On the impact of stale information on distributed online load balancing protocols for edge computing

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, Via Ariosto 25, Rome, 00185, Italy*
[b]*Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Via Universita' 4, Modena, 41121, Italy*

Abstract

The distributed nature of edge computing infrastructures requires a significant effort to avoid overload conditions due to uneven distribution of incoming load from sensors placed over a wide area. While optimization algorithms operating offline can address this issue in the medium to long term, sudden and unexpected traffic surges require an online approach where load balancing actions are taken at a smaller time scale. However, when the service time of a single request becomes comparable with the latency needed to take and actuate load balancing decisions, the design of online approaches becomes particularly challenging.

This paper focuses on the class of online algorithms for load balancing based on resource sharing among random nodes. While this randomization principle is a straightforward and effective way to share resources and achieve load balance, it fails to work properly when the interval between decision making and decision actuating times (called *schedule lag*) becomes comparable with the time required to execute a job, a condition not rare in edge computing systems, and provokes stale (out-of-date) information to be involved in scheduling decisions. Our analysis combines (1) a theoretical model that evaluates how stale information reduces the effectiveness of the balancing mechanism and describes the correlation between the system state at decision making and decision actuating times; (2) a simulation approach to study a wide range of algorithm parameters and possible usage scenarios. The results of our analysis provides the the designers of distributed edge

⋆This is the post print version of the article published with DOI `https://doi.org/10.1016/j.comnet.2022.108935`.

systems with useful hints to decide, based on the scenario, which load balancing protocol is the most suitable.

Keywords: Edge Computing, Load Balancing, Power-of-choice algorithm, Stale load information, Simulation

---

1. Introduction

The approach of edge computing [1] is becoming one of the most popular solutions for the deployment of large distributed applications. Application scenarios range from the Industrial IoT to smart cities, often following the architectural blueprints of the OpenFog architecture [2]. Most edge computing applications require to pre-process, filter and aggregate on the edge nodes the data sent from a plethora of sensors before sending such refined information to a cloud data centre for additional analysis and storage [3].

This edge computing paradigm is a clear step ahead of a traditional cloud-only approach because it can guarantee lower response times to latency-sensitive applications, part of which is now executed directly on the network edge. This approach aims to reduce the backhaul traffic towards the cloud data centers [2, 3].

Edge computing resource management is an essential open issue in the research agenda [4, 5], which boils down to determining a mapping A between units of computations C, submitted to the edge layer by users, and the available edge nodes E:

$$A : C \rightarrow E$$

in a way that some performance attributes are optimised. Depending on the strategy followed to determine A, the solutions can be divided into two categories: offline and online. Let T_A be the time required to calculate $A(\cdot)$, and T_C the service time to compute C.

In offline management strategies, several units C (in this context also called *jobs*) are collected and grouped in batches. Moreover, A is a solution to a well-defined optimisation problem, where optimisation actions include job execution migration or offloading from an edge node to another one or from the edge layer to the cloud one. The prerequisite for these algorithms is that the response time is not critical. A variation to the scheme is to divide the time into intervals, observe the performance during a time interval, and migrate the expected jobs of the next time interval towards other nodes, based on an estimation of the benefit of such migration. Here the prerequisite is that the load is stationary over a suitable period of time.

On the other hand, online resource management is used when the response time is critical and hence waiting to form a batch is not possible or the load is not stationary. Online approaches apply A on-the-fly to each new submitted job C. In this case, T_A includes the time to gather information from other nodes, eventually needed to make a forwarding decision, e.g. via probing, and the job transfer time[1]. The ratio among T_A and T_C may limit the applicability of online approaches. We will refer to this ratio as the **schedule lag** and denote it as $\eta = \frac{T_A}{T_C}$. The schedule lag measures the time interval between decision making and decision actuating times.

The value η depends on the application characteristics and on the structure of the edge computing system. For example, let us consider an image processing application where the job C corresponds to detecting and recognizing objects in video frames. An application with a frame rate of 60 FPS implies that $T_C < 16.67$ ms. Assuming a high-speed connection among edge nodes, for example of 1 Gbps, and a frame size of 2 MB, the frame transfer time is about 20 ms, so that $\eta > 1$. Image compression techniques can reduce this time, but still, it is likely to have $\eta \approx 1$. More complex image processing may require higher service times leading to $\eta < 1$. Another example is sensing applications that process thin data, like those collected by sensors. Here data are likely to be a small JSON file of a few Kbytes and requires very short service times, likely providing $\eta \geq 1$.

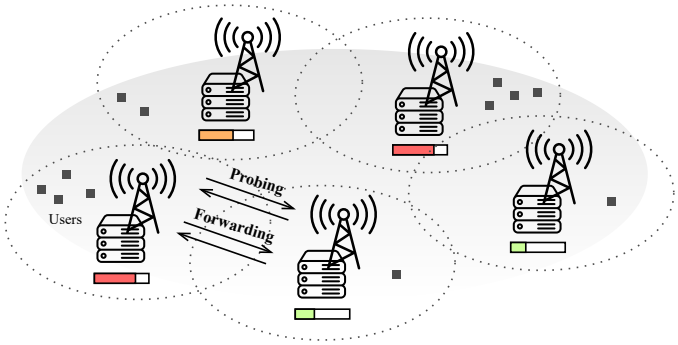


Figure 1: The cooperation in the edge computing system based on load-aware random probing: the overloaded edge node (bottom left of the figure) probes a random neighbor and then forwards a job to it.

The paper considers a family of distributed online protocols based on ran-

[1]For the sake of simplicity we count it as a component of T_A, even if strictly speaking the transfer time is needed to reach the edge node and not to determine it.

domisation algorithms in which the function A is computed by each node when it receives a new job C. The function A returns either a node id of a less loaded node picked at random among a small subset or none, which means executing the job locally. The implementation of A is done via random probing [6, 7], as shown in Figure 1.

The main contribution of the paper is a study of the impact of schedule lag η on the performance of such load *load-aware* balancing protocols based on randomization. The paper studies how and at which extent making a decision based on stale information concerning the load state of the nodes, weakens the effectiveness the algorithm and how load balancing can be achieved when this delay in communicating state information is unavoidable. The paper shows that it exists a 'critical' value of η starting from which load information has no value and a simpler blind forwarding algorithm performs better.

Figure 2 sketches this claim, in which the critical value is the dot line where $\eta = 1$. When $\eta > 1$ (upper triangle) there is a high risk for *load-aware* algorithms of schedule decisions based on stale (out-of-date) load information and consequent poor performance. On the other hand, in scenarios characterised by values of $\eta < 1$, the risk related to stale load information is low, as represented by the triangle at the bottom and a load-aware algorithm works at its best.

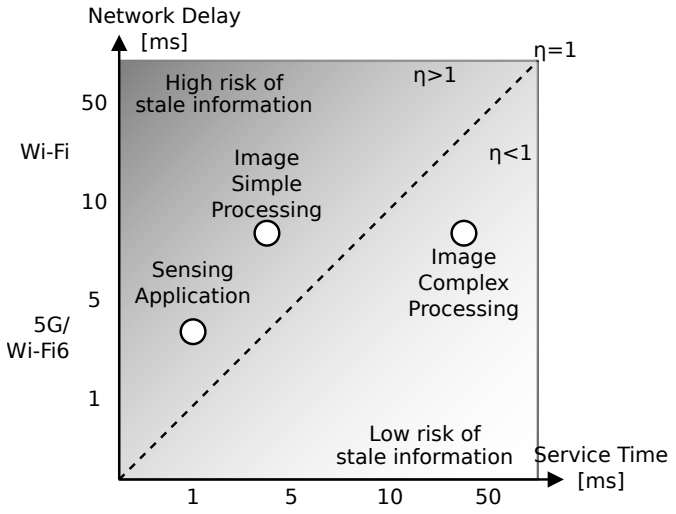


Figure 2: The figure qualitatively shows the risk of stale load information depending on η.

The paper presents a thorough analysis of the impact of stale information on

the effectiveness of load balancing protocols, with the final aim to support the system designers to decide, based on the scenario, which algorithm is most suitable. For our analysis, we consider typical application scenarios based on smart cities. We assume to have intelligent traffic lights that can monitor traffic (cars, pedestrians and bicycles) possibly equipped with cameras and sensors. The traffic information can be used for multiple purposes, from supporting autonomous driving to identifying suspicious behaviour from people.

A qualifying point of our analysis concerns the methodology used in our study. We combine a theoretical model and numerical solutions with a simulation approach. The theoretical model derives the correlation between two states as a function of η and quantifies when load-aware probe-based algorithms, which pulls load information from the other node, becomes, in fact, useless. This result is confirmed by detailed simulations that finds a similar conclusion. Moreover, a Sequential Forwarding algorithm [8], that follows a *load-blind* approach where the decision is based only on local load information, is also analysed.

Using simulations, we explore a wide set of parameters considering multiple scenarios: a *uniform mesh scenario* where incoming load is evenly distributed among nodes organised in a regular mesh, and a *geographic scenario* derived from the topology of a medium-size city in Italy.

The rest of this paper is organised as follows. Section 2 places our analysis with respect to the background, also providing an analysis of related work. Section 3 describes the proposed probe-based algorithms and the sequential forwarding algorithm used as a term of comparison. Section 4 presents a mathematical model to describe system performance. In Section 5 we provide an evaluation of the proposed load balancing algorithm based on the Omnet++ simulation framework over different scenarios. Finally, Section 6 presents the conclusions and future research directions.

2. Background and Related Work

The negative impact of stale information due to network-related delays on load balancing performance in distributed infrastructures was pointed out several years ago in the area of Web systems [9] and in a distributed setting in the seminal paper [10]. In this paper, we analyse the effect of network delays in the novel area of edge computing, in particular, our study focuses on load-aware probe-based approaches for load balancing among edge nodes, considering their potential issues. Furthermore, we do not limit our analysis to network delays. Indeed, we

consider the specific application requirements through the parameter η, which expresses the transfer-to-compute ratio, to understand under which circumstances the probe-based approach can be preferable to the load-blind approach of the sequential forwarding algorithm.

In the rest of this section we analyse the state of the art under two points of view: job offloading vs load balancing in edge systems and large distributed application scenarios.

2.1. Vertical Offloading vs Horizontal Load Balancing

With the increasing heterogeneity of the computational capabilities of mobile devices and sensors used in large distributed applications, both the edge and the cloud level can be exploited to share the load of the required data processing and satisfy the application requirements. The (total or partial) transfer of computationally intensive jobs from the local device/sensor to edge and cloud nodes, called offloading, has been widely studied in the context of mobile cloud computing [11, 12], where is used to transfer the computational load to the resource-rich cloud infrastructure.

The cloud is often remotely located and far from end-users and sensors, so the data transfer delays can be long and unpredictable. To reduce the perceived latency, mobile edge computing (MEC) has been proposed by several studies [13, 14] for offloading a part of the workload from mobile devices to the intermediate level of edge nodes with sufficient computational resources. However, these studies typically do not consider horizontal cooperation strategies for load balancing among edge nodes. Some studies focus on the edge server placement issue [15] or include dynamic service migration to deal with erratic user mobility [16] but do not consider cooperation strategies at the edge level.

On the other hand, we focus on the scenario where edge nodes trigger peer cooperation to balance the load if needed to avoid overload conditions and improve the system performance. The studies in [7, 17] adapted to the distributed fog/edge computing scenario the idea of a random selection of a node to trigger cooperation and job forwarding. This idea, which is typically used in the class of power-of-choices algorithms, was proposed by the same authors in [8] as the basis of the Sequential Forwarding algorithm considered in this paper as representative of a load-blind approach. Our contribution for the first time critically analyses whether a probe-based approach can be preferable to the load-blind one depending on network conditions and application requirements.

2.2. Application Scenarios

Many distributed applications, ranging from Industrial IoT to smart cities contexts, rely on sensors and, in general, end-devices to collect data that need to be processed for extracting features and information needed to provide the required final service [3]. Such applications may have very different requirements in terms of processing time and amount of data to be processed: these elements have a direct impact respectively on T_A (time required to calculate the mapping A between units of computations) and T_C (execution time) as defined in Section 1. In this paper, we introduce the parameter η to take into account the transfer-to-compute ratio characterising the application.

For example, a typical application that can take advantage of edge computing support is video processing: by extracting at the edge level only a few video features to be sent to the cloud, network resources can be saved and application latency decreased. The studies proposing edge computing solutions for image processing [18, 19] usually focus only on the high computational load required (sometimes addressed by splitting among edge nodes non-overlapping partitions of the video frames as in [19]) but do not consider the network-related contributions. However, depending on the specific characteristics of the application, it is not always correct to assume $\eta \ll 1$, meaning that the processing time is higher than the network contributions. In the case of high-performance image processing and high frame rate, the network latency and the jobs transfer time may become comparable or even higher than the job execution time. On the other hand, large distributed applications may also involve thin data to be transmitted and processed. This typically happens with crowdsensing applications based on data collected by geographically distributed sensors [20], but it may also be the case of Industry 4.0 or IoT-Based Manufacturing scenarios [21]: in all these applications, the data size may be around a few Kbytes (e.g., a small JSON file). However, thin data's execution time may vary significantly depending on the specific scenario. Indeed, for applications requiring simple computations of statistical indicators on the time series of the collected data, the execution time may be short and comparable with the network-related and transfer times, leading to a case characterised by a value of $\eta \approx 1$. On the other hand, in applications where machine learning techniques are applied [21] (for example, for complex forecasting, feature inference or patterns prediction), the execution time may significantly increase, leading to a scenario with $\eta < 1$. An added value of this paper with respect to the state of the art is to consider which load balancing approach is preferable depending on specific scenarios and application requirements.

3. Algorithm definition

We now introduce the two algorithms used in our study to investigate the effect of stale information on different approaches to load balancing. Specifically, we first introduce the *Probe-based* algorithm [22], representative of *load-aware* approaches, then we describe a load-blind algorithm, namely *Sequential forwarding*, presented in literature in [8].

3.1. Probe-based algorithm

The *probe-based* algorithm relies on a threshold to determine whether, upon receiving a new job, a probe for a less loaded neighbour is to be started. The threshold Θ is applied to the system load, that represents the number of jobs queued in the edge node (or being executed). This metric is used as an estimation of the waiting time for the incoming job. If the load exceeds the threshold, a probe is started, with the the edge node issuing query messages to a randomly selected neighbour.

Algorithm 1 Probe-based Algorithm

```
Require: Θ, Job
  if Job.IsForwarded() or System.Load() ≤ Θ then
    ProcessLocally(Job)
  else
    Neigh ← Random(System.Neighbors())
    NeighLoad ← ProbeNeighbor(Neigh)
    if System.Load() > NeighLoad then
      Forward(Job, Neigh)
    else
      ProcessLocally(Job)
    end if
  end if
```

Algorithm 1 presents the formalization of the probe-based algorithm. When a job from a sensor is received, the edge node uses the threshold Θ and the local load to decide if a probe for the neighbour load should be issued (jobs forwarded from other edge nodes are processed locally without additional evaluation). If probing is required, the edge node issues a query message to the neighbour and waits for the response. The neighbour provides its load status within the response, so the edge node can decide if the job has to be forwarded to the neighbour or

if the job is to be processed locally (if the neighbour has a higher load than the local node). It is worth noting that, in the case of high network delay, the load returned by a neighbour may be a *stale* information far different from the load the forwarded job will actually encounter. This may result in inaccurate forwarding decisions, already pointed out in the area of Web servers [9].

Algorithm 2 Local processing: *ProcessLocally*()

```
Require: Job
  if System.Queue() < System.MaxQueue() then
    Enqueue(Job)
  else
    Drop(Job)
  end if
```

Algorithm 2 details the case where a job is processed locally (for example, due to the call to the *ProcessLocally*() procedure in Algorithm 1). In this case, the job should be placed in the ready queue of the server. However, if the queue is already full (since it has a finite size), the job is dropped, resulting in a loss.

3.2. Sequential Forwarding algorithm

The *Sequential Forwarding* algorithm [8] uses the threshold Θ to decide if an incoming job must be forwarded to a random neighbour or locally processed. The algorithm relies on an additional parameter M, the maximum number of steps to guarantee a limit on the delay associated with the load balancing phase. Algorithm 3 presents the formalization of the algorithm. When a job arrives, if the job has not yet reached the M-th step, the system load is considered: if the value does not exceed the threshold Θ, the job is accepted and scheduled for local processing; otherwise, it is forwarded to a randomly-selected neighbour. If the job has already been forwarded M times, it is scheduled for local processing. This algorithm, which is load-blind, is extremely simple to implement. In this paper, we set the parameter $M = 5$, which is a value proved in preliminary experiments to provide low response time and low drop rate. A detailed description of this parameter and its impact on the algorithm performance has been provided in [23].

The sequential forwarding process is detailed in Algorithm 3. We assume that the data structure describing the job is enriched with metadata to keep track of the number of times the job is forwarded.

Algorithm 3 Sequential Forwarding Algorithm

Require: M, Θ, Job
  if Job.*Steps*() $\geq M$ **then**
    ProcessLocally(Job)
  else
    if System.Load() $\leq \Theta$ **then**
      ProcessLocally(Job)
    else
      Neigh $\leftarrow$ *Random*(System.*Neighbors*())
      Job.*IncrementSteps*()
      Forward(Job, Neigh)
    end if
  end if

4. System models

In this section we develop two models for the probe-based protocol that uses stale (out-of-date) information and for the sequential forwarding protocol. Table 1 summarizes the main term definitions.

4.1. A model for probe-based protocols with stale information

The probe-based algorithm relies on load state information gathered from the other nodes. We define the *schedule lag* the time interval elapsed from when a node reports its state until the node receives a job due to a scheduling decision based on that value. As the lag increases the scheduling decisions becomes sub-optimal since they are based on stale information. Intuitively this occurs because the job finds the probed node in a state which is progressively unrelated to the reported state.

We now compute this value considering the generic interaction pattern of the probe based protocol, see Figure 3. Let t_P be the time when the node A receives a job, T_P the time required to decide where to schedule a job and T_F the time required to forward. In addition, τ_P and τ_J denote respectively the transmission time of a probe or a job. The schedule lag value τ is the sum of two contributions due to queuing and transmission times.

In our model if the current state of the node is $k > \Theta$ the node stores the job into a Probe Queue (PQ) with k annotated and it selects another node B, picked at random. It waits from t_P to t_1 in the queue before the probe message for that job

Symbol	Explanation
λ	Job arrival rate.
μ	Service completion rate.
ρ	Traffic intensity ($\frac{\lambda}{\mu}$).
p_p	*Probing probability*: probability to probe a node.
τ_P	*Probe Sending time* Time required to send a probe message over the wire.
T_P	*Probe time*: total time required to decide where to serve a job, for sequential forwarding $T_P = 0$.
T_{PR}	*Probe Reply Time*: Time required to receive the state information from a node.
σ^2_{PR}	Variance of Probe reply time.
p_F	*Forwarding probability*: probability that a node forwards a job.
τ_J	*Job Sending time* time required to send a job over the wire.
T_F	*Forward Time*: total time required to forward a job.
T_{Bal}	*Balancer Time*: time required to move a job in a service queue either the local queue or a remote queue ($T_{Bal} = T_P + T_F$).
τ	*Schedule lag*: time difference between the time when a node reports its state until the node receives a job whose scheduling is based on that value.
η	Ratio between schedule lag and service time.
Θ	Activation threshold.
π_i	Probability that the state of the service queue is i.
$\tilde{\pi}_i$	Probability that the state of the service queue is at least i.
$\tilde{\pi}'_i$	Probability that the state of the service queue is at least $max\{\Theta, i\}$.
	Performance metrics
P_B	*Drop rate*: probability to drop a job
T_{Resp}	*Response Time*: Average time elapsing from when a job is received from and edge node until its service ends

Table 1: Summary of symbol definitions used in the model.

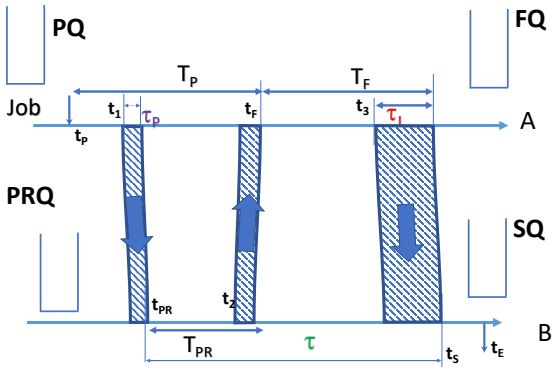


Figure 3: Time diagram model of a probe based algorithm.

is actually sent over the wire to B. At time t_{PR}, B receives the probe message, samples its load i that sends back to A through its Probe Reply Queue (PRQ). The total delay for this operation is T_{PR}, which is the first source of delay of the lag. At time t_F, node A receives the reply message and decides either to serve the job locally (if $k < i$) or to forward it to B through its Forwarding Queue (FQ). Job forwarding is the second source of the schedule lag because the job enters the service queue at time t_s when it is fully received. The value of schedule lag is then $\tau = T_P + T_F$. Note that the service time of PQ lasts from t_P to t_F and it is equal to $T_{PR} + \tau_P$. This analysis doesn't consider the propagation delay of signals because nodes are physically closed enough to neglect this contribution.

The schedule lag value is critical because it determines the probability that by the time that a job is received the state of B changes from i to j, which are exactly the events that cause sub-optimally. The goal of our model is to express such probability of as a function of τ and from here how its value affects the performance of the algorithm.

4.1.1. Model description

Based on the previous description, we now define a model for a probe based algorithm running on a large number of nodes. The model is based on three queues belonging to the node A and a queue belonging to node probed node B:

1. Probe Queue (PQ) has an unbounded capacity and stores jobs awaiting for a probe reply message; the average service time of the queue is $T_{PR} + \tau_P$ and

its variance is σ^2_{PR} (see below). The propagation delay is not considered. This queue is modelled as an $M/G/1$ queue.

2. Probe-Reply Queue (PRQ) has as unbounded capacity and constant service time τ_P equal to the transmission time of the probe reply message, which is modeled as an $M/D/1$ queue.
3. Forward Queue (FQ) has an unbound capacity for jobs to send to other nodes, whose service time is τ_J equals to the job transmission time. This queue is modelled as an $M/D/1$ queue.
4. service queue (SQ) with finite capacity K for jobs being served by the node. The service time of this queue is exponentially distributed with mean $T_S = \frac{1}{\mu}$). This time corresponds to the actual time required to process the job. The dynamic of the queue is modelled as a birth-death process.

Nodes receive jobs according to a Poisson flow with rate λ. We assume that these queues are independent from each other, [6] and denote by π_i the steady-state probability of the service queue length being i.

Probe Queue (PQ). The probe queue is modelled as an M/G/1 queue. A job enters the queue only when the length of the service queue (SQ) is higher than Θ, and it leaves the queue when the probe reply is received. The probability that the node probes another node is clearly:

$$p_p = \sum_{i>\Theta} \pi_i \tag{1}$$

The *mean* service time of the queue is $T_{PR} + \tau_P$, so that the traffic intensity for this queue is:

$$\rho_P = \lambda p_p (\tau_P + T_{PR})$$

The variance of this service time is σ^2_{PR} (see later). The mean time spent by message in this queue is computed from the Pollaczek-Khinchin mean formula, [24]:

$$T_P = \frac{1 + C_s^2}{2} \frac{\rho_P}{1 - \rho_P} (\tau_P + T_{PR}) \tag{2}$$

where $C_s^2 = \frac{\sigma^2_{PR}}{T_P^2}$ is the squared coefficient of the service time.

Probe-Reply Queue (PRQ). This queue is modelled as an M/D/1 queue. Because the transmission time of a probe message is τ_P, the traffic intensity towards this queue is:

$$\rho_{PR} = \lambda p_p \tau_P$$

The mean time spent by message in this queue is derived from the well known M/D/1 formula:

$$T_{PR} = \frac{\tau_P}{2}\frac{2-\rho_{PR}}{(1-\rho_{PR})} \tag{3}$$

The variance of the waiting time for this queue is: $\sigma^2_{PR} = \frac{\rho_{PR}\tau_p^2}{3(1-\rho_{PR})} + \frac{\rho^2_{PR}\tau_p^2}{4(1-\rho_{PR})^2}$, [24].

Forward Queue (FQ). The forward queue is also modeled as an M/D/1 queue. A job enters the queue only when the service queue length is higher Θ *and* the state reported by the probed node is lower than the current state, so that the probability that a job enters the queue is:

$$p_F = \sum_{i>\Theta}\sum_{j<i}\pi_i\pi_j \tag{4}$$

Similarly to the previous queue, the delay due to job forwarding is:

$$T_F = \frac{\tau_J}{2}\frac{2-\rho_J}{(1-\rho_J)} \tag{5}$$

where $\rho_J = \lambda p_F \tau_J$.

Service Queue. To study this queue we take a different approach considering $N \to \infty$ nodes, and using a deterministic fluid flow model. This approach has been successfully applied to other studies on load balancing, [6].

Let $P_{ij}(0,t)$ be the fraction of service queues in the system that at time $t = 0$ have length i and at time t have length j, $q_{ij}(t)$ the rate at time t at which the length of a queue changes from i to j , and $q_{jj}(t)$ the rate at time t at which it changes from j. The dynamic of these nodes is described through the set of equations, [10]:

$$\frac{dP_{ij}(0,t)}{dt} = -P_{ij}(0,t)q_{jj}(t) + \sum_{k\neq j} P_{ik}(0,t)q_{kj}(t)$$

The equations measure the rate at which the population of nodes change their state. We now specialise the above equations for jobs arriving to nodes according to independent Poisson processes with rate λ, exponentially distributed amount of service time with mean $\frac{1}{\mu}$ and finite queue capacity K. The only rates that are not zero are: $q_{jj} = \lambda_j + \mu$, $q_{jj+1} = \lambda_j$, $q_{jj-1} = \mu$. Hence:

$$\frac{dP_{ij}(0,t)}{dt} = -P_{ij}(0,t)[\lambda_j + \mu] + P_{ij+1}(0,t)\mu + P_{ij-1}(0,t)\lambda_{j-1} \quad 1 \leq j < K$$

$$\frac{dP_{i0}(0,t)}{dt} = -P_{i0}(0,t)\lambda_0 + P_{i1}(0,t)\mu$$
$$\frac{dP_{iK}(0,t)}{dt} = -P_{iK}(0,t)\mu + P_{iK-1}(0,t)\lambda_{K-1}$$

This set of equations can be given in the following matrix form due to Chapman-Kolmogorov, e.g. see [25]:

$$\frac{d\mathbf{P}(0,t)}{dt} = \mathbf{Q}\mathbf{P}(0,t)$$

where $\mathbf{Q}$ is the following $(K+1) \times (K+1)$ (infinitesimal generator) matrix:

$$\mathbf{Q} = \begin{bmatrix} -\lambda_0 & \lambda_0 & 0 & \dots & 0 \\ \mu & -\mu - \lambda_1 & \lambda_1 & \dots & 0 \\ 0 & \mu & -\mu - \lambda_2 & \dots & 0 \\ & & \ddots & & \\ 0 & 0 & 0 & \dots & -\mu \end{bmatrix}$$

Formally, the solution of this equation with initial condition $\mathbf{P}(0,0) = \mathbf{I}$ is:

$$\mathbf{P}(0,t) = e^{\mathbf{Q}t}$$

The challenging part to apply this equation is that the arrival rates λ_i that appear in the matrix depend on the time t. As a workaround we use a constant value for the rates to obtain an approximation of the real values of the matrix. Let focus of the node B probed at random by A (see Figure 3) and suppose that the time zero corresponds to the time when B samples its state, say i, i.e.. $t_{PR} = 0$. For the sake of simplicity we omit the symbol zero from the element of the matrix $\mathbf{P}$. The probability that after a time lag τ the state of B changes from i to j is $P_{ij}(\tau)$ because this is also the fraction of nodes that change their state from i to j and $P_{ij}(\tau)$ can be interpreted at the probability that B belongs to this fraction. The rate at which jobs find node B in state j, given that it announced i, is then:

$$\lambda_j^F(\tau) = \frac{1}{\pi_j}\lambda \sum_{i=0}^{K} \pi_i \tilde{\pi}'_{i+1} P_{ij}(\tau) \tag{6}$$

Equation (6) reflects the probabilities of the following events: (i) node B sends a reply message to the probe message reporting state i (which occurs with probability π_i), (ii) the state of A was $k \geq i+1$ (occurring with probability $\tilde{\pi}'_{i+1}$) and

(iii) during τ time units the state of B changed from i to j. These probabilities are conditioned to the event of B being in state j. The rates in the matrix $\mathbf{Q}$ are then:

$$\lambda_j(\tau) = \begin{cases} \lambda_j^F(\tau) + \lambda & \text{if } j \leq \Theta \\ \lambda_j^F(\tau) + \lambda\tilde{\pi}_j & \text{otherwise} \end{cases} \tag{7}$$

Indeed, node B receives jobs from nodes like A (first term), plus all the jobs coming from its users (when the state is $j \leq \Theta$), or from users if the state of the probed node was worst than j ($j > \Theta$). Without loss of generality, from now on we assume death rate $\mu = 1$.

4.1.2. Model solution.

To find the steady-state of the above set of queues we use a fixed point algorithm divided into two steps. The first step calculates the $\mathbf{Q}$ of the service queue for a given fixed τ, while the second step calculates the waiting times of all the three queues based on the steady-state of the service queue. Initially, $\tau = \tau_p + \tau_J$ that represents the minimum delay needed to schedule a job from node A to node B.

STEP 1. Given a value τ, first the matrix $\mathbf{Q}_0$ where $\lambda_i = \lambda$ is created. Then, using this matrix, the vector π of the steady-state probabilities is computed. From these values, the rates of Equation (7) are computed, and they are used to define another matrix $\mathbf{Q}_1$. This procedure is repeated until the numerical convergence to a matrix $\mathbf{Q}^*$.

STEP 2. This step uses $\mathbf{Q}^*$ to compute the steady-state probabilities of the service queue and from here, the waiting times associated with the probe, probe reply and forward queues from Equation (2) Equation (5) and Equation (3), which allows determining a new value, say τ'. A new STEP 1 is then executed to find a new matrix $\mathbf{Q}^*(\tau')$. This second step also monitors the distance between two successive matrix passed from STEP 1 and halts the computation if the value is lower than ϵ.

4.1.3. Model convergence

In our numerical examples, the above procedure converged in less than 500 steps for $\epsilon = 10^{-5}$. Figure 4 compares the result from the model and discrete event simulations.

4.2. A model for the sequential forwarding protocols

The model for sequential forwarding is easier since a job is forwarded blinding to another node and the schedule lag has τ has no effects. Recall that jobs are

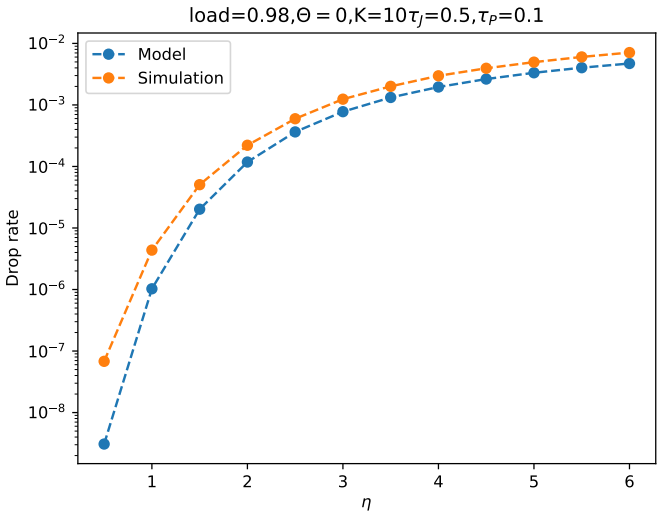


Figure 4: Probe based model validation

served by a node only if the current service queue length is at most Θ or the job was already forwarded M times. The rate of jobs that are forwarded only one time is $\lambda\tilde{\pi}_{\Theta+1}$, those that are forwarded two times is $\lambda\tilde{\pi}^2_{\Theta+1}$, etc. The total rate of jobs arriving to a node from other nodes is then:

$$\lambda_{SF} = \lambda \sum_{m=1}^{M} \tilde{\pi}^m_{\Theta+1} = \lambda\Big(\frac{1-\tilde{\pi}^{M+1}_{\Theta+1}}{1-\tilde{\pi}_{\Theta+1}} - 1\Big)$$

Since a job can find the landing queue at any state, the arrival rate of a generic queue is:

$$\lambda_j = \begin{cases} \lambda + \lambda_{SF} & \text{if } j \leq \Theta \\ \lambda_{SF} & \text{otherwise} \end{cases}$$

As before, to model job forwarding, we assume that a job is moved to a forwarding queue, modelled as an M/D/1 queue with $\rho_{SF} = \lambda_{SF}\tau_J$ and service time τ_J. Accordingly, the average waiting time is:

$$T_{SF} = \frac{\rho_{SF}}{2(1-\rho_{SF})} \tag{8}$$

4.3. Performance Metrics

4.3.1. Dropping rate

The drop rate is defined as the probability to drop a job. For the probe-based algorithm, it is given by:

$$P_B = \pi_K^2 + \sum_{i=\Theta}^{K-1} \tilde{\pi}_{i+1} \pi_i P_{iK}(\tau) \tag{9}$$

because a job is dropped if: (i) the receiving node is full but the job cannot be forwarded since the receiving node reports it is full as well (first term), or (ii) the job is forwarded, but during the time τ the target node becomes full (the job finds the node in state K) and drops the job.

For sequential forwarding:

$$P_B = \tilde{\pi}_{\Theta+1}^M \pi_K \tag{10}$$

which reflects the fact that a job is forwarded towards a congested node, i.e. whose state is K.

4.3.2. Response time

The *Response time* T_{Resp} of a job is defined as the time elapsed from when a job is received from a node (time t_p of Figure 3) until its service ends, t_E. This delay can be conveniently expressed as the sum of two contributions: the balancer time T_{Bal} due to move a job into a queue and the proper service time:

$$T_{Resp} = T_{Bal} + T_S$$

From Figure 3 the balancer time is the weighted sum $T_{Bal} = p_P T_P + p_F T_F$, see Equation (1), Equation (4), Equation (5) and Equation (3). The second term is determined by applying the Little's result to the service queue:

$$T_S = \frac{\sum_i i\pi_i}{\lambda(1 - p_B)}$$

For sequential forwarding:

$$T_{Resp} = T_S + \sum_{m=0}^{M-1} T_{SF}(1 - \tilde{\pi}_{\Theta+1})\tilde{\pi}_{\Theta+1}^m = T_S + T_{SF}(1 - \tilde{\pi}_{\Theta+1}^M)$$

where T_S is derived from the Little's result using Equation (10), while T_{SF} is computed in Equation (8).

Figure 5 shows the response time for the probe-based algorithm and sequential forwarding algorithms that reveal a good match between model prediction and simulations. A deep explanation of this shape is given in the simulation section.

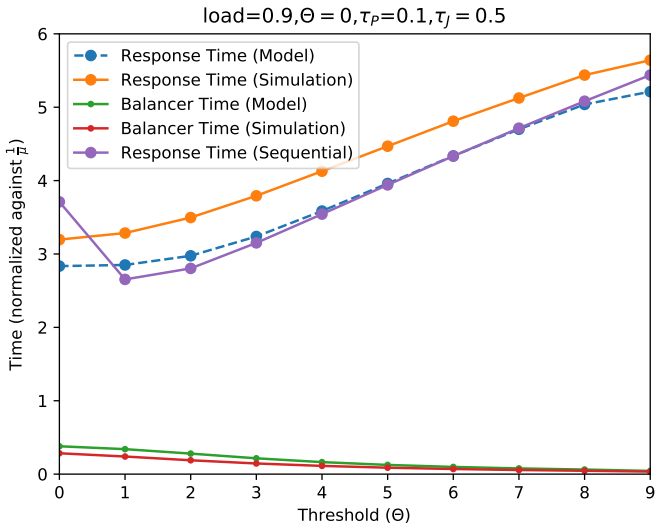


Figure 5: Response and Balancer time vs threshold Θ

4.4. Performance detriment and critical value of the schedule lag

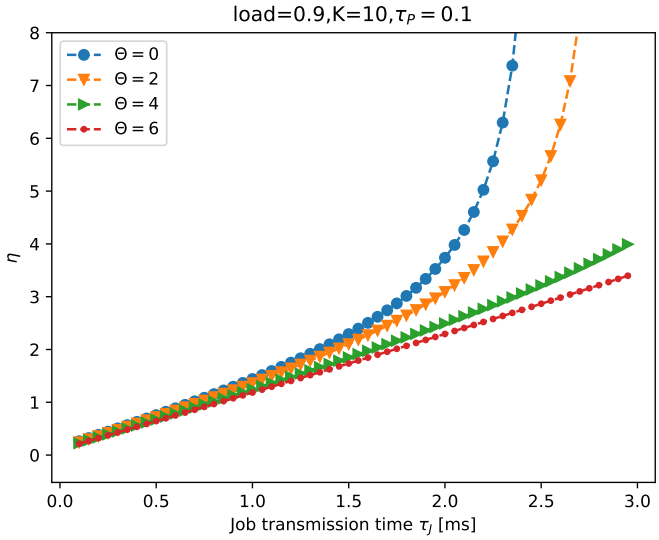


Figure 6: Normalized schedule lag vs job size.

With the model in hand we can now address the main question: how long the schedule lag can be before the detriment of the algorithm becomes prominent?

Before that, it is convenient to use a new quantity η which is the schedule lag normalised respect to the service time, $\eta = \tau\mu$ and that permits to set $\mu = 1$ in the numerical solutions. It it is also important to realized that the main source for a high η is the need to transfer long jobs. Figure 6 shows in fact how η increases with the size of the job that is forwarded for the probe-based protocol. A similar shape is found for the sequential forwarding algorithm. Clearly, because a lower threshold implies a less number of jobs that are forwarded by the Forward Queue, η decreases with the threshold.

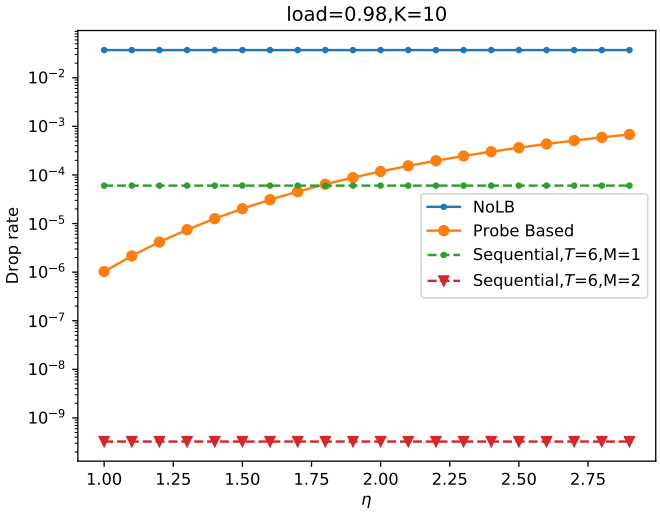


Figure 7: Drop probability vs η. For threshold of sequential forwarding minimizes the drop rate.

Figure 7 shows the drop rate for the probe based protocol and for the sequential forwarding with $M = 1, 2$. The drop rate of sequential forwarding is not affected by η, while it increases for the probe-based protocol. The value $\eta^* \approx 1.8$ is a cross point, after which the probe-based algorithm performs worst than the simpler sequential forwarding, i.e. its dropping rate is higher. The following table provides some example of critical values of the schedule lag for $M = 1$.

	K=6	K=8	K=10
0.9	1.90	3.4	4.9
0.95	0.69	2.0	3.5
0.98	0.13	0.54	1.8

Table 2: Examples of critical schedule lag

The Response time for the same setting is reported in Figure 8 and clearly it increases with M. In the simulation section we will discuss a way to limit this issue.

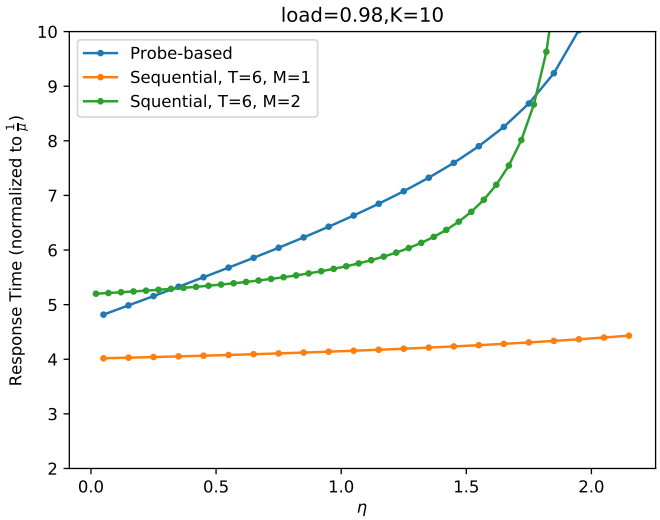


Figure 8: Total delay vs job transmission time for the sequential forwarding and the probe-based algorithms.

5. Simulation results

To provide an additional performance evaluation of the proposed load balancing algorithm, taking into account additional parameters and scenarios, we rely on a discrete event simulator based on the Omnet++ framework[2]. The load balancing algorithms are implemented in an additional module specifically developed. A specific additional module implements a dummy load balancing, namely *NoLB*, that processes locally every received request.

In our analysis, we take advantage of the insight provided by the model described in Sec. 4 and we aim to capture the impact of the network delay on the load balancing effectiveness, pointing out under which conditions each considered load balancing option is preferable.

[2]https://omnetpp.org/

5.1. Experimental setup

In our experiments, we consider the probe-based and the sequential forwarding algorithms. Furthermore, we consider also the *NoLB* dummy algorithm that does not perform any load balancing.

Throughout our experiments, we consider that in the probe-based algorithm, the query/response for probing incurs into a delay that we quantify as T_P, while the job forwarding introduces an additional delay T_F, as shown in Figure 3. This delay accounts for the schedule lag discussed in the model. On the other hand, the sequential forwarding algorithm requires a delay that is just T_F; but the forwarding can occur up to M times looking for a randomly-selected neighbour that is not overloaded. In the NoLB, no forwarding and no probing occurs.

In all our experiments, we model the nodes of our infrastructure as servers with an M/G/1/K queue. The incoming load is represented as a stream of incoming jobs with an inter-arrival time exponentially distributed. The service time is modelled according to a log-normal distribution, with a standard deviation that is comparable with the average value.

Finally, we consider that each node has a queue of finite size K. In our experiments, we set to $K = 9$. In this the system capacity is 9 jobs in the queue plus one job being executed way, that is consistent with the theoretical model of Sec. 4 (The system capacity is 9 jobs in the queue plus one job being executed). This assumption is consistent with application scenarios characterised by soft real-time requirements where long delays are not acceptable. The specific setting $K = 9$ is the result of an initial analysis where we explore the impact of the queue length on the algorithms' characteristics.

In our analysis we consider two scenarios, namely *uniform mesh*, and *geographic*.

The first scenario, namely *uniform mesh* consider a mesh of uniform nodes that have the same incoming load λ, and the same service rate μ. The inter-arrival time of jobs follows a Poisson distribution, while the service time is based on a log-normal distribution with a standard deviation comparable with the average value. In our experiments, we focus on an overall utilisation $\rho = 0.9$ of the infrastructure to capture the case where load balancing becomes a critical component of the infrastructure. We do not explicitly report all the timings of our experiments as we consider more general to provide normalised results with respect to the average service time $1/\mu$. We assume that the network introduces a delay normally distributed for both probing and job forwarding. Each probing phase (query and response) is characterised by an average delay equal to T_P, while the average job

forwarding delay is T_F. Throughout our performance evaluation, we consider different values for these delays. In particular, we provide a comprehensive sensitivity analysis on the impact of parameter η. We explore setups from $\eta \approx 0.1$, where schedule lag is significantly lower than the service time (for example, if the data to process is just an array of scalar values but significant mathematical analyses must be performed on these data), up to a case where $\eta \approx 10$ (for example if trivial computation must be carried out on large multimedia data). The intermediate case where schedule lag and service time are comparable is of particular interest in the area of edge computing and IoT because it is a common situation when data are transferred on long-range, low-power wireless links for edge-to-edge communication [26] and must be processed on low-end devices. Another analysis we carried out is evaluating the impact of the probe time compared to the job forwarding time. To this aim, we perform a sensitivity analysis to the parameter $\zeta = T_F/T_P$ where the job forwarding time ranges from $1\times$ to $10\times$ the probe time. This latter analysis is particularly interesting to understand under which circumstances the overhead probe-based approach becomes overwhelming, compared to the faster sequential-forwarding alternative.

In the *geographic scenario*, we focus on a more complex setup derived from a realistic topology based on an ongoing project of traffic sensing in Modena, a city in northern Italy of roughly 180'000 inhabitants. The sensors are located in the main city streets and collect information about the traffic (for example, taking pictures of the street when movement is sensed to count how many cars are passing). Fog nodes are placed in facilities belonging to the municipality and exchange data using long-range wireless links (such as IEEE 802.11ah/802.11af [26]) to interact both with the sensors and among themselves. The scenario description is generated using the PAFFI framework [27]. In these links, the available bandwidth decreases with the distance. Hence, we assume the delay of each link to be directly proportional to the distance between the two communication endpoints. In this scenario we consider also the impact of network congestion, considering that probe packets and jobs must queue before being sent to the neighbour node. Each sensor is connected to the closest fog node as in [28] so that the incoming load on each fog node is highly heterogeneous, ranging from cases where the incoming load is more than $3\times$ the processing capacity to cases where a fog node is nearly idle.

We summarise the main experimental parameters and performance metrics in Tab. 3 that expands the symbol list introduced in Tab. 1.

Symbol	Range	Explanation
		Scenario parameters
Θ	$[1, 10]$	Load Balancing activation threshold.
η_J	$[0.1, 10]$	service time. similar to η, without including probing $\eta_J = T_F\mu$; used to compare the algorithms.
ζ	$[1, 10]$	impact of the cooperation delay compared to job forwarding delay ($\zeta = T_F/T_P$).
		Performance metrics
P_B		*Drop rate*: probability of a job being discarded because the queue of the selected fog node is full.
T_{Resp}		*Response time* normalized against $1/\mu$.
T_{Srv}		*Service time*: time spent by jobs being processed; normalized to 1.
T_{Bal}		*Balancer time*: time taken for the load balancing jobs.
T_{Queue}		*Queuing time* that is the time spent in the fog node ready queue waiting to be processed; normalized against $1/\mu$.

Table 3: Summary of simulation parameters and metrics

5.2. *Uniform mesh scenario*

We start our analysis focusing on the uniform mesh scenario.

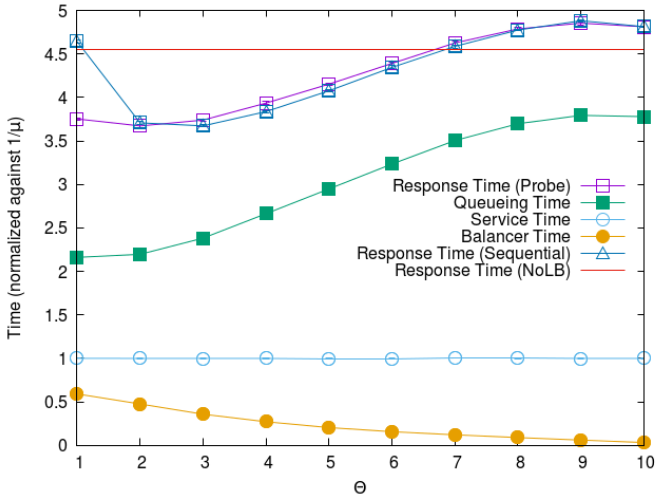
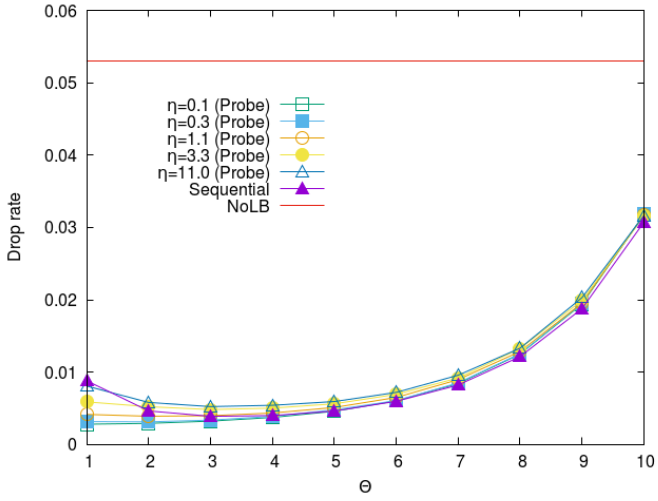
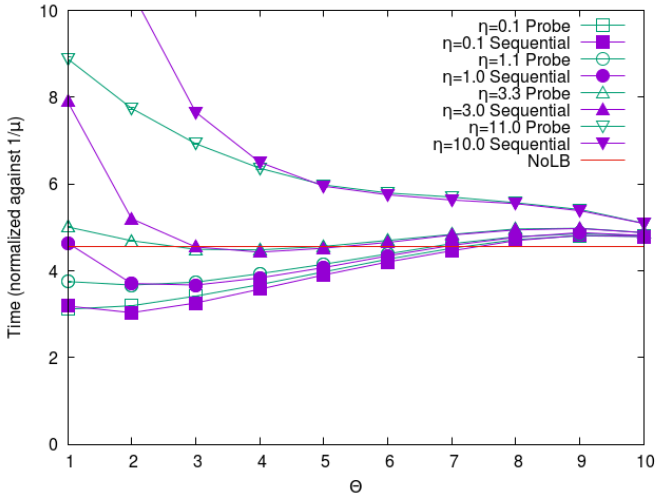


Figure 9: Response time *vs.* Θ

Figure 9 shows the response time of the probe-based and sequential forwarding algorithm for several values of the threshold Θ. The response time breakdown is provided to provide an insight into its components. Our results are related to the scenario where $\eta = 1.1$, $\zeta = 10$, for the probe-based algorithms, while for the sequential forwarding, we have $\eta = 1.0$ (the job forwarding delay T_F is the same, but in sequential forwarding, we do not have the probing contribution T_P). However, even if the example is referred to a specific scenario, the main findings have general validity and confirm previous observation for the impact of load balancing [8]: as the threshold increase, we observe a reduction of the time spent in the load balancing phase, due to a less frequent activation of the algorithm, at the expense of a higher queuing time, due to potential queue build-up. We also observe that the sequential forwarding algorithm response time has a similar shape but is characterised by a higher variance with respect to the threshold Θ, suggesting the need for careful tuning of this parameter. Finally, in the case where no load balancing occurs *NoLB*, we observe that the response time is generally higher compared with the alternatives. The response time is lower only for very high threshold values, where the load balancing algorithm seldom intervenes. This effect has already been observed in literature [8]



(a) Drop rate



(b) Response time

Figure 10: Sensitivity to η ($\zeta = 10$)

Figure10 provides a sensitivity analysis concerning the η parameter (the ratio between the schedule lag and the service time). In particular, Figure 10a shows the drop rate, while Figure 10b provides an analysis of the response time as a function of Θ for different values of η. In this analysis, we present our results just for the case where $\zeta = 10$, for space reasons.

Focusing on Figure 10a we observe that, especially for low values of Θ and for

low values of η (for example, in the curve marked with white squares), the probe-based algorithm outperforms the sequential forwarding alternative (reported as the curve with filled triangles). As Θ increases, the difference between the algorithms is reduced because the load balancing is activated less frequently and is, therefore, less effective. However, as η grows, the curve of the drop rate shifts from a monotone growing shape when $\eta = 0.1$ (meaning that the impact of delay is negligible) to a concave cup-shaped curve. This latter shape, which characterises the sequential forwarding algorithm, occurs when a job is sent to a randomly selected neighbour and is consistent with other results in literature [8]. This means that as the schedule lag grows, the load returned by the probing phase is less correlated with the load found on the node when the job is forwarded – ideally up to the point when the load encountered is completely unrelated to the probing result, reducing the probing to a random forwarding. This effect, already discussed in Sec. 4, is consistent with findings in other fields, such as the case of load balancing in Web servers [9]. The graph also shows the much higher drop rate that characterises the *NoLB* alternative: the fraction of dropped job is more than 5%, while the other load balancing solutions reach a drop rate typically below 1%.

Focusing on Figure 10b, we observe that the response time of the probe-based algorithm is generally lower compared with the sequential forwarding algorithm with the same threshold value, especially for low values of Θ when the load balancing is activated more frequently (again, when Θ grows, the difference between the two algorithms decreases). The case when η_J is very high (e.g. $\eta_J = 10$ – the curve with triangles) shows that load balancing is providing no actual benefit because the time to transfer the data is higher than the time to process them even with the queues are full (the longest wait is, on average, K/μ that becomes comparable with T_F). The *NoLB* solution (red line) guarantees a response time that is lower compared to the load balancing alternatives as long as $\eta_J > 1$, thanks also to the high drop rate. However, when the network delay is not so overwhelming, the benefit of load balancing is evident also from the response time point of view.

Figure 11 summarises the analysis on the impact of η over the load balancing performance. In the graph, we present a group of histograms for each value of η_J (in this graph we focus on η_J rather than on η because the former parameter has the same value for both algorithms for each application setup). For each considered value of η_J, we present the drop rate of the sequential forwarding and probe-based algorithms and the response time of the two algorithms measured for the threshold where the drop rate is minimum. We observe that for the sequential forwarding algorithm, the minimum drop rate remains stable with respect to the network delay, as expected. On the other hand, the drop rate of the probe-based

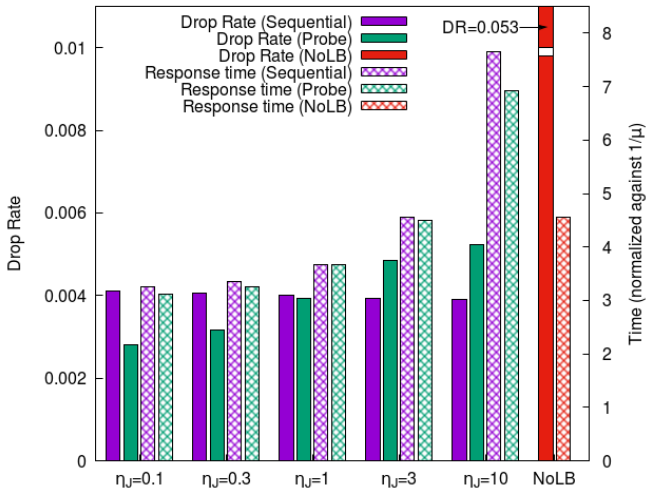


Figure 11: Algorithm comparison with respect to η_J

alternative increases. When the schedule lag becomes higher than the service time, the lowest achievable drop rate of the probe-base algorithm is worse than the sequential forwarding alternative. Considering the response time, we observe that the two algorithms have comparable performance unless the network delay is very high, in which case, the multiple load balancing hops in the sequential forwarding algorithm determine a clear performance penalty for this algorithm. The poor performance of the *NoLB* alternative are clearly visible. However, to make the figure more readable, the column of the drop rate is truncated. Indeed, the *NoLB* option is characterised by a drop rate that is more than $10\times$ compared with the load balancing algorithms.

Figure 12 provides further proof of the impact of the delay associated with the load balancing on the drop rate of the algorithm. For these analyses, we focus on the case where $\Theta = 1$, because this is the situation where the impact of load balancing is more evident. In particular, we consider two different measures of the balancing-related delay: the first is η; the second is the time spent in the load balancer T_{Bal} that is highly correlated with the schedule lag. Both measures provide consistent results, demonstrating that, as the delay increases, the drop rate grows, as suggested by the model in Sec. 4.

As the last sensitivity analysis for the uniform mesh scenario, we consider the impact of the ratio ζ between the time for job forwarding and the probing. In this analysis, we keep constant η_J, while η changes as we modify ζ. If we analyse the

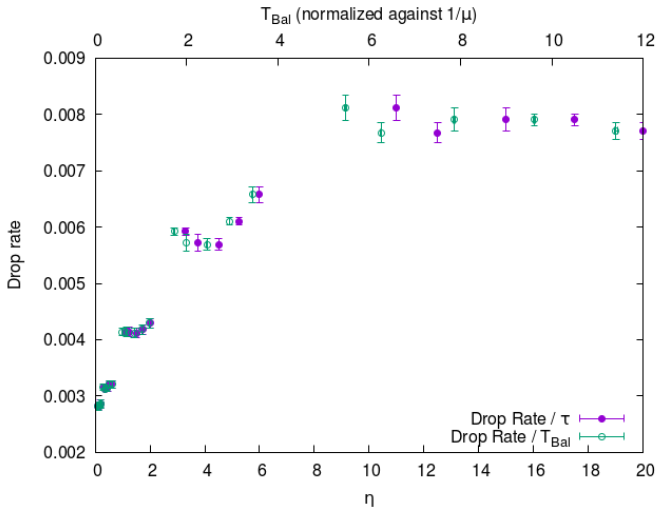
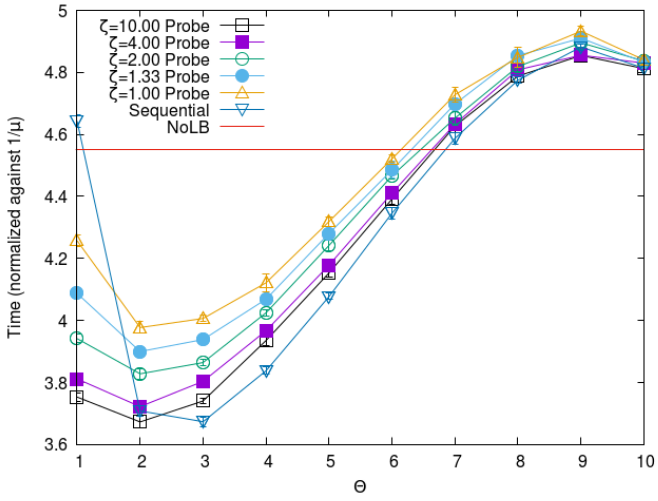


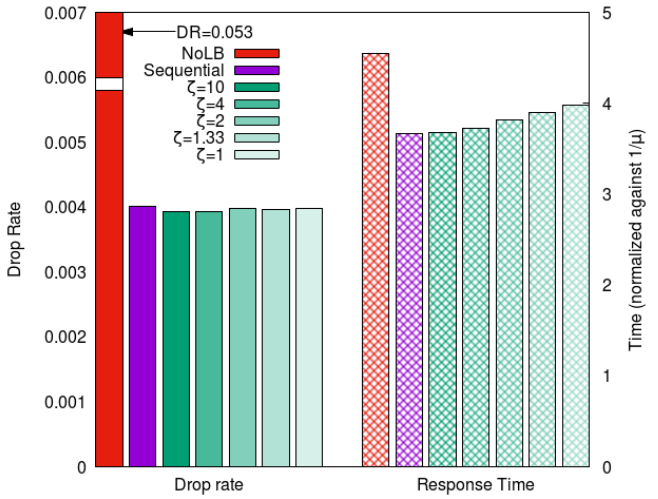
Figure 12: Schedule lag *vs.* Drop rate.

impact of ζ on the response time (Figure 13a) we observe that, as the probing time grows due to the reduction of ζ, the response time increases as well (for example from 0.038s to 0.043s, with an increment of 12% for $\Theta = 1$). Furthermore, as Θ grows, the reduction in the number of probes issued reduces the impact of the ζ parameter.

For a better comparison between the two considered algorithms, we present a histogram-based representation in Figure 13b, where the sequential forwarding is compared with several setups of the probe-based algorithm with different ζ. From the column on the left side of the graph, we observe that the drop rate remains unaffected by the slight increase in the network delay due to the probing overhead. On the other hand, in the right part of the graph, we show that, as the impact of the probing delay increases (ζ is reduced), the response time associated with the minimum drop rate grows accordingly. From this comparison, we can conclude that, when the probing time is comparable with the job forwarding time ($\zeta = 1$), which is common when the application is sending a limited amount of data to the edge nodes, we can expect a performance drop in response time in the order of 8% compared to a case where the time to transfer the data is $10\times$ compared to the probe. This effect should be further considered when selecting the most appropriate protocol for load balancing. We also report the performance of the *NoLB* solution, confirming its poor performance in terms of both drop rate and response time.



(a) Response time



(b) Algorithm comparison

Figure 13: Sensitivity to ζ ($\eta = 1$)

5.3. Geographic scenario

We now focus on the final scenario, where the role of load balancing is crucial. In this scenario some node receives an incoming load more that $3\times$ w.r.t. their processing capacity, while other nodes are nearly idle. Without load balancing the high overload in part of the infrastructure can lead to a drop rate up to 30% (again we omit in analysis the results for the case where no load balancing occurs due

to the overload conditions). This is the opposite situation compared to the mesh uniform scenario where load balancing must cope just with small temporary load fluctuations and, even in the worst conditions, the drop rate is below 5%.

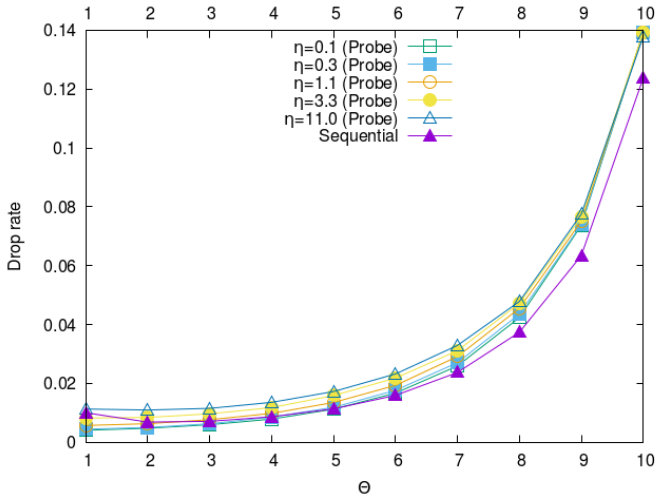


Figure 14: Drop rate *vs.* η_j ($\zeta = 10$)

Figure 14 provides the sensitivity analysis with respect to the η_J parameter. The analysis is similar to the one in Figure 10a. Even the presence of network effect, with the risk of congestion does not affect the general conclusions. However, we point out the main differences with the previously discussed mesh scenarios. It is interesting to observe that, in this highly skewed workload, reducing the intervention of the load balancing (for example, for $\Theta \geq 8$) determines a significant increase of the drop rate that rapidly grows beyond 5%. It is worth noting that only a small subset of the nodes experiences overloaded. Hence, only these nodes will issue probes very often. This reduces the loss of correlation in the state reported by probing and keeps the drop rate stable even when the threshold is very low (e.g., $\Theta = 1$), explaining the monotonic shape of the drop rate curve. This is a significant difference with respect to the previously considered scenarios, suggesting that, in the case of localised hot-spots, the problem of stale load information previously observed is much less critical.

We conclude our analysis with the algorithm comparison of the best drop rate and for the response time when the drop rate is minimum, as in the previous analyses. Again, we refer to the η_J parameter for this study, and we consider two extreme values of ζ that are 10 (low probe impact) and 1 (high probe impact),

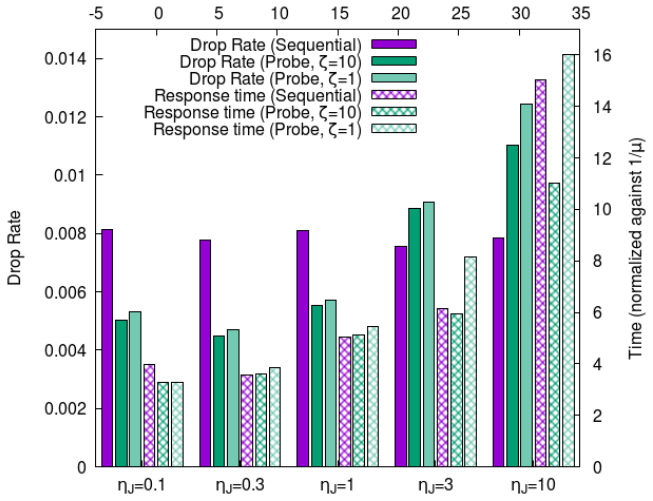


Figure 15: Algorithm comparison for η_J and ζ

respectively. Once again, as in this last set of experiments we aim to represent a scenario as realistic as possible, we consider that the network delay in fog-to-fog communication is caused by a bandwidth-constrained link, that can be subject to congestion.

In Figure 15 we observe that the minimum achievable drop rate of the sequential forwarding algorithm remains basically stable with the increasing network delay. On the other hand, for the probe-based algorithm, the delay has the already-proven negative impact on the drop rate. The ζ parameter has a limited effect on the drop rate, with the additional delay increasing slightly the drop rate due to its additional effect of the overall load balancing delay.

If we focus on the response time, we observe that the impact of ζ is much more significant: as the time for load balancing becomes comparable with the time for job forwarding, the delay experienced during the load balancing phase makes the probe-based algorithm slower than the sequential forwarding alternative. The effect is even worse when the network delay is higher than the service time. Further increasing the ζ and the η parameters would result in severe network congestion that can affect negatively both response time and drop rate.

6. Conclusions and future work

In this paper, we focused on the load balancing issue of distributing incoming jobs over the nodes of an edge computing infrastructure. Specifically, we

analysed the impact of stale load information caused by network latency on the effectiveness of load balancing algorithms based on randomisation of jobs over the nodes, showing that when this latency is comparable with the service time, the algorithm performs poorly. We carried out our study under two different points of view: a mathematical model and a full-fledged simulator. Our analysis revealed that taking schedule decisions based on state information received even with a small delay compared to the service time reduces the load balancing effectiveness considerably. In this setting, it is convenient to keep the randomisation principle incorporating it as a blind forward towards neighbouring nodes. The addition of a threshold to regulate the triggering of the algorithm is a valid method to reach high performance.

References

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture 98 (2019) 289 – 330. doi:https://doi.org/10.1016/j.sysarc.2019.02.009.

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

[3] M. Satyanarayanan, W. Gao, B. Lucia, The computing landscape of the 21st century, in: Proc. of the 20th International Workshop on Mobile Computing Systems and Applications, HotMobile '19, 2019, p. 45–50.

[4] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, P. A. Polakos, A comprehensive survey on fog computing: Stateof-the-art and research challenges, in: IEEE Communications Surveys, IEEE, 2017.

[5] P. Varshney, Y. Simmhan, Characterizing application scheduling on edge, fog, and cloud computing resources, Software: Practice and Experience 50 (5) (2020) 558–595. doi:https://doi.org/10.1002/spe.2699.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2699

[6] M. Mitzenmacher, The power of two choices in randomized load balancing, IEEE Transactions on Parallel and Distributed Systems 12 (10) (2001) 1094–1104.

[7] 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.

[8] R. Beraldi, C. Canali, R. Lancellotti, G. P. Mattia, Distributed load balancing for heterogeneous fog computing infrastructures in smart cities, Pervasive and Mobile Computing 67 (2020) 101221. doi:https://doi.org/10.1016/j.pmcj.2020.101221.

[9] M. Dahlin, Interpreting stale load information, IEEE Transactions on Parallel and Distributed Systems 11 (10) (2000) 1033–1047.

[10] M. Mitzenmacher, How useful is old information?, IEEE Transactions on Parallel and Distributed Systems 11 (1) (2000) 6–20.

[11] D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing, IEEE Transactions on Wireless Communications 11 (6) (2012) 1991–1995.

[12] J. Zheng, Y. Cai, Y. Wu, X. Shen, Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach, IEEE Transactions on Mobile Computing 18 (4) (2019) 771–786.

[13] Y. Wang, M. Sheng, X. Wang, L. Wang, J. Li, Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, IEEE Transactions on Communications 64 (10) (2016) 4268–4282.

[14] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Transactions on Networking 24 (5) (2016) 2795–2808.

[15] S. Wang, Y. Zhao, J. Xu, J. Yuan, C.-H. Hsu, Edge server placement in mobile edge computing, Journal of Parallel and Distributed Computing 127 (2019) 160–168.

[16] T. Ouyang, Z. Zhou, X. Chen, Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing, IEEE Journal on Selected Areas in Communications 36 (10) (2018).

[17] R. Beraldi, H. Alnuweiri, Exploiting power-of-choices for load balancing in fog computing, in: 2019 IEEE International Conference on Fog Computing (ICFC), IEEE, Piscataway, New Jersey, US, 2019, pp. 80–86.

[18] E. Eriksson, G. Dán, V. Fodor, Predictive distributed visual analysis for video in wireless sensor networks, IEEE Transactions on Mobile Computing 15 (7) (2016) 1743–1756.

[19] C. Long, Y. Cao, T. Jiang, Q. Zhang, Edge computing framework for cooperative video processing in multimedia iot systems, IEEE Transactions on Multimedia 20 (5) (2018) 1126–1139.

[20] Z. Zhou, H. Liao, B. Gu, K. M. S. Huq, S. Mumtaz, J. Rodriguez, Robust mobile crowd sensing: When deep learning meets edge computing, IEEE Network 32 (4) (2018) 54–60.

[21] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, Q. Zhang, Edge computing in iot-based manufacturing, IEEE Communications Magazine 56 (9) (2018) 103–109.

[22] R. Beraldi, C. Canali, R. Lancellotti, G. Proietti Mattia, Randomized load balancing under loosely correlated state information in fog computing, in: Proc. of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM, 2020.

[23] R. Beraldi, C. Canali, R. Lancellotti, G. P. Mattia, A random walk based load balancing algorithm for fog computing, in: The Fifth International Conference on Fog and Mobile Edge Computing (FMEC 2020), Paris, France, 2020, pp. 1–8.

[24] L. Kleinrock, Queueing Systems, vol 1: theory, Vol. John Wiley & Sons, Inc., 1975. doi:https://doi.org/10.1016/j.pmcj.2020.101221.

[25] E. U. Michael Mitzenmacher, Probability and computing: randomized algorithms and probabilistic analysis, CAMBRIDGE UNIVERSITY PRESS, The Edinburgh Building, Cambridge CB2 2RU, UK, 2005.

[26] 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.

[27] 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), Rome, Italy, 2019, pp. 1–8.

[28] 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.

[29] Q. Xie, X. Dong, Y. Lu, R. Srikant, Power of d choices for large-scale bin packing: A loss model, ACM SIGMETRICS Performance Evaluation Review 43 (1) (2015) 321–334.

[30] A. W. Richa, M. Mitzenmacher, R. Sitaraman, The power of two random choices: A survey of techniques and results, Combinatorial Optimization 9 (2001) 255–304.

Appendix A. Special cases

Appendix A.1. Completely correlated states

This condition occurs when communication is instantaneous , $\tau = 0$. In this case **P**(0) = **I**, i.e.:

$$P_{ij}(0) = \delta_{ij}$$

The state of the probed node cannot change wrt to the reported value, i.e., no state transition occurs. By substituting this value in Equ. (6) we get:

$$\lambda_j^F = \lambda \frac{1}{\pi_j} \sum_{i=0}^{K} \tilde{\pi}_{i+1} \pi_i \delta_{ij} = \lambda \frac{1}{\pi_j} \pi_j \tilde{\pi}'_{j+1} = \lambda \tilde{\pi}'_{j+1}$$

so that:

$$\lambda_j = \lambda \begin{cases} 1 + \tilde{\pi}_{\Theta+1} & \text{if } j \leq \Theta \\ \tilde{\pi}_j + \tilde{\pi}_{j+1} & \text{otherwise} \end{cases} \tag{A.1}$$

For $\Theta = 0$ the above equations describe the dynamic of the power of two random choices algorithm on a loss queue model [29]. Indeed, the generic balance equation of the MC associated to $\mathbf{Q}$ becomes (recall $\mu = 1$):

$$\lambda(\tilde{\pi}_j + \tilde{\pi}_{j+1})\pi_j = \pi_{j+1}$$

Since $(\tilde{\pi}_j + \tilde{\pi}_{j+1})(\tilde{\pi}_j - \tilde{\pi}_{j+1}) = (\tilde{\pi}_j^2 - \tilde{\pi}_{j+1}^2)$ and $\pi_{j+1} = \tilde{\pi}_{j+1} - \tilde{\pi}_{j+2}$ the equations can be rewritten in the so-called supermarket fluid model form (where conventionally $\tilde{\pi}_{K+1} = 0$) [30]:

$$\lambda(\tilde{\pi}_{j-1}^2 - \tilde{\pi}_j^2) = \tilde{\pi}_j - \tilde{\pi}_{j+1}$$

These equations have the following fluid flow interpretation. In a population of $N \to \infty$ nodes, any node sends its jobs to a central scheduler. The scheduler then sends the jobs to the least loaded among two random nodes. Here $\tilde{\pi}_j$ is interpreted as the *fraction* of nodes with at least j jobs en-queued.

Appendix A.2. Completely uncorrelated states

This case ideally corresponds to $\tau \to \infty$ because this ensures to observe the node in two random steady states.

$$P_{ij}(\infty) = \pi_j$$

hence:

$$\lambda_j^F = \frac{1}{\pi_j}\lambda \sum_{i=0}^{K} \tilde{\pi}'_{i+1}\pi_i\pi_j = \lambda \sum_{i=0}^{K} \tilde{\pi}'_{i+1}\pi_i = \\ = \lambda \sum_{i=0}^{\Theta} \tilde{\pi}_{\Theta+1}\pi_i + \lambda \sum_{i=\Theta+1}^{K} \tilde{\pi}_{i+1}\pi_i \tag{A.2}$$

so that:

$$\lambda_j = \begin{cases} \lambda + \lambda_j^F & \text{if } j \leq \Theta \\ \lambda\tilde{\pi}_j + \lambda_j^F & \text{otherwise} \end{cases} \tag{A.3}$$

Which are the same as sequential forwarding.

Appendix A.3. Correlation

The correlation between the values of the state of a node at probing time at the job's receiving time is measured using the Pearson's correlation coefficient:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

where X is a random variable representing the value of the state reported by a node when probed and Y the state of that node when it receives a job, where:

$$\sigma_X^2 = \sum Pr\{X = i\}(i - m_X)^2$$

$$\sigma_Y^2 = \sum_j Pr\{Y = j\}(j - m_Y)^2$$

and

$$\sigma_{XY} = \sum_j \sum_i Pr\{Y = j, X = i\}(i - m_X)(j - m_Y)$$

while:

$$m_X = \sum_i iPr\{X = i\} \quad m_Y = \sum_i iPr\{Y = i\}$$

We can compute the coefficient by recognizing that from our model we have:

$$Pr\{X = i\} = \mathbf{P}_{*i}(\infty)$$

and:

$$Pr\{Y = j | X = i\} = \mathbf{P}_{ij}(\tau)$$

so that we also get:

$$Pr\{Y = j\} = \sum_i Pr\{Y = j|X = i\}Pr\{X = i\}$$

$$= \sum_i \mathbf{P}_{ij}(\tau)\mathbf{P}_{*i}(\infty)$$

For $\tau \to \infty$ X and Y become independent which implies $\rho_{XY} = 0$.

Appendix A.3.1. Benefit of threshold

The idea of threshold can also be applied to the probe-based protocol as a way to reduce negative effect of the schedule lag.

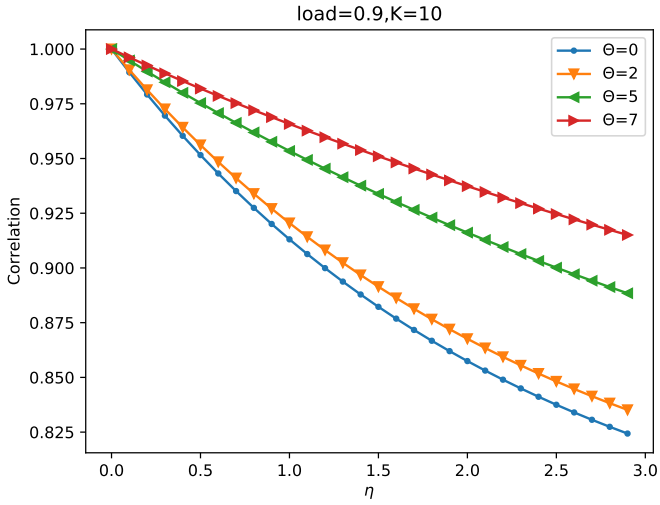


Figure A.16: Correlation vs normalized schedule lag for different thresholds.

Figure A.16 shows the correlation index as a function of Θ, when $\rho = 0.9, K = 10$. We can see how the correlation among states becomes weaker as η increases and as Θ decreases. Clearly, the weaker the correlation the higher the probability the scheduler takes a wrong decision, i.e. forwarding to remote node when the state of the remote node is indeed worst than the local one, or not forwarding to remote nodes when the state is better than the local one. Wrong decisions reduces the effectiveness of the load balancing performance.

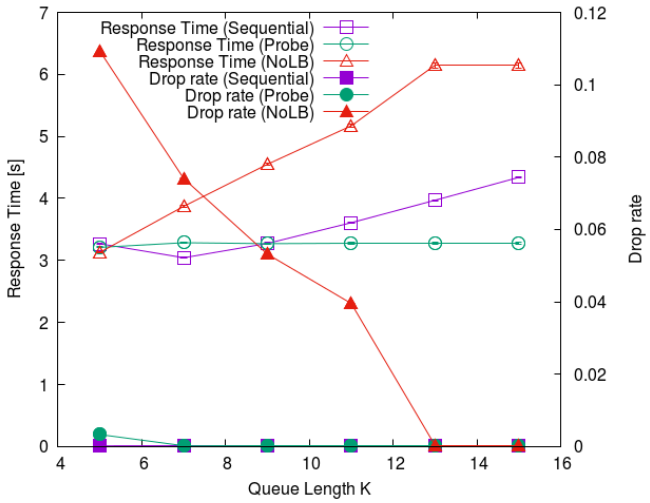


Figure B.17: Sensitivity to queue length K

Appendix B. Additional simulation results

Appendix B.1. Sensitivity to queue length

In our experiments, the finite queue length of each node K is to $K = 9$. This assumption is consistent with application scenarios characterised by soft real-time requirements where long delays are not acceptable. The specific setting $K = 9$ is the result of an analysis where we explore the impact of the queue length on the algorithms' characteristics. Specifically, we consider a scenario where fog nodes are connected in a regular, uniform, mesh. For the Probe-based algorithm, we use a threshold value $\Theta = 1$, while for the Sequential forwarding algorithm, we consider $\Theta = K/2$. Fig. B.17 shows the response time and drop rate as a function of K. We observe that the impact of the K parameter is quite limited on the drop rate, except for the case when the queue is very short (i.e., $K = 5$). Similarly, the impact of the queue size on the response time is almost negligible for the probe-based approach. The impact is slightly more evident for the sequential probing algorithm. However, in this case, the effect is mainly due to the increase of the threshold (that grows with K), as already pointed out in [8]. In the graph, we also show the case where no load balancing occurs among the fog nodes (the red set of curves labelled *NoLB*). In this case, it is evident that, as the queue grows, the response time grows (due to the higher potential waiting in the curve), and the drop rate grows (as the risk of finding the queue full is reduced). It is worth observing that the NoLB set of curves demonstrates that load balancing provides

a major benefit both in response time and drop rate.

Appendix B.2. Bimodal scenario description

In order to enrich our simulation analysis of the considered load balancing algorithms, we consider also a scenario, namely *bimodal mesh*. Like the *uniform mesh* sceanrio, the for nodes have the sam computing power and the network is a fully-connected unform mesh. However, we relax the assumption that the load is evenly distributed across the nodes, and we divide the nodes into two sets: 50% of the nodes is underloaded while 50% is overloaded. Specifically, let $\rho = 0.9$ be the average load of the infrastructure. The load on the first set of nodes is $\rho_L = \rho - \Delta\rho$, while it is $\rho_H = \rho + \Delta\rho$ for the remaining nodes. We create the different loads on the nodes altering their incoming job rate. For the bimodal mesh scenario, we perform the same sensitivity analyses previously described for the uniform scenario plus an analysis focusing on the impact of the $\Delta\rho$ parameter. However, we provide only a subset of relevant results for space reasons.

Appendix B.3. Simulation results

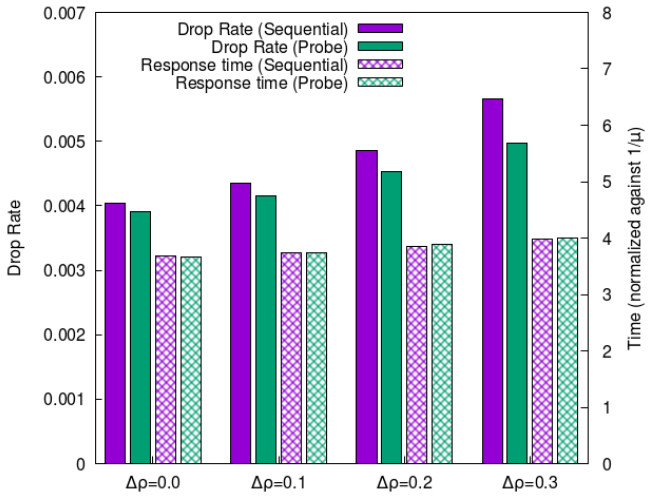


Figure B.18: Sensitivity to $\Delta\rho$ ($\eta_J = 1, \zeta = 10$)

In the previous scenario, the incoming load is uniformly distributed over the infrastructure, and the load balancing must address just temporary fluctuations that cause a local overload. The situation is more critical when the load is inherently unevenly distributed. The bimodal mesh scenario addresses this case.

In this experiment, we focus on the most interesting setup previously identified, that is, the case where the load balancing and the service times are comparable (i.e. $\eta \approx 1$) and when the probe is significantly smaller than the job forwarding (that is $\zeta = 10$).

Figure B.18 compares the two algorithms as $\Delta\rho$ ranges from 0, corresponding to the previously presented uniform scenario, to 0.3, corresponding to a clearly unbalanced workload where half the nodes of the infrastructure receive an incoming load $\lambda = 0.6\mu$, while the other half of the infrastructure is overloaded with $\lambda = 1.2\mu$.For this set of experiments, we omit the comparison with the case when no load balancing occurs (*NoLB*) as it would make no sense to consider scenarios where nodes are explicitly overloaded.

We observe that in the uniform scenario ($\Delta\rho = 0$), the drop rate of the two sequential forwarding and of the probe-based algorithms are similar (the drop rate is 0.0040 *vs.* 0.0039 with a relative difference of 3%). However, as the workload unbalancing increases, load balancing becomes harder (as testified by the drop rate increase). Furthermore, the probe-based algorithm is capable of a more effective load balancing, resulting in a lower drop rate (for $\Delta\rho = 0.3$ the drop rate is 0.0057 for the sequential forwarding and 0.0050 for probe-bases, with a relative difference of 12%).