics in Table 2, together with their units of measure.

5.2. Simulation validation

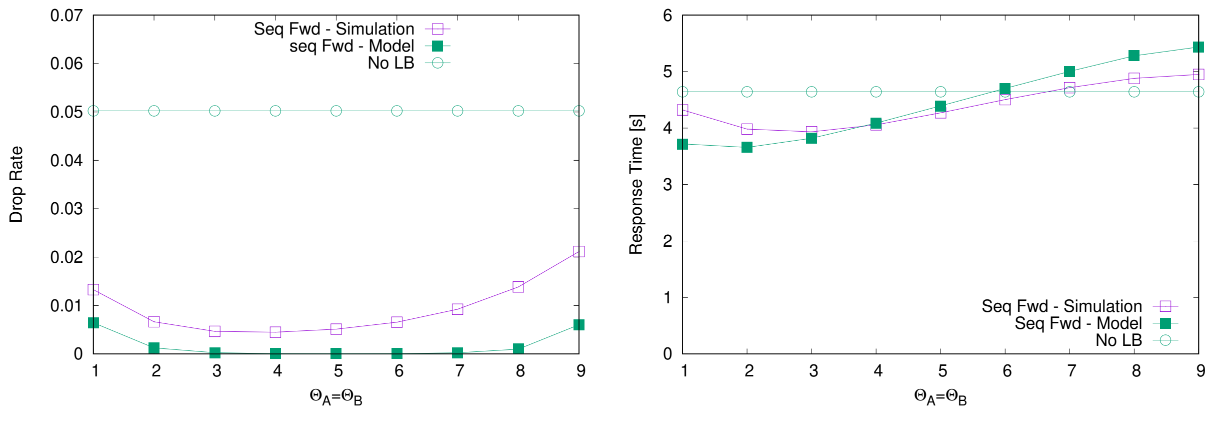
The first step in our analysis is a cross-validation between the results obtained with the simulator and the results of the theoretical model described in Sec. 4. Specifically, we compare the performance of the sequential forwarding algorithm in the simplified scenario with restrictive hypotheses that are: $N_A = N_B$ (that is we have the same amount of A-class

[3]https://omnetpp.org/

Table 2: Parameters and metrics for the proposed experimental scenarios.

Scenario parameters	
N_A, N_B	Percentage of type A nodes (B) [%]
λ_A, λ_B	incoming job rate in type A nodes (B) [jobs/s]
μ_A, μ_B	processing rate of type A nodes (B) [jobs/s]
Θ_A, Θ_B	Threshold on type A nodes (B)
M	Maximum number of steps
Q	Maximum queue length
Metrics	
T_{Resp}	Response time. Time elapsed between receiving a job from a sensor and completing its processing. $T_{Resp} = T_{Bal} + T_{Queue} + T_{Srv}$ [s]
T_{Bal}	Time spent in the load balancing phase [s]
T_{Queue}	Time spent waiting in queue [s]
T_{Srv}	Time spent being processed ($= 1/\mu$) [s]
Drop rate	Fraction of job dropped (values in range $[0, 1]$)

and B-class nodes such that the probabilities α and β are the same), $\mu_A = \mu_B$, $\Theta_A = \Theta_B$. Furthermore, due to the limitation of the model, we set the maximum number of hops $M = 1$.



(a) Drop rate (b) Response Time

Figure 5: Comparison with model ($M = 1$)

Fig. 5 compares the drop rate and the response times as a function of $\Theta_A = \Theta_B$. We show both the sequential forwarding algorithm and the case where no load balancing occurs.

Focusing on Fig. 5a on the drop rate (that corresponds with the blocking probability in the theoretical model of Sec. 4.1.1), we observe that the performance of the No LB case is poor, with a loss rate close to 5%. For the proposed algorithm both the simulation and the model confirm a similar behavior resulting in a U-shaped curve where the values of $\Theta_A = \Theta_B$

very high or very low provide higher drop rates. Indeed, when the threshold is low, most jobs are forwarded to a random neighbor that may be overloaded. In a similar way, when the threshold is high, the jobs are unlikely to be forwarded and are processed locally even is the node is at risk of overload. Finally, comparing the results of the model and the simulator, we observe that both approaches capture the main characteristics of the algorithm. The discrepancy in the numeric value is likely due to the relatively low number of nodes used in the simulation.

Fig. 5b shows the response time of a job using both the simulator and the model, compared with the case where no load balancing occurs. Again, we observe that the simulator and the model present a similar behavior, with a range of threshold values ($\Theta_A = \Theta_B < 7$) where the proposed algorithm outperforms the case where no load balancing occurs (please note that the relatively good performance of the non-cooperative approach are due to the high loss rate that reduces the amount of jobs that are served).

5.3. Evaluation in the simplified scenario

We now provide a more detailed analysis of the proposed algorithms carried out with the simulator.

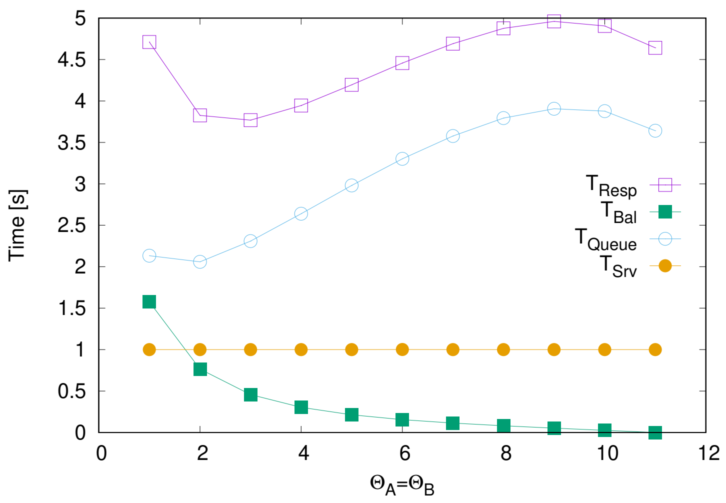
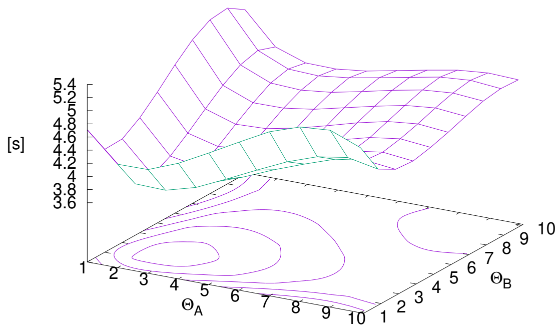


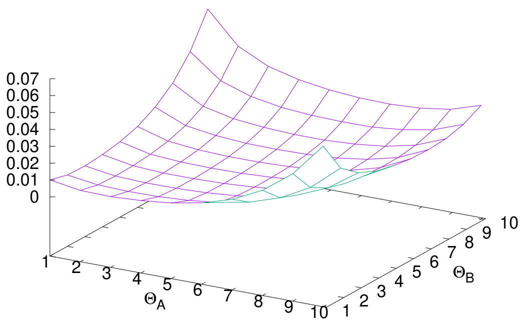
Figure 6: Breakdown of response time

Fig. 6 shows the response time for the same conditions considered in Sec. 5.2, focusing on the sequential forwarding algorithm with $M = 10$ (as we are no longer comparing our results with the model, we no longer need to limit the number of hops). We also provide a breakdown of the response time (T_{Resp}, purple line with empty squares) in its main component. As expected the service time (T_{Srv}, yellow line with filled circles) corresponds to $1/\mu_A = 1/\mu_B$ and does not depend on the threshold. The balancer time (T_{Bal}, green line with filled squares) decreases as the threshold grows making the load balancing less aggressive; however, as the load balancer becomes less aggressive, we accept to process jobs on nodes with a longer queue, thus explaining the increase in the queuing time (T_{Queue}, blue line with empty circles). The combination of these contributions determines a local minimum of T_{Resp} for $\Theta_A = \Theta_B = 3$. It is worth to note the similarities among Fig. 6 and Fig. 2 of the model, although M is different in the two cases. Comparing the Sequential Forward algorithm with the NoLB alternative the results are clearly in favor of our proposal, with a response time reduced by 19% (3.77s *vs.* 4.64s) and a drop rate reduced by a factor of nearly 17 (0.3% *vs.* 5%)

Having discussed the behavior of the sequential forwarding algorithm in this simple experimental setup, we now introduce the impact of having parameters that differ for the two classes of fog nodes.



(a) Response Time



(b) Drop rate

Figure 7: Response time and Drop rate, $N_A = 50\%$, $N_B = 50\%$, $\mu_A = 1.0$, $\mu_B = 1.0$

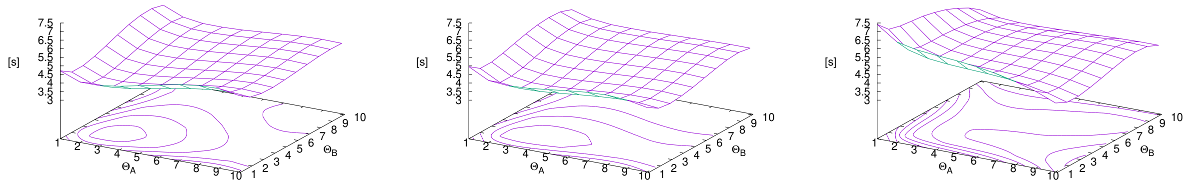
In Fig. 7 we still focus on a case where $N_A = N_B$, $\mu_A = \mu_B$ to understand the impact of the different threshold values Θ_A and Θ_B over the infrastructure. In particular, we consider the drop rate and the response time as the main performance metrics.

Fig. 7a shows the response time as a function of Θ_A and Θ_B. If we cut the surface for $\Theta_A = \Theta_B$, we obtain the curve in Fig. 6. Considering the whole space, we observe that, when a threshold (for example Θ_A) is very low, we experience an increase of response time due to the higher number of hops experienced by the jobs passing trough the node of that class (in the example, A). As the threshold increases (again, let us consider the case where Θ_A grows), we accept to process jobs on A-class nodes with a higher load. This increases the queuing time, and, as a result, leads to higher response times. The figure shows the presence of an optimal configuration for the response time when $\Theta_A = \Theta_B = 3$.

Fig. 7b shows the drop rate. Again, we observe that, as $\Theta_A = \Theta_B$ we have the U-shaped curve similar to the one in Fig. 5a. On the other hand, as we explore a parameter space where, for example $\Theta_A \ll \Theta_B$, the low threshold in the A-class nodes determines a higher load in the B-class nodes (a similar behavior is shown for Fig. 2 in Sec. 4) resulting in a global increase of the drop rate. The same effect occurs for $\Theta_B \gg \Theta_A$. As for the results in Fig 7a, we observe a configuration ($\Theta_A = \Theta_B = 6$) that minimizes the drop rate. The results are consistent with the findings of Tab. 1 obtained using the theoretical model.

5.4. Sensitivity to μ_A

Having provided some insight on the behavior of the Sequential Forwarding algorithm, we now consider how its performance is affected by a change in the processing power of the A-class nodes. For this analysis, we consider a population with 50% A-class nodes and 50% B-class nodes. Throughout our experiments $\mu_B = 1.0$ jobs/s while $\mu_A \in [1.0, 2.0]$ jobs/s. Again we point out that, unlike the experiments in Sec. 4, we have $\lambda_A = \lambda_B = \lambda$ for both A-class and B-class fog nodes and λ grows with the average computing power of the infrastructure so that the average utilization of the infrastructure is $\rho = 0.9$. This means that, for same configurations we have a potential overload of half of the infrastructure, creating a major challenge for the load balancing algorithms.



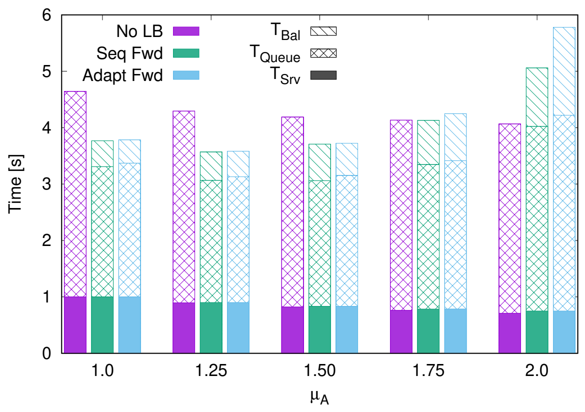
(a) $\mu_A = 1.0$ (b) $\mu_A = 1.5$ (c) $\mu_A = 2.0$

Figure 8: Response time for different μ_A

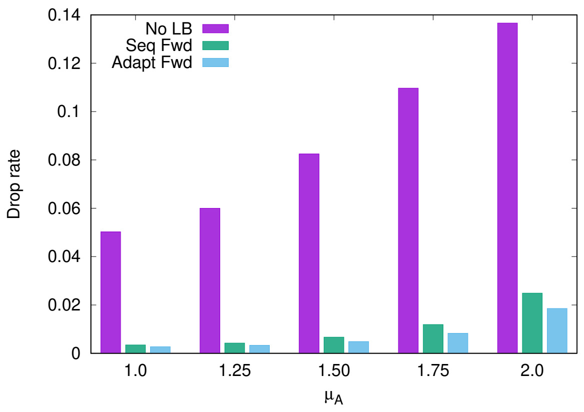
Figure 8 shows the average response time as a function of Θ_A, Θ_B and describes how such metric changes with μ_A for three values of μ_A. We observe that as the A-class nodes become more powerful, the surface becomes asymmetric stretching towards increasing values of Θ_A. Contour lines in Fig. 8a and 8b help observe this effect. However, as μ_A approaches 2.0 the shape of the curve changes significantly. This is caused by the incoming load λ exceeding the μ_B. As a consequence a large fraction of the infrastructure is at risk of overload and the low response times for high threshold values (e.g., $\Theta_A = 10, \Theta_B = 3$) corresponds to trashing conditions of the system with a significant drop rate (in some cases higher than 10%).

We now provide a comparison of the alternatives. In particular, we focus on the *No LB* case, on the Sequential Forwarding and on the Adaptive Forwarding algorithms. For the Sequential Forwarding algorithm, we tune Θ_A and Θ_B (for every value of μ_A) to provide the best response time without causing unacceptably high drop rates. For the adaptive algorithm, that aims at requiring little tuning, we consider the same value of M for every μ_A (the value $M = 6$ was found in preliminary experiments as a good trade-off between a smooth increase in the threshold and the ability to adapt to heterogeneous conditions reducing the number of hops).

For each column in Fig. 9a we provide a breakdown of the response time divided in service time (T_{Srv}, solid color, bottom part of the histogram), queuing time (T_{Queue}, crossed pattern, middle part of the histogram) and balancing time (T_{Bal} oblique pattern, top of the histogram). In the no LB case, the balancing time is obviously absent. Looking at the response time histograms we observe a general performance degradation for the two



(a) Response Time



(b) Drop rate

Figure 9: Response time and Drop rate for different μ_A

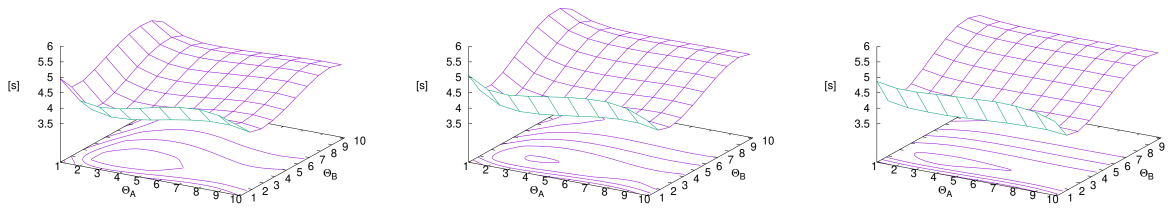
proposed algorithms as μ_A grows. This is due to the scenario that increases lambda as μ_A increases. For high values of lambda, half of the infrastructure is at risk of overload with the twofold effect of the need to forward more jobs (resulting in a growth of T_{Bal}) and of increasing the waiting time as the queue tends to be longer in the nodes at risk of overload (explaining the increase in T_{Queue}). The No LB case seems to provide better performance as the load increases, but this is an effect of the high drop rate shown in Fig. 9b that increases with the load. Indeed, for the NoLB approach the drop rate increases from 5% to 13.7%. For the cooperative algorithms, the drop rate remains below 1% as long as μ_A remains less or equal to 1.75, and in the worst case ($\mu_A = 2.0$) it reaches 2.5 for the Sequential Forward algorithm and remains below 2% for the Adaptive Sequential one. Hence the drop rate for the proposed algorithms remains from 19 to 5.5 times lower compared to the NoLB alternative.

Comparing the Sequential Forwarding and the Adaptive Forwarding algorithms, we observe that the two alternatives provide similar performance, with the adaptive solution offering slightly better performance in terms of drop rate and the sequential forwarding achieving slightly lower response times. However, it is worth to note that the adaptive algorithm provides a major advantage as it can reach a performance level similar to the Sequential Forwarding alternative but does not require a complex tuning of the threshold.

5.5. Sensitivity to N_A and N_B

The second sensitivity analysis carried out in our experiments concerns the ratio between the A-class and the B-class nodes. In this case, we keep the values of μ_A and μ_B fixed ($\mu_A = 1.5, \mu_B = 1.0$ jobs/s).

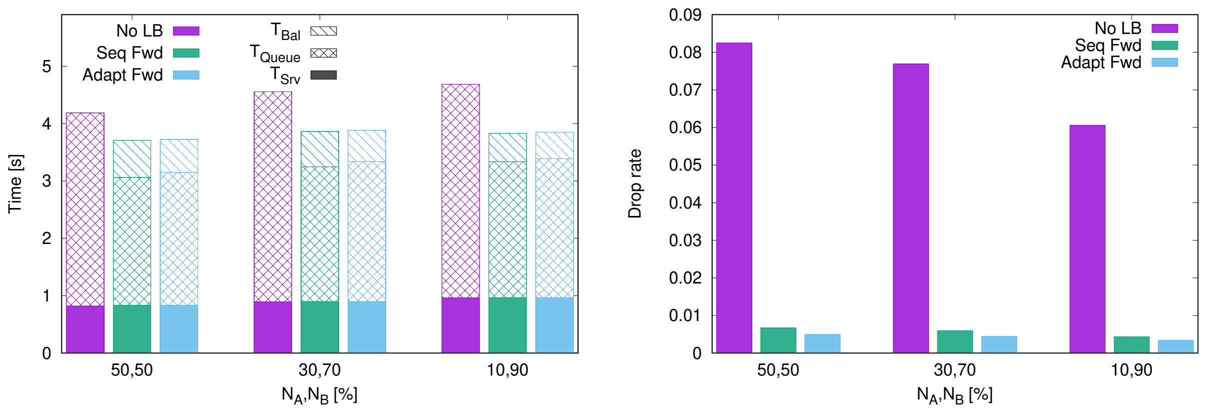
As in the previous sensitivity analysis, Fig. 10 shows the response time for different percentages of A-class and B-class nodes (represented with the N_A and N_B parameters). We observe that the overall shape of the surface remains similar. However, as we reduce the number of A-class nodes, we observe that the contour lines of the figures become more and more similar to parallel lines in the direction of a single value of Θ_B. This means that the



(a) $N_A = 50\%$, $N_B = 50\%$ (b) $N_A = 30\%$, $N_B = 70\%$ (c) $N_A = 10\%$, $N_B = 90\%$

Figure 10: Response time for different N_A, N_B

weight of the Θ_A parameter becomes less and less significant compared to Θ_B. On one hand, this result is expected because, as we reduce the number of fast A-class nodes, the slower and more numerous B class nodes become the real bottleneck of the infrastructure. On the other hand, this result provides an important lesson to learn: adding a few powerful nodes in a slow infrastructure is unlikely to solve any performance problem unless an adequate tuning of the large fraction of the remaining nodes is carried out.



(a) Response Time (b) Drop rate

Figure 11: Response time and Drop rate for different N_A, N_B

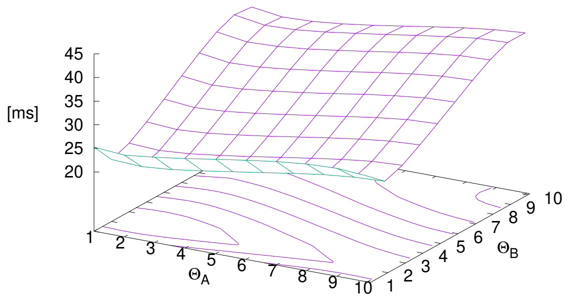
Figure 11 summarizes the main findings of the sensitivity analysis with respect to the N_A, N_B parameters. As in the previous analysis, we present a breakdown of the response time with service time (T_{Srv}), queuing time (T_{Queue}) and balancing time (T_{Bal}) in Fig. 11a. Again, for the Sequential Forwarding algorithm, the results are referred to the best scenario for every (Θ_A, Θ_B) couple considered, while for the Adaptive Forwarding algorithm we relay on the initial tuning. Fig. 11b shows the drop rate for the various considered alternatives. From a comparison of the three considered options, the No LB is clearly the worse solution, with higher drop rate and higher response time (in this case the infrastructure does not reach a level of trashing such that the drop rate reduces significantly the response time). Indeed, the response time remains from 11% to 19% higher compared to the proposed algorithms while the drop rate of the NoLB approach remains roughly 15 times higher compared to our proposals.

On the other hand, the two other load balancing algorithms provide similar performance in terms of drop rate and response time, confirming the main finding of Sec. 5.4.

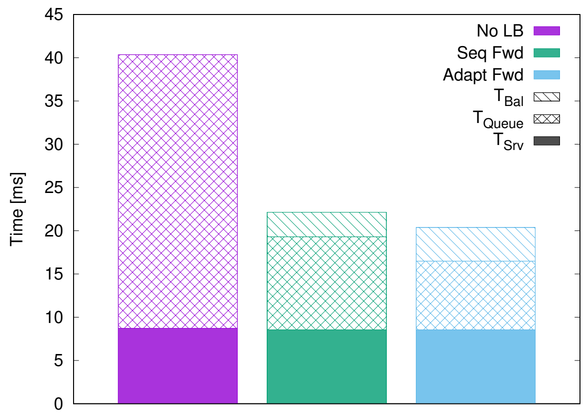
5.6. Evaluation in the realistic scenario

We now focus on the realistic scenario, that is a case where loads are unevenly distributed over the fog nodes, the network link delays are uneven and where the processing time is no longer described as an exponential (that is the fog nodes are described as M/G/1/Q queuing network elements). We recall that in this scenario, the setup is based on a geographic placement of nodes based on real locations.

Fig. 12 provides a performance evaluation for the different considered algorithms in terms of response time. Specifically, Fig. 12a shows the response time for the Sequential Forwarding algorithm as a function of the threshold Θ_A and Θ_B. We confirm the main findings of Sec. 5.5 about the major impact of the threshold Θ_B that affects the performance of the more numerous slower nodes. However, the high incoming load in the A-class nodes (we recall that in this scenario $\lambda_A \gg \lambda_B$ due to the criteria used to select the A-class nodes) makes the impact of the parameter Θ_A less negligible compared to the analysis in 5.5. Using the results in Fig. 12a to tune the Sequential Forwarding algorithm, in Fig. 12b we compare the response times of the three considered alternatives, that is the No LB case, and the Sequential Forwarding and Adaptive Forwarding algorithms. As in the previous analyses, for the adaptive algorithm, we consider $M = 6$ that is the value identified previously as a value providing good and stable performance. It is worth to note that we do not provide a figure concerning the drop rate because the two algorithms that provide load balancing have a drop rate very low (less than 0.2%), while the No LB alternative is characterized by a drop rate in the order of 13%.



(a) Response time of sequential forwarding algorithm



(b) Response time comparison

Figure 12: Performance evaluation in the realistic scenario

The main outcome of this comparison confirms the main finding of the previous experiments: the two load balancing algorithms provide similar performance both in terms of response time and drop rate (response time between 19 and 21 ms, with a drop rate in both cases below 0.2%) and far outperform the NoLB approach that is characterized by a response time nearly double compared to our algorithms. However, we point out once again that the good performance of the sequential forwarding algorithm is the result of careful tuning of the algorithm parameters, while the adaptive alternative provides good and stable performance with minimal effort.

6. Conclusions and Future Work

Throughout this paper, we proposed two innovative algorithms, namely *Sequential Forwarding* and *Adaptive Forwarding*, that aims to provide load balancing in a fog computing infrastructure. The two algorithms are explicitly designed to be extremely simple and to be fully distributed to provide a fair load sharing in highly heterogeneous scenarios with variable workload levels and high network delays that are typical characteristics of fog computing.

The analysis of the protocol is strongly focused on exploring the impact of heterogeneous fog infrastructure on the behavior of the algorithms. We consider a population of fog nodes that differ for number, processing power and configuration, exploring the impact of each aspect on the overall system performance. Our tests consider both a simplified and controlled environment (used for the sensitivity analyses) and a realistic scenario based on the design of a geographic deployment of a smart city fog infrastructure. Another qualifying point of our analysis is that we combine both a mathematical model and a simulation-based approach, to cross-validate our main findings.

The results of our experiments suggest that the proposed algorithms clearly outperform the case where no load balancing is applied with a reduction in the drop rate by a factor of 19 and by a reduction in the response time up to 19%. The results in a realistic scenario are even more impressive as we can nearly halve the response time and we can reduce the loss rate from 13% to less than 0.2%.

As a further remark, we point out that the self-tuning adaptation mechanism of the Adaptive Forwarding algorithm provides stable performance in terms of low response time and low loss rate comparable with the Sequential Forwarding algorithm, but it does not require a careful parameter tuning, and therefore, it is easier to deploy in real and highly heterogeneous scenarios.

Finally, in this paper we focused on the basic behavior of the algorithm. Additional features such as managing jobs with different priorities and different dropping policies is an interesting space not considered in the present paper but that could be addressed in future works.

References

[1] M. Satyanarayanan, W. Gao, B. Lucia, The computing landscape of the 21st century, in: Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications, HotMobile '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 45–50. doi:10.1145/3301293.3302357. URL `https://doi.org/10.1145/3301293.3302357`

[2] OpenFog Consortium Architecture Working Group, Openfog reference architecture for fog computing, Tech. Rep. OPFRA001.020817, OpenFog Consortium (Feb 2017).

[3] 5G PPP Architecture Working Group and others, View on 5g architecture, White Paper, December.

[4] J. Hong, Y. Hong, J. Youn, Problem statement of iot integrated with edge computing (2018).

[5] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, D. Raychaudhuri, Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1270–1278.

[6] H. Inaltekin, M. Gorlatova, M. Chiang, Virtualized control over fog: Interplay between reliability and latency, IEEE Internet of Things Journal 5 (6) (2018) 5030–5045. doi:10.1109/JIOT.2018.2881202.

[7] C. Bailas, M. Marsden, D. Zhang, N. E. O'Connor, S. Little, Performance of video processing at the edge for crowd-monitoring applications, in: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), 2018, pp. 482–487. doi:10.1109/WF-IoT.2018.8355170.

[8] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, P. A. Polakos, A comprehensive survey on fog computing: State-of-the-art and research challenges, IEEE Communications Surveys Tutorials 20 (1) (2018) 416–464. doi:10.1109/COMST.2017.2771153.

[9] R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption, IEEE Internet of Things Journal 3 (6) (2016) 1171–1181.

[10] C. Canali, R. Lancellotti, GASP: Genetic Algorithms for Service Placement in fog computing systems, Algorithms 12 (10) (2019) 1–19. doi:10.3390/a12100201.

[11] E. C. P. Neto, G. Callou, F. Aires, An algorithm to optimise the load distribution of fog environments, in: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017, pp. 1292–1297. doi:10.1109/SMC.2017.8122791.

[12] A. Kapsalis, P. Kasnesis, I. S. Venieris, D. I. Kaklamani, C. Z. Patrikakis, A cooperative fog approach for effective workload balancing, IEEE Cloud Computing 4 (2) (2017) 36–45. doi:10.1109/MCC.2017.25.

[13] A. Yousefpour, G. Ishigaki, J. P. Jue, Fog Computing: Towards Minimizing Delay in the Internet of Things, Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017 (2017) 17–24.

[14] Y. Song, S. S. Yau, R. Yu, X. Zhang, G. Xue, An approach to qos-based task distribution in edge computing networks for iot applications, in: 2017 IEEE International Conference on Edge Computing (EDGE), 2017, pp. 32–39. doi:10.1109/IEEE.EDGE.2017.50.

[15] R. Beraldi, H. Alnuweiri, A. Mtibaa, A power-of-two choices based algorithm for fog computing, IEEE Transactions on Cloud Computing (2018) 1–1doi:10.1109/TCC.2018.2828809.

[16] R. Beraldi, H. Alnuweiri, Exploiting power-of-choices for load balancing in fog computing, in: 2019 IEEE International Conference on Fog Computing (ICFC), 2019, pp. 80–86. doi:10.1109/ICFC.2019.00019.

[17] C. Canali, R. Lancellotti, Paffi: Performance analysis framework for fog infrastructures in realistic scenarios, in: 2019 4th International Conference on Computing, Communications and Security (ICCCS), 2019, pp. 1–8.

[18] R. Beraldi, H. Alnuweiri, Sequential randomization load balancing for fog computing, in: 2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2018, pp. 1–6. doi:10.23919/SOFTCOM.2018.8555797.

[19] C. Graham, Chaoticity on path space for a queueing network with selection of the shortest queue among several, Journal of Applied Probability 1 (37) (2000) 198–211.

[20] E. Khorov, A. Lyakhov, A. Krotov, A. Guschin, A survey on IEEE 802.11 ah: An enabling networking technology for smart cities, Computer Communications 58 (2015) 53–69.