

# One Edge to Serve Both: A System Model for Joint 6G User Plane and Service Function Allocation

Toni Dimitrovski<sup>1,2</sup>, Nassima Toumi<sup>1</sup>, Maria Raftopoulou<sup>1</sup>, Suzan Bayhan<sup>2</sup>, and Geert Heijenk<sup>2</sup>

<sup>1</sup>Networks Department, TNO

<sup>2</sup>Design and Analysis of Communication Systems, University of Twente

**Abstract**—In line with the growing cloudification, mobile networks have been increasingly moving their Network Functions (NFs) to cloud environments so they can unlock resource efficiency gains by shared use of the underlying physical infrastructure. Despite this potential, sharing resources among user plane NFs and Service Functions (SFs) may lead to unexpected performance degradation due to contention for the compute and networking resources, especially under variable load scenarios. The system models from current literature on NF placement assume deterministic, load-independent performance with strict resource partitioning and allocation. This simplification makes the models less accurate for flow level Service Level Agreement (SLA) evaluations because they do not account for traffic variability or resource load impact. Furthermore, the enforcement of strict resource partitioning and allocation is complex and impractical at scale. To mitigate these issues, we (i) propose removing the load-independence assumptions by using a scalable, queueing-based stochastic system model that accurately captures load-dependent performance and workload collocation effects in edge infrastructures, (ii) develop latency quantile approximations at the compute node level, and (iii) develop end-to-end latency quantile approximations across the core network. We validate the model and analyze its strengths and limitations in predicting network flow performance and guiding SLA-compliant workload allocation. Our analysis shows that the model predicts the end-to-end core network latency with a Mean Relative Error (MRE) of 0.1 and results in less than 1% of false positives and 13% of false negatives for flow level SLA constraint evaluation.

**Index Terms**—edge-computing, virtualization, function placement, performance model, resource efficiency

## I. INTRODUCTION

Edge Computing enables the deployment of services in proximity to end users, thereby offloading the core network, reducing core network latency, and enabling efficient resource management. The services traditionally deployed on centralized servers can thus be distributed over multiple edge servers [1] with 5G networks, using local resources in the amount dependent on requirements. The User Plane Function (UPF) of the 5G core network is the main candidate for distribution across edges, where multiple lightweight UPF instances can be deployed on edge servers to accommodate the current user demands and load [2], [3]. The next generation of networks (6G) is only expected to scale up this distribution and the requirements for in-network computing.

In this setting, it is advantageous for network operators to use the same compute infrastructure for both UPFs, which are the functions forwarding packets, and Service Functions (SFs),

which are the functions processing service information. Hence, UPFs would compete for resources with SFs. Although this approach provides benefits in terms of physical infrastructure costs, it poses a significant problem in the management of performance [4] and its associated trade-off with resource efficiency. Solving it would bring value to network operators, decreasing costs and simplifying operations while keeping Service Level Agreements (SLAs) fulfilled. However, the issue has not been adequately addressed, as UPFs are typically envisioned to run on purpose-made telco clouds, so their resource allocation is usually considered as a separate problem often solved by over-provisioning dedicated resources. This keeps the cost of networks high and decreases resource efficiency [5].

In order to optimally allocate UPFs and SFs to shared edge resources, it is essential to capture the dependency between resource load, resource contention and processing times. Queueing Theory (QT) models are well known abstractions of systems to estimate load-based performance. Taking the two aspects discussed above into consideration, we are interested in the following research question: What QT systems are applicable, if any, to improve the infrastructure models from current literature to incorporate the aspects of resource contention between UPFs and SFs and to capture the impact of load on the end-to-end core network latency? Our contributions towards answering this question are: (i) we use functional aspects of Central Processing Unit (CPU) packet forwarding and process scheduling on general purpose hardware and argue that it behaves as a two-queue cyclic service system with exhaustive k-limited service strategy and switchover times, (ii) we approximate the quantiles of the packet processing response time distribution on the CPU, and (iii) we express the end-to-end core network latency of service flows as a function of the quantiles on each node the flow traverses. To assess the accuracy of our model we use simulations and also discuss its performance with regards to optimal workload placements.

This work is organized as follows: Section II reviews the models used in current state-of-the-art. Section III defines the context of our problem and describes the considered service and physical infrastructure. Section IV proposes an infrastructure model with a focus on two resources of the computing infrastructure - CPU and Network Interface Card (NIC). Section V provides the evaluation of the model while Section VI concludes and discusses future work.

## II. STATE OF THE ART

The purpose of the system model proposed in this work is to predict latency performance which is used to find optimal function distribution. Similar infrastructure models in current literature are usually developed as part of a larger work focused of Network Function (NF) resource allocation problems. For example, the authors of [6] developed an algorithm called DPC-UPCR which modifies UPF placements from a Service Function Chain (SFC) in online scenarios. However, their infrastructure model explicitly excludes the SFs from the destination edge and assumes that it uses a different resource pool with guaranteed performance. It also uses static performance indicators for service and network processing, which makes it inapplicable for use in common cloud deployments as it requires separate, dedicated resources. Similarly, the authors of [7] used Deep Reinforcement Learning (DRL) to solve a Virtual Network Function (VNF) placement problem but also defined the performance of VNFs as a sum of load-independent latency terms.

In contrast to a more global view of the whole infrastructure, another approach is to perform local adjustments on the assignment of single machine resources to a UPF, treating it as an isolated problem per function. These types of solutions usually take an experimental approach and solve the problem using local scaling. For example, the authors of [8] and [9] propose dynamic algorithms for scaling out UPF instances depending on CPU consumption or traffic intensity. Although this type of scaling is often used in practice due to its simplicity, it leads to unexpected performance impacts to the latencies because it ignores the impacts of other workloads on the same resources.

There are also mixed approaches, with both global and local optimization processes. An example of this is [10], where the provided solution optimizes SFC deployments on an infrastructure globally by using Integer Linear Programming (ILP) and heuristics but also scales the resources assigned to the VNF based on local information. Although this work includes the SF at the end of the SFC on the same edge resources, the UPF and SF performance indicators in the infrastructure model are load-independent.

To summarize, all infrastructure models for resource allocation present in the literature, to the best of our knowledge, express the performance of VNFs and SFs as deterministic and independent of underlying resource load. This simplifies the problem solving because it removes nonlinearities in the performance. Although these models are suitable for long term capacity planning, they are less practical for autonomous workload placement and not usable to estimate tail performance which is typically needed for ensuring SLA compliance in shared edge environments. Hence, a load-based performance model with resource sharing at the scale of an edge infrastructure is necessary.

## III. SYSTEM MODEL

In this section, we describe our service architecture and the underlying infrastructure to provide the service to the users.

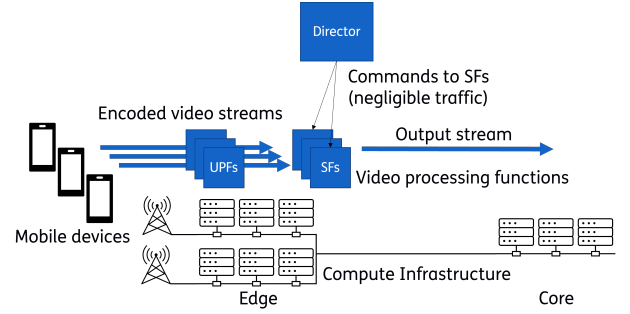


Fig. 1. System and service description for an event streaming use case.

### A. Service architecture

Let us consider the scenario depicted in Figure 1, where multiple devices are connected to a mobile network and stream live videos uplink from an event. Let us also assume that the mobile network hosts the service in its edge and the streams end in a local edge video processing center via one or more UPFs, with a given latency requirement. We focus our analysis on the encoded streams to the SFs and their distribution on the edge infrastructure. Aside from the service and its traffic, other devices on the network also generate background traffic directed towards the core network. The computing infrastructure also hosts other background workloads that share resources with the UPFs and SFs.

The underlying computing infrastructure comprises multiple small edge data centers with a wireless access point in the vicinity of the event, where UPFs are deployed on the edge servers to route packets towards the SFs. We assume all resources are shared among the deployed UPFs, SFs and background workloads, with no resource reservation for exclusive use. Furthermore, each edge is interconnected with other nearby edges, and the core network is accessible from all edges, with all background traffic directed towards it. In this environment, we focus on the end-to-end core network latency in the edge data centers and the impact of resource load and sharing between UPFs and other workloads on the latency.

### B. Mobile network edge infrastructure

We represent the considered infrastructure as a graph where each node represents a general purpose compute server from a small edge data center. Figure 2 shows an example with small edge data centers and one core data center interconnected with links in a given topology. Specific nodes from each edge have the interconnection links to the core and between the edges.

Each node in the topology can perform processing tasks using the CPU resource and packet forwarding tasks using both CPU for header processing and the NIC resource for forwarding the packet to the next node en route to the destination node. We assume that each node has multiple CPU cores and one outgoing NIC. Since our main bottlenecks are the CPU and the outgoing NIC, these are the resources we focus on in this work and omit the incoming NIC.

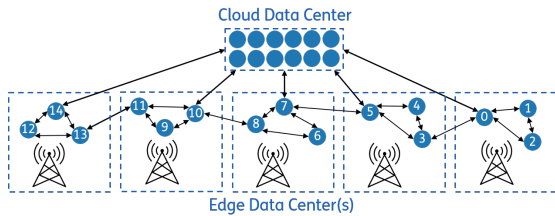


Fig. 2. An example of the considered infrastructure topology.

## IV. MODELING APPROACH

Network operators typically consider percentiles of packet latency for their SLAs (e.g., 95% of the packets experience latencies below the value). Hence, we focus on the higher percentiles of network latencies in the core rather than the mean. In this section we first detail the representation of each node from the infrastructure, the associated routing and the traffic model. Then we discuss the network packet latency distribution on a single node considering both the CPU and NIC resources and provide an approximation of the higher quantiles for this distribution. Finally, using the single node higher quantiles and the fact that the end-to-end core network latency distribution is a phase-type distribution, we approximate its higher quantiles. Table I lists the most relevant notations used in the rest of the paper.

TABLE I  
SELECTED LIST OF NOTATIONS

| Symbol                      | Description                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------------------|
| $\lambda_f$                 | Mean packet arrival rate (Poisson) per flow                                                     |
| $\gamma_n, \alpha_n, \nu_1$ | Mean packet arrival rate (Poisson) at the CPU                                                   |
| $\omega_n$                  | Mean packet arrival rate (Poisson) at the NIC                                                   |
| $\beta_n, \delta_n, \nu_2$  | Mean job arrival rate (Poisson) at the CPU                                                      |
| $\mu_{NET}$                 | Mean service rate (Poisson) of packets at the CPU                                               |
| $\mu_{JOB}$                 | Mean service rate (Poisson) of jobs at the CPU                                                  |
| $\mu_{NIC}$                 | Mean service rate (Poisson) of packets at the NIC                                               |
| $k_1$                       | Maximum number of packets served per cycle at the CPU                                           |
| $k_2$                       | Maximum number of jobs served per cycle at the CPU                                              |
| $\rho_1$                    | System utilization of packet processing at the CPU                                              |
| $\rho_2$                    | System utilization of job processing at the CPU                                                 |
| $\rho$                      | Total system utilization at the CPU                                                             |
| $\sigma$                    | Mean switchover time between forwarding packets and processing jobs (exponentially distributed) |

### A. Infrastructure Node

To calculate the latency a service will experience from the source to the destination node, we need to account for the latency at each node en route. This has two components if the destination is not reached - CPU and NIC. Let us first describe the packet forwarding operation on modern CPUs. We refer to jobs as chunks of service processing (e.g. an SF running on the CPU) or background workload processing as discussed in Section III-A. Packet processing is the forwarding of packets for the service (e.g. UPF running on the CPU) or the background traffic towards the core. Each CPU core forwards packets within a single process in NAPI [11] cycles and alternates between this process and other jobs. The time to process a packet varies depending on the CPU

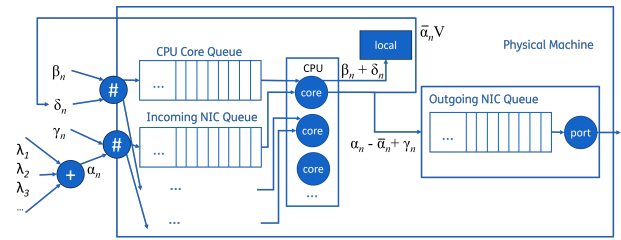


Fig. 3. Infrastructure node.

pipelining and memory access structure. In a single cycle, a maximum of  $k$  packets are processed by the core after which it switches over to other jobs. This job then uses the core and executes instructions until it either finishes/blocks and yields or it reaches a maximum execution time after which it is forced to yield. The switchover between the two processes incurs a time penalty which varies depending on the state the core was in. However, if we assume these are three random independent processes (packet service, job service and switchover) with exponentially distributed times, this behavior is similar to the cyclic service system with exhaustive k-limited service strategy and switchover times studied in [12]. Thus, we model each CPU core as two-queue k-limited polling model with exponential service and switchover times (depicted on Figure 3), one queue for the packets (Incoming NIC Queue) and another for the jobs (CPU Core Queue). With this, we assume that the input arrivals of jobs and packets are Poisson processes, equally split over all the cores.

Next to the CPU, we represent the outgoing NIC as a simple server queue. Since the input process at the NIC is a superposition of all CPU core outputs, we also assume that to be a Poisson process, which together with the assumption that packet sizes are exponentially distributed allows us to use the M/M/1 system model.

### B. Traffic Model

Considering the infrastructure topology in Figure 2, we have a set  $\mathcal{N}$  of nodes and a set  $\mathcal{F}$  of flows, where  $N = |\mathcal{N}|$  and  $F = |\mathcal{F}|$  are the number of nodes and flows in the network, respectively. We denote the starting nodes of the flows with  $\mathbf{s} = [s_0, \dots, s_{F-1}] \in [0, N-1]^F$ , where  $s_f$  is the starting node of flow  $f \in \{0, F-1\}$ . The ending node of each flow is represented by a binary variable matrix denoted as  $\mathbf{Y} = [\mathbf{y}_0, \dots, \mathbf{y}_{N-1}] \in \{0, 1\}^{F \times N}$ , where  $\mathbf{y}_n = [y_{n,0}, \dots, y_{n,F-1}] \in \{0, 1\}^F$  indicates which flows are ending at node  $n$ . In particular,  $y_{f,n} = 1$  when flow  $f$  ends in node  $n$ , and  $y_{f,n} = 0$ , otherwise. For the routing of a flow  $f$  from its start node  $s_f$  to its ending node, different nodes are traversed. This is fixed and given by the routing tensor  $\mathbf{R} \in [0, N-1]^{N^3}$ :

$$\mathbf{R} = \begin{bmatrix} \mathbf{v}_{0,0} & \dots & \mathbf{v}_{0,N-1} \\ \dots & \dots & \dots \\ \mathbf{v}_{N-1,0} & \dots & \mathbf{v}_{N-1,N-1} \end{bmatrix},$$

where vector  $\mathbf{v}_{n,n+1} = [r_{n,n+1,0}, \dots, r_{n,n+1,N-1}] \in \{0,1\}^N$  indicates the nodes that need to be traversed for communication between node  $n$  and  $n+1$ . Specifically,  $r_{s_f,n,k} = 1$  when flow  $f$  starting from node  $s_f$  and ending at node  $n$  is traversing node  $k$ , and  $r_{s_f,n,k} = 0$ , otherwise. Note that the vectors  $\mathbf{v}_{n,n+1}$  do not indicate the sequence of traversal.

Let us denote with  $\lambda_f$  the average packet rate of flow  $f$ , with  $\beta_n$  the average job rate representing background CPU load, and with  $\gamma_n$  the average packet rate of the background network load (CPU and NIC) on node  $n$ . Depending on the distribution of flows across the network, part of the service traffic will pass both the CPU and NIC, whereas another part only passes the CPU and terminates locally, generating local service processing load. Denote with  $\alpha_n$  the average packet rate representing service traffic going in node  $n$ . This is equal to the sum of the average packet rates of all flows terminating in node  $n$  plus the sum of the average packet rates of all flows traversing node  $n$ , which we can represent as follows:

$$\alpha_n = \underbrace{\sum_{f=0}^F y_{f,n} \lambda_f}_{\text{Terminating at node n}} + \underbrace{\sum_{f=0}^F \sum_{j=0}^N y_{f,j} R_{s_f,j,n} \lambda_f}_{\text{Traversing node n}}. \quad (1)$$

The part of the service traffic terminating locally, denoted with  $\bar{\alpha}_n$ , is equal to the sum of the average packet rate of all flows  $f$  that terminate on this node:

$$\bar{\alpha}_n = \sum_{f=0}^F y_{f,n} \lambda_f. \quad (2)$$

Denote with  $\delta_n$  the average job rate (CPU load) on the node representing the service processing (SF) on that node, i.e. video decoding/transcoding. This will be equal to the sum of the packet rates of all the flows terminating locally times some constant  $V$  which represents the ratio between the service processing rate and packet processing rate. Consequently, we can represent  $\delta_n$  as follows:

$$\delta_n = \bar{\alpha}_n V. \quad (3)$$

### C. CPU and NIC response time distribution

Now that we defined the resource models for the CPU and NIC, let us discuss the quantiles of their response time distributions. Let  $\mu_{NET}$  and  $\mu_{NIC}$  be the mean service rates of network processing on the CPU core and the outgoing NIC respectively. Let  $\mu_{JOB}$  be the mean service rate of service processing on the CPU core. We first focus on the representation of the NIC resource, which is an M/M/1 queueing system with exponential response time distribution. If we take the arrival rate  $\omega_n = \alpha_n - \bar{\alpha}_n + \gamma_n$  and percentile  $P$ , we can define the response time quantile at the NIC as:

$$Q_{NIC}(P, \omega_n) = -\frac{\ln(1-P)}{\mu_{NIC} - \omega_n} = -\ln(1-P) E_{M/M/1}(\omega_n), \quad (4)$$

where  $E_{M/M/1}(\omega_n)$  is the mean response time for an M/M/1 system with  $\omega_n$  packets/s traffic arrival rate. Now, let us focus on the expected latency at the CPU. Since the CPU behaviour resembles a k-limited polling model, there is no closed-form of the cumulative distribution and the quantile function. Let  $Q_1$  be the queue of packets (Incoming NIC Queue in Figure 3) and  $Q_2$  be the queue of CPU jobs on the core (CPU Core Queue in Figure 3). As mentioned earlier, the mean service times of packets are many orders of magnitude smaller than those of jobs. Example values for packet forwarding times on an infrastructure like ours are  $(1-10)\mu s$  [13] whereas examples of job times are  $(100-1000)\mu s$  for short jobs finishing/pausing execution and yielding the CPU, and  $(4-20)ms$  for long jobs forced to yield by the CPU scheduler [14]. We represent all of these as exponentially distributed random variables with mean packet service time  $\tau_1 = 8.74\mu s$ , mean switchover time  $\sigma = 13.5\mu s$  [15] and mean job service time  $\tau_2 = 500\mu s$ . We also set the number of packets served in a cycle to  $k_1 = 300$ , which is a default value for a physical machine with the Linux operating system [16], and the number of jobs served in a cycle  $k_2 = 1$ , which means that after processing a job the machine always checks the network queue. With this choice of  $k_2$  we emulate the relative priority that network traffic has in many operating systems due to its interrupt driven nature. In this system, the response time distribution of packets from  $Q_1$  is a mixture of the packet waiting times, service times and the (residual of) inter-visit intervals of the core to serve jobs. This inter-visit interval is in turn a mixture of the switchover and the job service time.

The approximation of the mean response time [12]  $E_i$  for packets or jobs from queue  $Q_i$  when we have  $C$  cores and two queues with mean arrival rate of packets at the CPU core  $\nu_1 = \frac{\alpha+\gamma}{C}$ , and mean arrival rate of jobs at the CPU core  $\nu_2 = \frac{\beta+\delta}{C}$  is as follows:

$$E_i(\nu_1, \nu_2) = \frac{(1-\rho_i)(1-\rho)}{1-\rho-\frac{\nu_i\sigma}{k_i}} \frac{D + \frac{\sigma}{1-\rho} \sum_{j=1}^2 \frac{\rho_j^2}{k_j}}{\sum_{j=1}^2 [\rho_j(1-\rho_j) + \frac{\rho_j^2}{k_j} \frac{2-\rho}{1-\rho}]} + \tau_i. \quad (5)$$

Here

$$D = \rho \frac{\sum_{j=1}^2 \nu_j \tau_j^{(2)}}{2(1-\rho)} + \rho \frac{\sigma^{(2)}}{2\sigma} + \frac{\sigma}{2(1-\rho)} [\rho^2 - \sum_{j=1}^2 \rho_j^2],$$

$\tau_1 = \frac{1}{\mu_{NET}}$  is the mean service time of a packet on the CPU,  $\tau_2 = \frac{1}{\mu_{JOB}}$  is the mean service time of a job on the CPU. The switchover time  $\sigma$  is the time that the CPU spends to switch from serving packets to serving jobs.  $\tau_i^{(2)}$  and  $\sigma^{(2)}$  are the second moments of the service and switchover times respectively. Since we only deal with exponential distributions, those will be equal to  $\tau_1^{(2)} = \frac{2}{\mu_{NET}^2}$ ,  $\tau_2^{(2)} = \frac{2}{\mu_{JOB}^2}$  and  $\sigma^{(2)} = 2\sigma^2$ . Furthermore,  $\rho_1 = \nu_1 \tau_1 = (\frac{\alpha+\gamma}{C}) \tau_1$ ,  $\rho_2 = \nu_2 \tau_2 = (\frac{\beta+\delta}{C}) \tau_2$  and  $\rho = \rho_1 + \rho_2$ . Consequentially,  $E_1(\nu_1, \nu_2)$  is the mean response time of the packets traversing the CPU and  $E_2(\nu_1, \nu_2)$  is the mean response time of the jobs running

on the CPU. Since we only have an approximation of the mean response time  $E_1(\nu_1, \nu_2)$ , let us discuss the tail of the response time distribution of packets from  $Q_1$ . Although these arguments are in principle valid for any value of  $P > 0.95$ , we will focus the discussion on  $P = 0.99$  for low and high loads.

Under **low loads**, the system alternates between short, quiet periods where  $Q_2$  is empty (meaning quick inter-visit interval), and rare long streaks where  $Q_2$  is busy for a long period. This mixture will produce a very uneven inter-visit interval distribution with many short intervals and a few very long ones. A packet experiencing a long inter-visit interval waits for a switchover time, a long exponential service time on  $Q_2$ , another switchover time back to  $Q_1$  and possibly any packets before it in  $Q_1$ . Since in low loads we can neglect the packets before it,  $Q_{CPU}(P = 0.99)$  will be dominated only by these very long inter-visit intervals, which are on average  $2\sigma + \tau_2$  long. Hence, we can approximate the tail as a single rate from a hyper-exponential distribution with  $F_{CPU}(t) \approx 1 - p_1 e^{-\frac{t}{2\sigma + \tau_2}}$  because all the other rates do not contribute. If we replace  $F_{CPU}(t) = P$ , we can specify the quantile in low loads as:

$$P \approx 1 - p_2 e^{-\frac{t}{2\sigma + \tau_2}} \quad (6)$$

$$Q_{CPU}(P) \approx (2\sigma + \tau_2) \ln\left(\frac{p_2}{1 - P}\right),$$

where  $p_2$  is the probability that a packet will experience the long inter-visit interval while the CPU is serving  $Q_2$ .

As overall load increases, the server spends more of its time busy and the queue(s) are rarely empty, so the pattern “long streak vs. quiet periods” disappears and every visit interval includes some  $Q_2$  service.  $Q_1$  is also busy more often, so even when the server returns to  $Q_1$ , it finds a backlog there. The system gets more into a steady heavy-traffic rhythm with a nearly continuous workload and smaller differences between states. That means the duality of the inter-visit intervals gets washed out and inter-visit times become more regular. Waiting times at  $Q_1$  are then a consequence of the overall system congestion and not by rare excursions.

Under **high loads**, the dominant contributor to the quantile is not the job from  $Q_2$  and the switchover times but the time needed to serve the packets from  $Q_1$  in front of the current arrival. Using Little’s Law we can express the length of  $Q_1$  as  $L_1 = \nu_1 E_1(\nu_1, \nu_2)$  and the mean time to serve all the packets in the queue as  $\tau_1 L_1$ . Then, we can approximate the tail as:

$$P \approx 1 - p_1 e^{-\frac{t}{\tau_1 L_1}} \quad (7)$$

$$Q_{CPU}(P) \approx \tau_1 L_1 \ln\left(\frac{p_1}{1 - P}\right),$$

where  $p_1$  is the probability that a packet will experience the long queue of packets from  $Q_1$  before it.

In order to approximate  $p_1$  and  $p_2$ , we use the mean cycle time of the system  $\eta$ . This is mean time it takes for the CPU to cycle through both queues and is equal to  $\eta = 2\sigma + \tau_2 + \tau_1 L_1$ , assuming  $k_1 \geq L_1$ . Under the assumption of exponential arrivals, we can use the PASTA property and calculate the probability that a packet will experience a long inter-visit

interval  $p_2$  as the mean time that the CPU spends on that interval divided by the mean cycle time. Correspondingly,  $p_1$  will be equal to the mean time the CPU spends on serving the packets from  $Q_1$  divided by the mean cycle time. We can formally express these as:

$$p_1 = \frac{\tau_1 L_1}{\eta} \quad (8)$$

$$p_2 = \frac{2\sigma + \tau_2}{\eta}.$$

Having  $p_1$  and  $p_2$ , we can represent the high quantiles  $P > 0.95$  of response time of packets from  $Q_1$  as a sum of both rates:

$$Q_{CPU}(P, \nu_1, \nu_2) = (2\sigma + \tau_2) \ln\left(\frac{p_2}{1 - P}\right) + \tau_1 L_1 \ln\left(\frac{p_1}{1 - P}\right). \quad (9)$$

To verify the accuracy of our approximation, we run simulations with the OMNeT++ INET framework<sup>1</sup> in which we develop the polling model that represents a CPU. We select 3 different values for each of the 3 model parameters  $\tau_1, \tau_2, \sigma$ , running a total of 27 scenarios each with 351 combinations of  $\nu_1, \nu_2$  where  $0.2 < \rho < 1$ . We measure the accuracy of the approximation with Mean Relative Error (MRE) for  $P = (0.97, 0.98, 0.99)$ , resulting in  $MRE = (-0.04, 0.01, 0.03)$  respectively. The majority of the MRE is distributed in the interval  $(-0.1, 0.1)$  for all values of  $P$ , with a relatively-accurate estimation of the Quantile Function (QF).

#### D. End-to-End latency

Since we now have the packet response time QFs for both resources of a machine  $Q_{CPU}$  and  $Q_{NIC}$ , in order to get the end-to-end latency of the network we need to sum the response times for all the CPU and NIC resources used by the flow on its way from source to destination. Since we cannot perform a simple sum on quantiles, let us discuss the end-to-end response time distribution. Under the assumption that the service times of all our resources are independent and exponentially distributed and the output is a Poisson process, the end-to-end packet response time distribution of a flow follows a phase-type distribution, or more specifically hypoexponential [17] with each phase being the CPU or NIC resource that the flow traverses.

Although the following holds for any number of phases, for illustration of our approximation let us use a 2 phase hypoexponential distribution with Cumulative Distribution Function (CDF)  $F(x) = 1 - \frac{l_2}{l_2 - l_1} e^{-l_1 x} - \frac{l_1}{l_1 - l_2} e^{-l_2 x}$  where  $l_1$  and  $l_2$  are the rates of each phase. Since it does not have a closed-form QF, let us replace  $F(x) = P$  and numerically solve the equation:

$$(1 - P)(l_2 - l_1) = l_2 e^{-l_1 x} - l_1 e^{-l_2 x} \quad (10)$$

for  $x$  so we can use the solution to find an approximation of  $Q(P)$ . An important feature of the approximation we need is to include the sum of means since we have them in closed form and we can simply sum them when we calculate flow

<sup>1</sup><https://github.com/inet-framework/inet>

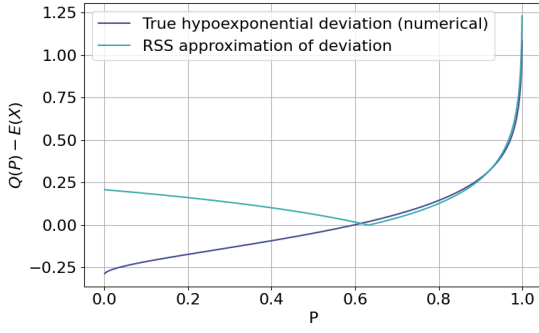


Fig. 4. Quantile deviation from the mean

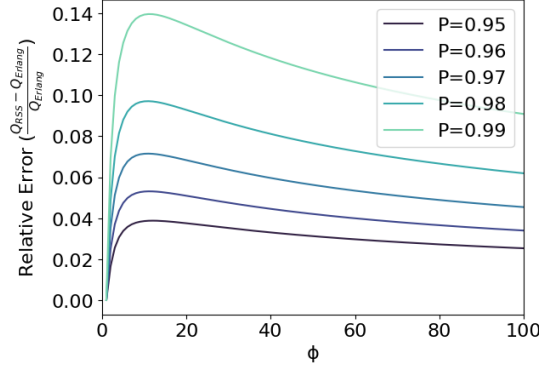


Fig. 5. Relative Error of the quantile approximation to a  $\phi$ -phase Erlang distribution

latencies. We start with the assumption that the deviation of the quantile from the mean  $Q(P) - E[X]$  for the hypoexponential adds in quadrature as Root-Sum-of-Squares (RSS), just like the standard deviation does in the law of propagation of uncertainty with independent variables [18]. This leads us to the following expression for the quantile:

$$Q(P) = \sum_{i=1}^2 \underbrace{\frac{1}{l_i}}_{\text{Mean latency at phase } i} + \underbrace{\sqrt{\sum_{i=1}^2 \left( \underbrace{\frac{-\ln(1-P)}{l_i}}_{\text{Quantile of the latency at phase } i} - \underbrace{\frac{1}{l_i}}_{\text{Mean latency at phase } i} \right)^2}}_{\text{Propagation of uncertainty}}. \quad (11)$$

If we plot the numerical solution of Eq. (10) and Eq. (11) together, as shown in Figure 4 for the example values of  $l_1 = 8, l_2 = 6$ , we observe that for  $P > 0.6$  the deviation from the quantiles for the hypoexponential approximately

behaves as an RSS of the deviations from its phases. The reason it happens mathematically is because the QF of the hypoexponential has the form of a natural logarithm of a sum of exponentials, which in the areas where slowest phase dominates can be approximated by RSS. In the system, this can be explained by the fact that the tail of the end-to-end response time distribution  $P > 0.6$  starts to be dominated by the phase(s) with the slowest response time, while the other phases contribute less. Hence, the accuracy of this approximation depends on the magnitude of differences between the phases with the best accuracy if only one phase contributes, in which case Eq. (11) becomes the quantile of a single exponential distribution  $Q(P) = \frac{-\ln(1-P)}{l_i}$  where  $l_i$  is the dominant phase rate, and the worst accuracy when we have the same phase rates, which is in fact an Erlang distribution with  $\phi$  phases. Although we also do not have a closed-form quantile function for the Erlang distribution, we can analyze the relative errors between its quantile calculated numerically and the RSS based approximation. This is illustrated in Figure 5, with the number of phases  $\phi$  on the  $x$ -axis and the relative error on the  $y$ -axis. We can observe that there is an upper bound to the error at least on the interval  $\phi = (1, 100)$  which are enough phases for the size of our infrastructure. This upper bound depends on the percentile  $P$  and allows us to compensate for the error with a heuristic. The simplest way to do this is to take the mean of the lower and upper bounds of the error, which for  $P = 0.99$  is  $H_P = \frac{0.14-0}{2} = 0.07$ . Then we can introduce a correction factor  $K_P = \frac{1}{1+H_P}$ , which is a simplification that introduces errors in the end-to-end quantile estimations because  $K_P$  actually depends on the phase distributions. Finally, since the end-to-end core network latency is assumed to be hypoexponentially distributed we can approximate the higher quantiles ( $P \geq 0.97$ ) as the sum of the mean latency plus the RSS of the distance between the quantile and the mean latency at each CPU and NIC resource traversed by the flow. Formally, we can express this as Eq. (13).

#### E. Constraints

Since we defined our performance metrics as end-to-end latency, we now need to add the constraints under which our model is valid. Due to the fact we use queueing models, all of them have to be stable. For the M/M/1 model of the NIC that means that the average input rates must be lower than the average service rates which can be formally represented as:

$$\frac{\alpha_n - \bar{\alpha}_n + \gamma_n}{\mu_{NIC}} < 1. \quad (12)$$

$$d_f = K_P \sum_{i=0}^N y_{f,i} \left[ \underbrace{E_1(\nu_{1,i}, \nu_{2,i})}_{\text{Mean network processing time at node } i} + \sum_{j=0}^N R_{s,i,j} \underbrace{[E_1(\nu_{1,j}, \nu_{2,j}) + E_{M/M/1}(\omega_j)]}_{\text{Mean network processing from traversing node } j} \right] + \sqrt{\underbrace{(Q_{CPU}(P, \nu_{1,i}, \nu_{2,i}) - E_1(\nu_{1,i}, \nu_{2,i}))^2}_{\text{From end node } i} + \sum_{j=0}^N R_{s,i,j} \underbrace{[(Q_{CPU}(P, \nu_{1,j}, \nu_{2,j}) - E_1(\nu_{1,j}, \nu_{2,j}))^2 + (Q_{NIC}(P, \omega_j) - E_{M/M/1}(\omega_j))^2]_{\text{From traversing node } j}}}_{\text{Propagation of uncertainty for network processing}}}.$$

For the polling model of the CPU the following relations must hold for each queue:

$$\frac{\alpha_n + \gamma_n}{C} < \frac{k_1(1 - \rho)}{\sigma}, \quad \frac{\beta + \delta_n}{C} < \frac{k_2(1 - \rho)}{\sigma}. \quad (14)$$

We can also set SLAs for each of the flow latencies, so  $P$  ratio of packets from each flow must arrive within  $D_f$  seconds:

$$d_f < D_f, f \in 1..F \quad (15)$$

Finally, a flow can only end in one node:

$$\sum_{n=0}^N y_{f,n} = 1, f \in 1..F. \quad (16)$$

## V. PERFORMANCE EVALUATION

Now that we have fully specified the model, let us discuss the accuracy of its predictions which include both approximations made in Section IV. We first evaluate the relative error between the output of the model with the results obtained via simulation of the same scenario. Then, to gain insights on the impact of the placement of the SFs guided by these estimations, we evaluate the percentage of cases where the model output differs from the simulation on the fulfillment of the SLA for various latency thresholds  $D_f$ .

TABLE II  
MODEL AND SIMULATION PARAMETERS

| Symbol        | Description                                | Value                                                                                           |
|---------------|--------------------------------------------|-------------------------------------------------------------------------------------------------|
| C             | Number of CPU cores                        | 10                                                                                              |
| N             | Number of nodes in the topology            | 15,25,50                                                                                        |
| $\tau_{CPU}$  | Mean service time of jobs on the CPU       | 500 $\mu$ s                                                                                     |
| $\tau_{NET}$  | Mean service time of packets on the CPU    | 8.7 $\mu$ s                                                                                     |
| $\tau_{NIC}$  | Mean service time of packets on the NIC    | 6.1 $\mu$ s                                                                                     |
| $\sigma$      | Mean switchover time                       | 13.5 $\mu$ s                                                                                    |
| $k_{packets}$ | Maximum number of packets served per cycle | 300 packets/cycle                                                                               |
| $k_{jobs}$    | Maximum number of jobs served per cycle    | 1 job/cycle                                                                                     |
| $B$           | Number of background flows                 | $\begin{cases} (50, 150), & N = 15 \\ (100, 200), & N = 25 \\ (150, 250), & N = 50 \end{cases}$ |
| $\lambda_B$   | Packet rates of background flows           | (6,4166) packets/s                                                                              |
| $F$           | Number of service flows                    | $\begin{cases} (5, 30), & N = 15 \\ (20, 50), & N = 25 \\ (30, 60), & N = 50 \end{cases}$       |
| $\lambda_F$   | Packet rates of service flows              | (2976,4166) packets/s                                                                           |
| $\beta$       | Job rates of background jobs               | (2000,14000) jobs/s                                                                             |
| $V$           | Service constant, see Section IV-B         | 1/30                                                                                            |
| $P$           | Network percentile value                   | 0.99                                                                                            |

We run simulations with 3 topologies of 5 edges as shown in Figure 2 but with different number of machines per edge data center: 3, 5, and 10. For each topology, we run 300 different traffic and background load scenarios. Please refer to

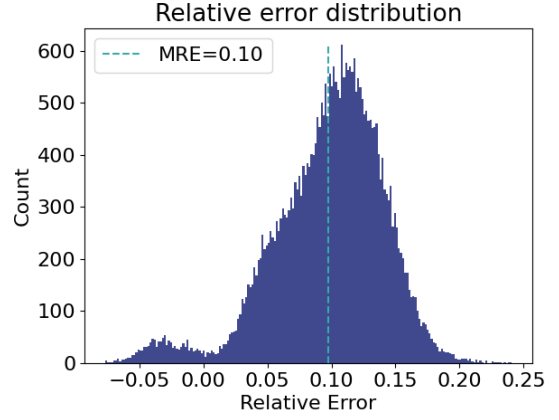


Fig. 6. MRE and relative error distribution of latency predictions.

Table II for the parameters used in the simulations and model predictions. Service times, switchover times, packet and job arrivals are all exponentially distributed. The accuracy of the model is measured with MRE, where the relative error per flow is defined as  $RE_f = \frac{d_{f,predicted} - d_{f,simulated}}{d_{f,simulated}}$ .

The evaluation methodology for an experimental setup with a real infrastructure remains the same and only requires the same topology, size and machine parameters. The measurements needed are only end-to-end network latency quantiles of the service flows. Furthermore, comparisons of the latency predictions to existing models would not be fair because the consequences of their assumptions are that they neither consider load-based performance nor resource sharing between network and CPU-bound workloads. This effectively makes them deterministic models that omit waiting times and variability. To the best of our knowledge, this is the first stochastic performance model at the scale of an edge infrastructure.

### A. Model accuracy

The results are depicted in Figure 6 as a histogram of all the relative errors from flow latency predictions on all topologies (sample size of  $\sim 33000$ ). The model performs well for all topologies, with  $MRE = 0.1$  and relative error distribution approximately equally spread around the mean. We can identify two reasons for the variance. First, the variance of the quantile values in the simulations is very high for higher loads ( $\rho > 0.8$ ). Hence, if a flow traverses a node at high load, the relative error will be higher. Second, the number of hops that a flow traverses on the infrastructure and the differences in the effective service rates on each resource traversed affects the error, which we discuss further in Section IV-D.

### B. SLA evaluation

Since the key motivation for proposing our model is to guide optimization of NF and SF deployments on the mobile network operator edge infrastructure, we also evaluate its performance in terms of SLA violations for different threshold values of  $D_f = [1, \dots, 9]ms$ . Toward this goal, we evaluate the percentage of cases where the model output differs from

the simulation output in the evaluation of the SLA latency constraint specified in Eq. (15). We show the result of the evaluation in Figure 7 separately for  $N = \{15, 25, 50\}$  nodes, with the number of flow SLA evaluations per network size equal to  $\{8400, 10367, 14272\}$ . False positives is the percent of cases in which the model predicted the constraint is fulfilled but the simulation indicated it is not, resulting in SLA violations. False negatives is the percent of cases in which the model predicted the constraint is not fulfilled, resulting in flow rejections, but the simulation indicated it is. The values shown in the figure are for the value of  $D_f$  which was the worst case. We can observe that the model is very conservative and results mostly in false negatives for about 12-14% of the flows. This will result in less efficient utilization but better service quality, which is beneficial in some cases. However, depending on the requirements the model can be tuned to be less conservative with higher values of  $H_P$  which will improve the MRE. The variance of the error will remain similar without an improvement in  $H_P$ , which we leave for future work.

## VI. CONCLUSION AND FUTURE WORK

This paper presented a system model of an edge computing infrastructure based on QT, incorporating load based performance and resource sharing between NFs and SFs. We approximated the quantile function of network packet latency on a single machine and end-to-end on the whole infrastructure. We also validated the model performance in core network latency predictions and SLA fulfillment, confirming its ability to estimate latencies a flow will experience and produce a low number of false positives for SLA evaluations. Since the latency expression has closed form, it is useful in the search of optimums by algorithms because they can manipulate it within constraints or objectives. The computational complexity of the latency expression grows linearly in the number of flows  $F$  and in the mean hop length  $L$  -  $O(FL)$ .

Future work could include a deeper analysis of the error bounds and improvements to model accuracy by developing a better heuristic that makes the correction factor  $K_p$  dependent on the phase distributions of the nodes that a flow traverses. Furthermore, the model assumes that packet, job arrivals and service times are exponentially distributed. In practice, real traffic is often self-similar and service times have higher variance, which may lead to the model underestimating the latency quantiles and is the main reason why we intend to remove this assumption by using a data-driven approach and learn a Graph Neural Network (GNN) model of the infrastructure. The model developed here serves as a good baseline for that work. We also intend to use the model and solve the optimal service placement problem with respect to minimal resource use.

## ACKNOWLEDGMENT

This work was partially funded by the Dutch National Growth Fund through the Future Network Services (FNS) program and the EC through the SNS JU Hexa-X-II project under grant agreement No.101095759.

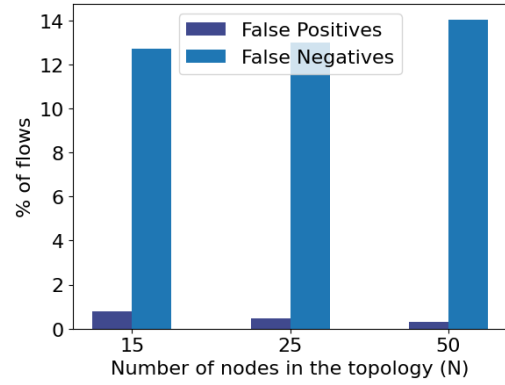


Fig. 7. Differences between model and simulation in the evaluation of the SLA constraint.

## REFERENCES

- [1] W. Attaoui *et al.*, "VNF and CNF placement in 5G: Recent advances and future trends," *IEEE Trans. on Nw. and Service Management*, 2023.
- [2] 3GPP, "Technical specification group services and system aspects; system architecture for the 5g system (5gs); stage 2," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 23.501, 2024.
- [3] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116974–117017, 2020.
- [4] A. Manousis *et al.*, "Contention-aware performance prediction for virtualized network functions," in *ACM SIGCOMM*, 2020, pp. 270–282.
- [5] S. K. Moghaddam, R. Buyya, and K. Ramamohanarao, "Performance-aware management of cloud resources: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 52, no. 4, Aug. 2019. [Online]. Available: <https://doi.org/10.1145/3337956>
- [6] I. Leyva-Pupo, C. Cervell6-Pastor, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic upf placement and chaining reconfiguration in 5g networks," *Computer Networks*, 2022.
- [7] Z. Chen, H. Li, K. Ota, and M. Dong, "Deep reinforcement learning for Aol aware VNF placement in multiple source systems," in *IEEE Global Communications Conference*, 2022, pp. 2873–2878.
- [8] S. Christakis, N. Makris, T. Korakis, and S. Fdida, "On the automated scaling of user plane function for 5g: An experimental evaluation," in *2024 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, 2024, pp. 979–984.
- [9] C. Rotter and T. Van Do, "A queueing model for threshold-based scaling of UPF instances in 5G core," *IEEE Access*, vol. 9, pp. 81 443–53, 2021.
- [10] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks," *Computer Networks*, 2020.
- [11] "NAPI," 2022. [Online]. Available: <https://wiki.linuxfoundation.org/networking/napi>
- [12] S. W. Fuhrmann and Y. Wang, "Analysis of cyclic service systems with limited service: Bounds and approximations," *Performance Evaluation*, 1988.
- [13] S. Geissler, S. Lange, F. Wamser, T. Zinner, and T. Hoßfeld, "Komon — kernel-based online monitoring of vnf packet processing times," in *2019 International Conference on Networked Systems (NetSys)*, 2019, pp. 1–8.
- [14] "About use of kernel parameter 'sched'," 2024. [Online]. Available: <https://access.redhat.com/solutions/177953>
- [15] T. Dimitrovski, B. Turkovic, A. Ganesh, T. Lynar, H. van den Berg, and G. Heijenk, "On performance modeling for the management of cloud-native network functions in closed-loops," *Proceedings of CNSM IWNC '25*, 2025.
- [16] "Tuning net.core.netdev\_budget for network performance," 2025. [Online]. Available: <https://akmatori.com/blog/netdev-budget-tuning>
- [17] "Hypoexponential Distribution," 2016. [Online]. Available: <https://reference.wolfram.com/language/ref/HypoexponentialDistribution.html>
- [18] P. R. Bevington and D. K. Robinson, *Data Reduction and Error Analysis for the Physical Sciences*, 3rd ed. New York: McGraw-Hill, 2003.