Discrete-Time Modeling of NFV Accelerators that Exploit Batched Processing

STEFAN GEISSLER, University of Wuerzburg
STANISLAV LANGE, Norwegian University of Science and Technology
LEONARDO LINGUAGLOSSA, Telecom ParisTech
DARIO ROSSI, Telecom ParisTech
THOMAS ZINNER, Norwegian University of Science and Technology
TOBIAS HOSSFELD, University of Wuerzburg

Network Functions Virtualization (NFV) is among the latest network revolutions, promising increased flexibility and avoiding network ossification. At the same time, all-software NFV implementations on commodity hardware raise performance issues when comparing to ASIC solutions. To address these issues, numerous software acceleration frameworks for packet processing have been proposed in the last few years. One central mechanism of many of these frameworks is the use of *batching techniques*, where packets are processed in groups as opposed to individually. This is required to provide high-speed capabilities by minimizing framework overhead, reducing interrupt pressure, and leveraging instruction-level cache hits. Several such system implementations have been proposed and experimentally benchmarked in the past. However, the scientific community has so far only to a limited extent attempted to model the system dynamics of modern NFV routers exploiting batching acceleration. In this paper, we propose a simple, generic model for this type of batching-based systems that can be applied to predict all relevant key performance indicators. In particular, we extend our previous work and formulate the calculation of the queue size as well as waiting time distributions in addition to the batch size distribution and the packet loss probability. Furthermore, we introduce the waiting time distribution as a relevant QoS parameter and perform an in-depth parameter study, widening the set of investigated variables as well as the range of values. Finally, we contrast the model prediction with experimental results gathered in a high-speed testbed including an NFV router, showing that the model not only correctly captures system performance under simple conditions, but also in more realistic scenarios in which traffic is processed by a mixture of functions.

CCS Concepts: • **Networks → Network performance modeling**; **Programmable networks**; **Network measurement**.

Additional Key Words and Phrases: packet processing, vector packet processing, DPDK, performance model, discrete-time analysis, waiting time, network measurement

ACM Reference Format:
Stefan Geissler, Stanislav Lange, Leonardo Linguaglossa, Dario Rossi, Thomas Zinner, and Tobias Hossfeld. 2021. Discrete-Time Modeling of NFV Accelerators that Exploit Batched Processing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 0, 0, Article 0 (2021), 27 pages. https://doi.org/00000000

Authors' addresses: Stefan Geissler, stefan.geissler@uni-wuerzburg.de, University of Wuerzburg; Stanislav Lange, stanislav.lange@ntnu.no, Norwegian University of Science and Technology; Leonardo Linguaglossa, leonardo.linguaglossa@telecom-paristech.fr, Telecom ParisTech; Dario Rossi, dario.rossi@telecom-paristech.fr, Telecom ParisTech; Thomas Zinner, thomas.zinner@ntnu.no, Norwegian University of Science and Technology; Tobias Hossfeld, hossfeld@uni-wuerzburg.de, University of Wuerzburg.

2376-3639/2021/0-ART0 $15.00
https://doi.org/00000000

1 INTRODUCTION

All-software processing of network traffic has unleashed the possibility to rapidly deploy and update new protocols and features in both the control and the data plane. Particularly, ASICs still dominate the network *core*, where the network fabric performs simple processing like IP forwarding or MPLS switching at several terabits per second. In contrast, all-software stacks are gaining popularity at the network *edge*, where software can deliver feature-rich packet processing for a large variety of protocols at tens to hundreds of gigabits per second. Software routers have been introduced nearly two decades ago [31] but their adoption has been slow due to severe performance bottlenecks, which made the idea appealing but limited to research prototypes. Yet, the situation changed drastically in the last decade, with the introduction of the so-called "kernel-bypass" network stacks [26, 49] that started offering efficient low-level building blocks for multi-threaded user-space processing of network traffic at line-rate. As a result, full-blown software stacks, enabling more complex use cases in the Software Defined Networks (SDN) and Network Functions Virtualization (NFV) areas started rising in the software ecosystem. Open Virtual Switch (OVS) [42] and Vector Packet Processor (VPP) [18] are two examples.

To achieve high-speed processing, these software frameworks share commonalities [7, 34] such as the use of lock-free multi-threading as well as the use of *poll-mode batched processing*. While the use of multi-threading allows horizontal scaling and makes each thread independent from the others, the use of *batching* is a distinctive characteristic of modern high-speed packet processing frameworks. Particularly, batching is used for both fetching packets from the Network Interface Card (NIC) by low-level drivers to reduce interrupt pressure [26, 49], as well as for processing batches of packets in higher-level applications to amortize framework overhead [7, 18, 28, 29].

However, while a large number of system *implementations* exist, and while some work recently started undertaking an experimental comparison of these implementations [7, 20, 44], to the best of our knowledge our initial model is the first system *model* that can explain and accurately predict the measurable system performance of such batch-based packet processors. Although a model for VNF processing times is proposed in [22], its applicability is restricted to systems that process each packet individually. However, batching departs radically from such classic models where packets arrive independently and are independently buffered and treated. Indeed, batching not only correlates arrival and departure, but can also influence the average per-packet processing time. While queueing models that feature batched arrivals at the processing unit are not entirely new and have been used to better capture phenomena such as bursty TCP behavior [1, 30, 38, 39], both the use case and the particular processing schemes differ significantly. Finally, modern systems for high-speed packet processing adopt several low-level techniques to speed-up the processing time. The efficiency of such techniques is severely affected by the experienced batch sizes. This in turn introduces a dependency between the processing efficiency, and hence the service time, and the batch size.

This paper extends our previous model of high-speed software routers using batching acceleration [32]. To keep this work self-contained, we present the general model that is able to accurately characterize the most distinctive parameters of next-generation software routers, including the packet loss probability, batch size distribution and queue size distribution. Furthermore, we extend the model to predict further KPIs, such as a more detailed queue size distribution as well as waiting time and per-packet processing time distributions. Finally, we perform a detailed parameter study, investigating both the performance impact of various parameters as well as validate the fit of the model when compared to measurement results in a dedicated testbed.

In the remainder of this paper, we first introduce the architecture of a modern NFV software router in Sec. 2. We develop a discrete-time queuing model in Sec. 3, after which we describe the

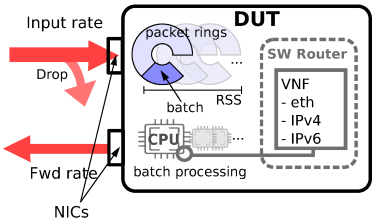


Fig. 1. Synopsis of the device under test (DUT) with receive side scaling (RSS) queues.

experimental setup used for the model validation in Sec. 4. Results of the validation as well as the performed parameter study are presented in Sec. 5. Finally, we put this work in the context of related efforts in Sec. 6 and summarize our findings in Sec. 7.

2 BATCHED PACKET PROCESSING

We start by presenting background information about the latest generation of high-speed software packet processors that is represented at a high level in Fig. 1. We refer to the same figure later to detail our experimental testbed in Sec. 4. The Device Under Test (DUT) consists of a Common Off-The-Shelf (COTS) server equipped with one or more Network Interface Cards (NICs). The DUT runs an instance of a software router that implements a set of Virtual Network Functions (VNFs). Examples of such functions include Ethernet switching, IPv4/IPv6 forwarding, Access Control Lists, load balancing, proxying, etc. Irrespectively of the specific functions, the system has a number of low-level architectural characteristics that we introduce here, and abstract in the next section, to provide a tractable yet accurate analytical model.

2.1 Packet Ring and RSS

When packets are received at the NIC, they are written to a buffer, called *packet ring*, that is also accessed by the software to retrieve the incoming packets. Writing happens without involving the CPU, using Direct Memory Access (DMA), and does not involve costly memory copy operations. This memory area acts as a *circular queue*: when the input rate is higher than the processing rate, the oldest packets might be overwritten by new arrivals. Hence, unlike in classic FIFO queues, older packets are dropped when the buffer is full.

Modern NICs expose multiple RX/TX hardware queues for the same link. Software frameworks can leverage Receive Side Scaling (RSS) [25] to bind different CPU cores to different of these RSS hardware queues. Thereby, incoming traffic is balanced across different RSS queues based on a hashing function, which allows parallelizing packet processing with the number of available CPU cores. Therefore, each CPU is assigned to a separate instance of the software router, managing its own specific RSS queue with its own packet ring. Since RSS makes each thread independent, it is sufficient to analyze the performance of a single RSS queue as handled by a single core. Indeed, due to the lack of synchronization and locking issues, the aggregated system performance scales linearly with the number of cores [6]. Hence, for modeling purposes, it is sufficient to focus on a single RSS queue.

2.2 Polling and I/O Batching

Traditionally, the networking stack generates an *interrupt* every time a new packet is received by the NIC, signaling the CPU that all processing should stop in order to deal with packet I/O. Under heavy load, this mechanism is known to be very inefficient, leading to a livelock on the CPU [37]. To alleviate this issue different interrupt mitigation mechanisms have been introduced.

One such mechanism is *polling* [48]: at very high traffic rates, the CPU continuously checks for packets stored in the packet ring without raising any interrupt.

Polling mechanisms are typically coupled with *batching*. Meaning when the CPU polls a device, it gathers a group of contiguous packets in the ring and the whole batch is passed to the processing application. A similar procedure is executed during packet transmission, when packets scheduled to be transmitted are forwarded in batches. Batching is a powerful mechanism that speeds up overall processing, as it amortizes the fixed costs of the I/O over multiple packets [24, 29] and is as such supported by all modern networking stacks [26, 49].

A maximum batch size β is usually defined to specify an upper limit on the number of packets to be taken by an atomic poll operation, so that the size of the polled batch can take any value in $[0, \beta]$. This is done to parametrize the trade-off between the processing efficiency of larger batches and the reduced jitter of smaller batches [35]. The impact of this value on the overall system performance under different circumstances is evaluated later in this work in Section 5.

2.3 Compute Batching

More recently, the use of batching has been extended beyond packet I/O and has been applied to the processing of packets as well. Indeed, network function computation can similarly benefit from grouped processing, which is known as *compute batching*.

Shortly, when a VNF is executed on a batch, this allows sharing the overhead of the packet processing frameworks between multiple packets, e.g., all processing instructions are initialized once per batch rather than once every packet. Additionally, it increases the efficiency of the underlying CPU pipelines since the VNF code raises a single miss for the first packet in the batch, but is then subsequently cached in the L1 instruction cache for the remainder of the batch.

Whereas the actual implementation of compute batching differs among frameworks (e.g. the compute batching implementations of G-opt [28], DoubleClick [29], FastClick [7] and VPP [18]), compute batching is another popular technique in modern high-speed packet processing frameworks. Hence, defining a *general* model that can be applied to different frameworks with heterogeneous implementations of compute batching techniques is a relevant goal.

2.4 Workflow and System Parameters

In a typical scenario, a software router binds one (or more) CPU(s) to one (or more) RSS queue(s). The CPU then enters a main loop in which the NIC is polled at every iteration. Consequently, the application collects a batch of packets and starts performing the packet processing. During this time new packets might have arrived at the NIC and are stored in the packet ring(s) until next iteration. We now describe the most important metrics and parameters regarding the tuning and evaluation of our model and briefly highlight their respective interaction effects.

2.4.1 Batch size. The first parameter is the batch size of the system, describing the number of packets that are processed in a single polling, processing and transmitting cycle. There is an intuitive correlation between the size of the batches and the input rate observed at the system. In the extreme case of input rate zero, the CPU keeps polling the NIC, but at every iteration it will find no packets, thus the average batch size is zero as well. The opposite extreme is when the input rate is higher than the processing rate. In such case, packets are written to the packet rings and the CPU cannot cope with the incoming rate. Hence, packets are overwritten and losses occur. Since the CPU will always find a full ring, it will intuitively pick up as many packets as possible, which is limited by the internal maximum batch size. Subsequently, the measured average batch size will converge to the maximum batch size. The most interesting scenario happens in between the two extreme cases, where the input rate is between zero and the maximum sustainable rate of the software router. In

this case, at every iteration, the CPU will poll the device and find some amount of packets that will be processed. The number of packets observed at every polling event depends on the processing time of the batch collected in the previous cycle as well as the incoming packet rate and interarrival time distribution. In practice, this relation between the number of arrivals during a processing cycle in combination with the fact that the system becomes more efficient for larger batches, leads to an automatic feedback loop that helps maintaining a stable equilibrium regarding the batch size. As already mentioned, smaller batches are processed less efficiently due to caching effects and the distribution of framework overhead in batch processing scenarios. Hence, small batches lead to an increase in batch size for the next cycle. At some point, the system reaches a state at which the processing efficiency has increased until the mean number of arrivals during a processing cycle equals the number of arrivals during the previous cycle. From that point, fluctuations in the batch size are mainly attributed to fluctuations in the arrival process. The impact of the maximum batch size is evaluated during the parameter study by investigating the impact of different values under varying circumstances in Section 5.

2.4.2 Processing time. Next, we describe the processing time of batches. When an increase in the arrival rate occurs or the system has not yet reached equilibrium, the next batch will be larger but the processing efficiency will be higher due to amortizing the fixed costs over a greater number of elements. Therefore, when a batch is larger, the per-packet processing time is smaller, impacting the size of the batch collected in the following polling cycle. Further, the processing time of batches is also affected by the network function that has to be applied to each packet. Some simple functions such as Ethernet switching intuitively require less processing than more complex functions such as IPSec or DPI. Finally, it needs to be taken into account that packets within a single batch could potentially require differentiated processing and will hence exhibit varying processing times. In these scenarios, the processing of the batch is considered complete once the last packet of the batch has been processed. The processing time of different batch sizes is one of the input parameters of the proposed model. Section 5.1 provides insights into the tuning of the model based on observed values.

2.4.3 Packet loss. Moving on, packet loss is one of the most crucial metrics when it comes to software routers as it relates directly to how well a VNF performs its job. Typical software routers are able to sustain a rate of 12 to 14 millions packets per second (Mpps) [7, 49]. When sending 10Gbps traffic of minimum-sized packets on a wire, this translates into a rate of 14.88[1] Mpps. As mentioned before, losses occur if the packet arrival rate is higher than the packet processing rate as this leads to packets being overwritten in the receive side (RX) rings. However, losses may occur even at lower rates, because of the aforementioned dependencies between packet processing time, batch size and efficiency of the framework or in edge cases with arrival processes exhibiting large bursts. The packet loss probability is one of the output metrics generated by the proposed model and is evaluated for different scenarios during the parameter study.

2.4.4 Queue Size and Waiting Time. Finally, directly related to the packet loss probability as well as the sojourn time of the system is the queue size, meaning the number of available slots in the RX ring able to hold arriving packets. As mentioned before, loss occurs, whenever a new packet arrives while all slots are currently occupied. In addition, the queue size impacts the waiting time experienced by arriving packets that are not collected in the next polling cycle. Instead, these packets have to wait for one or more full processing times before they start processing. In this context, the queue size dictates the trade-off between potentially long waiting times and resilience against bursts of packets. The queue size distribution is one of the output parameters predicted by

[1]The minimum-size of a packet is 64 byte, to which we should add the Ethernet header and trailers for an additional 20 bytes

the proposed model. Similarly, the model is able to predict packet waiting times. However, due to technical limitations, we are not able to monitor waiting times in the technical system. Instead, we compare sojourn time distributions composed of the waiting time plus the processing time. Since the processing time is part of the model input, the only unknown remains the waiting time and the comparison is hence valid to establish the quality of our waiting time prediction.

Finally, it is important to note that, in the model, we represent the RX ring as a simple FIFO queue, meaning that, in lossy scenarios, the predicted waiting time will likely deviate form the measurement values. In a ring, old packets get overwritten and hence lost, while new arrivals are discarded instead of enqueued when it comes to the FIFO queue. The error is observed and detailed in Section 5. However, in lossy scenarios, the quantification of the waiting time distribution is only of limited value, since packet loss probability becomes the more meaningful parameter in that case.

3 SYSTEM MODEL

In this section, we describe the queueing model that is used to evaluate the performance of batching-based packet processors. Fig. 2 illustrates its main components, namely an arrival process with arbitrarily distributed packet interarrival times, a limited-capacity FIFO queue as well as a processing unit that regularly polls the queue, picks up limited-sized batches, and processes them with service times that depend on the batch size. We deliberately abstract the circular packet ring with a FIFO queue for the sake of tractability. Intuitively, this does not alter the system performance w.r.t. the amount of lost packets, only *which* packets are lost changes. Furthermore, experiments show that the model achieves a high level of accuracy even despite this simplification (Sec. 5). In this section, after a brief overview of the system states that are captured by the model, we outline how to extract the key performance indicators from the system steady state.

3.1 Discrete-Time Model

Before diving into the details we introduce definitions and notations as well as provide an outline of the model. As is required for the discrete-time approach, we need to discretize time into fixed intervals Δt. For the remainder of this work, we use $\Delta t = 10\,\text{ns}$, as it represents the most suitable resolution to describe our obtained measurement data. Note that the model resolution could be increased further by selecting a smaller Δt, at the expense of additional computational complexity.

For the sake of readability, we provide an overview of the notation used in this manuscript in Tab. 1. The top half contains constants and random variables that constitute the model input, whereas outputs are listed in the bottom half. To disambiguate between random variables (RVs), distributions, and distribution functions, we use the following convention: uppercase letters such as A denote RVs, their distribution is represented by

$$a(k) =_{\text{def}} \mathrm{P}(A = k), \quad k \in [0, \infty)\,,$$

and the corresponding distribution function is defined as

$$A(k) =_{\text{def}} \mathrm{P}(A \leq k) = \sum_{i=-\infty}^{k} a(i), \quad k \in [0, \infty)\,.$$

In the proposed model, the system state at a given time is represented by the corresponding queue size Q_n at the time the n-th batch is polled from the NIC. As highlighted in Fig. 3, all system events, such as packet arrivals as well as polling and batch processing, have a direct impact on the queue size. While each packet arrival leads to an increment of the queue size by one, polling by the processing unit decrements it by the number of packets that are picked up. The latter is limited by the maximum batch size which is denoted as β and the number of packets that reside in the

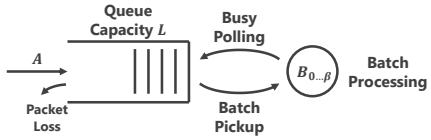


Fig. 2. Model overview.

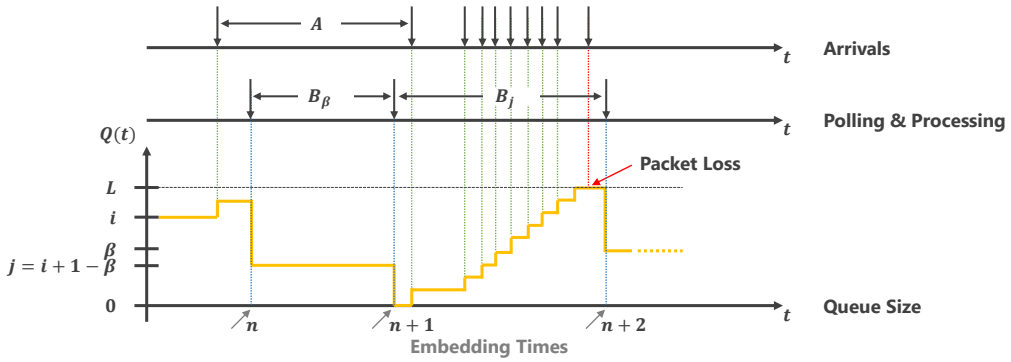


Fig. 3. Exemplary state development of the model.

queue at the time of the polling event. Finally, if an arriving packet finds the queue at its maximum capacity L, the packet is dropped. Hence, the *queue size distribution at the times of embedding during steady state* $q(k)$ can be used to derive relevant performance indicators of the modeled system such as the batch size distribution and the packet loss probability. Additionally, it is possible to derive the *queue size at arrival times* Q_A, which is required for computing the waiting and overall processing time distributions. To obtain event-independent system information, we also present a way of calculating the *queue size at random times* $\bar{Q}$.

In order to derive the distribution of the queue size, we consider an embedded Markov chain whose embedding times are defined to be immediately before the busy polling events of the processing unit. Based on the queue size distribution $q_n(k)$ at these embedding times, we can derive the state probability distribution at consecutive embedding times by taking into account the current batch size and the number of arrival events during the corresponding service time. Finally, we use a fixed-point iteration in order to determine the queue size distribution at steady state $q(k)$. To this end, we leverage the recursive relationship in (1) to compute the queue size distribution immediately before the $(n+1)$-st batch is picked up, based on the queue size distribution immediately before the n-th batch is picked up.

$$q_{n+1}(k) = \begin{cases} \sum_{i=0}^{L} q_n(i) x_{b_{\min(i,\beta)},a}(k - (i - \min(i,\beta))) & \text{for } 0 \le k < L, \\ \sum_{i=0}^{L} q_n(i) \sum_{j=0}^{\infty} x_{b_{\min(i,\beta)},a}(L + j - (i - \min(i,\beta))) & \text{for } k = L, \\ 0 \text{ otherwise.} \end{cases} \tag{1}$$

The first case covers the probability to reach a state with a queue size that is below its capacity L. In order to calculate this probability, every possible previous value for the queue size i at the previous time of embedding is considered. Given i, the size of the batch that is processed

Table 1. Notation.

Variable	Description
L	Queue capacity, equals 4096 if not stated otherwise.
β	Maximum batch size, equals 256 if not stated otherwise.
A, $a(k)$	Packet interarrival time.
B_i, $b_i(k)$	Service time of size i batches.
$X_{\tau,a}$, $x_{\tau,a}(k)$	Number of arrivals whose interarrival time is distributed according to a during an interval whose length is distributed according to τ [21]. If τ is a constant, we implicitly apply the deterministic distribution with probability mass 1 at τ.
Q_n, $q_n(k)$	Queue size immediately before the n-th batch pick up.
Q, $q(k)$	Queue size at embedding times during steady state, before batch pick up.
Q_A, $q_A(k)$	Queue size at arrival times.
$\bar{Q}$, $\bar{q}(k)$	Queue size at random times.
V_n, $v_n(k)$	Batch size immediately before the n-th batch pick up.
V, $v(k)$	Batch size at embedding times.
S, $s(k)$	Batch service time at random times / among all batches.
p_{loss}	Packet loss probability.
W, $w(k)$	Waiting time.
D, $d(k)$	Processing time.

between embeddings equals $\min(i, \beta)$ since the processing unit can pick up at most β packets. From this, we can derive the number of arrivals during the corresponding service time by means of $X_{b_{\min(i,\beta)},a}$, which describes the number of arrivals with interarrival time a in an interval of length $b_{\min(i,\beta)}$ [21]. Since embeddings are placed immediately before polling events, a queue size of k is reached when the number of arrivals during the service time is equal to the difference between k and $i - \min(i, \beta)$, the size of the queue immediately *after* the batch is picked up.

The special case of $k = L$ is calculated in an analogous fashion but it is necessary to take into account packet loss, i.e., the arrival of packets beyond the queue capacity which also results in a queue size of L. Each number of lost packets is represented by the summation index j.

Under stationary conditions, the indexes n and $(n + 1)$ in (1) can be suppressed, i.e.,

$$q(k) = \lim_{n\to\infty} q_n(k).$$

Finally, we note a limitation of the outlined model, namely that the variability of the arrival process has to be reasonably smaller than batch service times. Otherwise, the number of arrivals is overestimated resulting in inaccuracies w.r.t. the KPIs. This is, however, a reasonable assumption taking into account the utilized traffic patterns.

3.2 Key Performance Indicators

Given the queue size distribution, the *batch size distribution* and *packet loss probability* can be derived according to (2) and (3), respectively. While the former is representative of the system's efficiency, i.e., larger batches correspond to lower per-packet processing times, a non-zero value of the latter is indicative of an under-dimensioned system. Furthermore, the queue size distribution at

embedding times also allows calculating queue size distributions both at arrival as well as at random times. These statistics are representative of the system state encountered by arriving packets and a random observer, respectively. In particular, the former serves as the foundation for determining waiting and processing time distributions.

3.2.1 Batch Size Distribution. If the queue size is lower than the maximum batch size β, the two are identical, i.e., the entire queue is emptied upon batch pickup, which is covered in the first case of (2). Queue sizes larger than β result in batch sizes of exactly β and are instead covered by the second case.

$$v(k) = \begin{cases} q(k) & 0 \leq k < \beta, \\ \sum_{i=\beta}^{\infty} q(i) & k = \beta, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

3.2.2 Packet Loss Probability. As noted in the description of (1), packet loss occurs when the number of arrivals during a service interval would lead to a queue size that exceeds the capacity L. Hence, we can describe the packet loss probability as the ratio of the expected number of arrivals beyond this threshold N_{Lost} and the expected total number of arrivals N_{Arrivals}:

$$p_{loss} = \frac{\mathrm{E}\left[N_{\text{Lost}}\right]}{\mathrm{E}\left[N_{\text{Arrivals}}\right]} = \frac{\sum_{i=0}^{L} q(i) \sum_{j=0}^{\infty} (j \cdot x_{b_{\min(i,\beta)},a}(L + \min(i,\beta) - i + j))}{\sum_{i=0}^{L} q(i) \sum_{j=0}^{\infty} (j \cdot x_{b_{\min(i,\beta)},a}(j))} \tag{3}$$

Similarly to (1), we consider all possible queue sizes i and use the corresponding probability $q(i)$ as a weighting factor. For each number of lost packets, represented by the summation index j in (3), we calculate the probability for the arrival of $(L + \min(i,\beta) - i + j)$ packets that are required for filling and exceeding the queue. For the expected total number of arrivals, we proceed in an analogous fashion but do not have to shift the distribution of the number of arrivals.

3.2.3 Queue Size Distribution at Arrival Times. While the queue size distribution at the times of embedding allows us to compute the batch sizes that are picked up during busy polling events, we need a shift of perspective in order to determine the waiting time distribution of packets. In particular, the waiting time of a packet starts as soon as it arrives in the system and the time it spends in the system depends on its position in the queue.

Equation 4 shows how the queue size distribution as seen by arrivals, $q_A(k)$, can be computed. Based on the queue size at embedding times, it is possible to determine the resulting batch size as well as the distribution of the number of arrival events during the corresponding batch service interval. The probability of an arrival finding a certain queue size can then be computed via the ratio of the number of arrivals that find the queue at that specific level and the total number of arrivals.

$$q_A(k) = \frac{\sum_{i=0}^{L} q(i) \sum_{j=0}^{\infty} x_{b_{\min(i,\beta)},a}(j) \sum_{m=i-\min(i,\beta)}^{i-\min(i,\beta)+j-1} \mathbf{1}_{\{\min(m,L)=k\}}}{\sum_{i=0}^{L} q(i) \sum_{j=0}^{\infty} x_{b_{\min(i,\beta)},a}(j) \cdot j} \tag{4}$$

3.2.4 Waiting and Processing Time Distributions. Given the queue size distribution at arrival times, we can derive the distribution of the waiting time by decomposing it into the following two parts. First, the time between a packet's arrival and the next batch pick-up event. Second, zero or more service times of size-β batches, depending on the packet's position in the queue.

We obtain the first component by considering the recurrence time of the overall batch service time R_S i.e., the time a random observer has to wait to encounter a batch pick-up event. To this end, we derive the distribution of observed batch service times $s(k)$ by weighting the batch size-dependent service times with the occurrence probabilities of the corresponding batch sizes as follows:

$$s(k) = \sum_{i=0}^{\beta} v(i) \cdot b_i(k).$$

For the second component, we leverage the fact that each possible queue size Q_A that can be encountered by an arriving packet can be mapped to a specific number of full batches that are serviced before it.

Subsequently, the distribution of the corresponding duration for servicing the respective number of batches can be obtained by convolving b_β with itself. In terms of random variables, the waiting time can therefore be expressed as

$$W = R_S + \sum_{i=1}^{\left\lfloor \frac{Q_A}{\beta} \right\rfloor} B_\beta, \tag{5}$$

where R_S denotes the recurrence time of $s(k)$.

Finally, the total time a packet spends in the system can be calculated as the sum of the waiting time W and the service time S:

$$D = W + S. \tag{6}$$

3.2.5 Queue Size Distribution at Random Times. While the perspective of individual packets that arrive at the system can be useful when calculating performance indicators such as the waiting time, the queue size distribution at random times provides generic steady-state system information that is independent of specific events.

In order to determine the queue size distribution at random times, we reason about the possible development of the queue between two times of embedding. We illustrate the development between the n-th and $(n+1)$-st embedding times in Fig. 4. Let the queue size equal Q_{init} immediately before a batch of packets is picked up for processing. Depending on Q_{init}, the resulting batch size V of the observed batch at the time of the n-th embedding is between 0 and β. During the corresponding batch service time B_V, a number of $X_{b_V,a}$ arrival events take place and sequentially increase the queue size. The relative time the queue spends on each level is proportional to the interarrival time A for most packets. Exceptions include the first and last level since they are interrupted by the current and subsequent batch pick-up events, respectively. Hence, their duration is proportional to the recurrence time of the interarrival time, R_A. Other exceptions include the case of no arrivals and arrivals that find the queue fully occupied. In those cases, the queue spends the entire time on the same level or a prolonged time on the maximum level, respectively. The recurrence time R_A [52] of a RV A can be computed as

$$r_A(t) = \frac{1}{E[A]} \cdot (1 - A(t)). \tag{7}$$

We use Equation 8 to calculate $\bar{q}(k)$, i.e., the probability of observing a queue size of $0 \leq k < L$, as follows. $\mathbf{1}_{\{condition\}}$ thereby represents the indicator function, assuming 1 if the condition is true, 0 otherwise. The first term considers reaching queue size k immediately after the batch pick-up. In this case, the queue size either remains equal to k the entire time if no additional arrivals take place

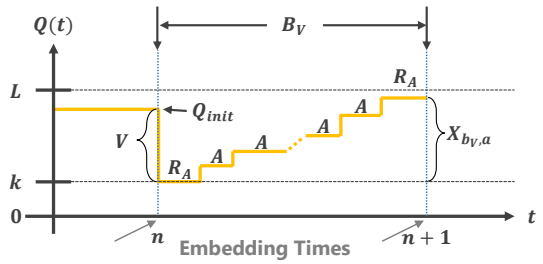


Fig. 4. We use the queue size development between two embedding times to derive $\bar{q}(k)$.

during the batch service time, or it stays at size k for a time proportional to the recurrence time if there are additional arrivals. The second term deals with cases in which the queue size is reduced to a value lower than k when a batch is picked up, and a queue size of k is reached either as an intermediate state or as the final state prior to the next pick-up event. Depending on this, the time spent at a queue size of k is proportional to either the interarrival time A or its recurrence time R_A.

$$\begin{aligned}\bar{q}(k) = &\sum_{i=0}^{L} q(i) \cdot \mathbf{1}_{\{i-\min(i,\beta)=k\}} \cdot \sum_{j=0}^{\infty} b_{\min(i,\beta)}(j) \cdot \left(x_{j,a}(0) + \sum_{l=1}^{\infty} x_{j,a}(l) \cdot \frac{\mathrm{E}[R_A]}{(l-1)\,\mathrm{E}[A] + 2\,\mathrm{E}[R_A]} \right) + \\ &\sum_{i=0}^{L} q(i) \cdot \mathbf{1}_{\{i-\min(i,\beta)<k\}} \cdot \sum_{j=0}^{\infty} b_{\min(i,\beta)}(j) \cdot \left(\sum_{l=k-i+1}^{\infty} x_{j,a}(l) \cdot \frac{\mathrm{E}[A]}{(l-1)\,\mathrm{E}[A] + 2\,\mathrm{E}[R_a]} + \right. \\ &\left. x_{j,a}(k-i) \cdot \frac{\mathrm{E}[R_A]}{(l-1)\,\mathrm{E}[A] + 2\,\mathrm{E}[R_A]} \right) \\ &\text{for } 0 \le k < L. \end{aligned} \tag{8}$$

For the special case of $k = L$, we need to account for the fact that the proportion of time the queue spends at its maximum occupancy may be larger due to the occurrence of packet loss. We derive the corresponding term in Equation 9:

$$\bar{q}(L) = \sum_{i=0}^{L} q(i) \cdot \sum_{j=0}^{\infty} b_{\min(i,\beta)}(j) \sum_{l=L-i+\min(i,\beta)}^{\infty} x_{j,a}(l) \cdot \frac{\mathrm{E}[R_A] + (L-l-1)\,\mathrm{E}[A]}{(l-1)\,\mathrm{E}[A] + 2\,\mathrm{E}[R_A]}. \tag{9}$$

4 EXPERIMENTAL SETUP

To validate our model, we instrument a testbed operating a real NFV software router following the IETF benchmarking guidelines [8]. This section describes our hardware and software setups as well as the scenarios we use to assess the accuracy of our model. We point out that, to assist reproducibility of our work, all the experimental data we collect is available at [33].

4.1 Hardware Setup

We reproduce the experimental setup that has been illustrated earlier in Fig. 1. Our hardware consists of two COTS Desktop PCs, equipped with two Intel 82599ES dual port NICs operating at 10 Gbps. Each node has an i7-2600 processor, running at 3.40 GHz. Each processor has 3 levels of cache hierarchy, ranging from 32 KB for the L1 to 8 MB for the L3. Both machines are equipped with 16 GB of memory.

Table 2. Experimental configuration parameters.

	Parameter	Value
HW	NIC	Intel 82599ES dual-port 10 Gbps
	CPU	i7-2600@3.4 GHz
	Caches L1/L2/L3	32 KB/256 KB/8 MB
DUT	Software router	VPP 19.04
	Number of CPU cores	1
	Number of RSS queues	1
	Memory allocated	4 GB
	Size of input queue (pkts)	$L = \{1024, 2048, 4096\}$
	Max DPDK batch size (pkts)	32
	Max VPP batch size (pkts)	$\beta = \{64, 256, 512, 1024\}$
TX/RX	Traffic Generator	MoonGen
	Rate span [min:inc:max]	$[0.5 : 0.5 : 10]$ Gbps
	Hi/Lo rates	10 Gbps / 2.5 Gbps
	Packet sizes	{64, 128, 256, 512, 1024} B
	Arrival rate process	Constant bit-rate (CBR)
	Data points per configuration (pkts)	138k
	Functions	{ XC, Eth, IPv4, IPv6 }
	Scenarios	Homogeneous vs Heterogeneous

We use one node as our Device Under Test (DUT) and the other for traffic generation (TX) and reception (RX). The DUT receives traffic from one input line-card, performs the packet processing, and then proceeds with the forwarding to the designated output port. We conduct our measurements at the TX and RX side in order to assess the packet ingress and egress rate as well as packet loss. Additionally, we measure directly within the DUT in order to obtain batch sizes and per-batch processing time.

In order to ensure reproducibility and eliminate operating system scheduling, we run the DUT on a single CPU core attached to a single RSS queue, as is typically done in stress-test conditions.

4.2 Software Setup

4.2.1 DUT. To validate the model, we select a state-of-the-art NFV software stack that employs batched processing. In particular, we conduct experiments with the Vector Packet Processor (VPP) [18]. VPP implements VNFs as software components (*nodes*) that can be linked together in a specific configuration (*forwarding graph*). A specific input node (*dpdk-input*) polls the line-card for new packets, grabbing a batch (*vector*) from the ring for processing. Notice from Tab. 2 that VPP compute-batches may aggregate several DPDK I/O-batches, as the maximum VPP batch size is larger than DPDK's. VPP then processes all packets in the vector node-by-node instead of traversing the graph packet-by-packet. Hence, in addition to sharing the framework overhead over the batch, only the first packet triggers fetching of processing code in the L1-instruction cache of the CPU, whereas processing of subsequent packets benefits from L1-instruction cache hits [5, 35].

Also notice that this process naturally introduces branches, as packets may trigger different functions implemented in different nodes of the forwarding graph. This requires splitting the original heterogeneous batch into smaller homogeneous batches for the subsequent nodes. This is expected to change the operational point of the NFV router, as not only the splitting process incurs an additional overhead, but also since the framework overhead is now shared over a smaller batch, and the code heterogeneity increases the L1-instruction cache miss rate. It is thus important to assess experimental performance under realistic scenarios involving multiple functions. Furthermore, we investigate the impact of varying maximum batch sizes as well as the size of the RX ring for different load levels.

4.2.2 TX/RX. For traffic generation and reception, we use MoonGen [15], a state-of-the-art programmable tool capable of sustaining 10 Gbps line-rate. MoonGen also provides APIs to perform basic measurements at the TX/RX side. For example, it is possible to access the NIC's hardware counters to precisely measure the number of packets transmitted and received, which allows to derive the experimental forwarding and loss rates for comparison with the model.

Typically, a single DUT thread on a single RSS queue under commonly considered NFV workloads is able to sustain a rate of 12–14 Mpps [7, 49]. As such, when sending 10 Gbps worth of traffic at minimum-sized 64 Bytes packets on a wire, corresponding to a rate of 14.88 Mpps, we expect the system to be in a lossy regime. As such, we assess the system performance for different rates, ranging from 0.5 Gbps to 10 Gbps with a step increment of 0.5 Gbps. For the sake of illustration, we also consider two exemplary operational points, representing a high-rate (10 Gbps) and a low-rate (2.5 Gbps) regime. Additionally, we assess the system performance for differently sized packets, ranging from 64 Bytes to 1024 Bytes.

4.3 Scenarios

We consider two VNF cases, in which the router is stressed with either *homogeneous* traffic that triggers the same function or *heterogeneous* traffic that activates a mixture of functions. We select popular functions in the NFV ecosystem that allow us to focus on different components of the framework. We use the simplest function to investigate I/O batching and introduce different types of lookup and data structures to provide instances of compute-batching with different complexity.

4.3.1 Homogeneous Cross-Connect Function. In this scenario a single VNF, usually referred to as *cross-connection* (XC), is applied to all packets, representing the baseline of *homogeneous functions* in an NFV router. In this case, the VPP DUT is configured to take all the packets from one input interface and immediately forward them to a fixed output interface. Notice that for the XC VNF, no computation is needed on the headers of the transferred packets since the DUT simply moves batches from the input to the output NIC. Therefore, this scenario helps assessing whether the model faithfully reproduces the impact of I/O-batching.

We generate our workload using a MoonGen script that sends a stream of packets at a fixed rate, namely copies of a templated UDP packet. Notice that for such a simple VNF, the type of traffic does not affect the processing time. Since neither processing nor branching happens, XC performance represents an upper bound for the performance of the NFV router.

4.3.2 Heterogeneous Eth/IPv4/IPv6 Functions. As pointed out in [44], as network traffic is heterogeneous, NFV routers need to handle a mixture of different functions. We therefore consider the case of three different functions that operate on the same traffic batch. Specifically, we consider three functions with different sizes of inputs (48, 32, and 128 bits), lookup types (exact vs longest-prefix match), and data structures (hash tables vs tries). In particular, we consider traffic that triggers the following operations, in increasing order of complexity: (i) a 48 bit exact-match Ethernet lookup, (ii)

a 32 bit IPv4 longest-prefix match lookup using a trie structure, and (iii) a 128 bit IPv6 longest-prefix match lookup that performs a lookup over multiple hash tables for different netmask lengths.

For the sake of simplicity, our experiments are performed with an even split of the functions, i.e., each of the above traffic types consume $\frac{1}{3}$ of the bandwidth, so that each function activates with probability $\frac{1}{3}$, resulting in different function breakdowns across batches. We point out that more complex scenarios (e.g., featuring an uneven split, a larger set of functions, or longer chains) are within the capability of the model, but are out of the scope of this paper.

5 MODELING VS EXPERIMENTAL RESULTS

Before we validate our model via experimental results from the homogeneous and the heterogeneous traffic scenarios, we discuss several options that are available for tuning the model inputs. These options represent different trade-offs in terms of the resulting prediction accuracy, the model's general applicability, as well as the amount of measurements that are required prior to its application. Using appropriate model settings, we then compare model-based performance predictions with our measurements. In particular, we focus on the batch size and the waiting time.

5.1 Model Tuning Options

As detailed in Sec. 3, the model input consists of the queue capacity L, the maximum batch size β, the distribution of packet interarrival times $a(k)$, and the size-dependent distributions of batch processing times $b_i(k)$. For the purpose of model tuning, we fix the values for L and β at 4096 and 256, respectively.

While the mean packet interarrival time $\mathrm{E}[A]$ can be determined from the applied rate, our model provides a degree of freedom by allowing to set an *arbitrary distribution* to reflect aspects like the traffic's burstiness: to this end, we consider a total of four distributions that have varying degrees of variation. In particular, these include (i) the Poisson distribution whose coefficient of variation equals $\frac{1}{\sqrt{\mathrm{E}[A]}}$, (ii) the geometric distribution with a coefficient of variation of $\frac{1}{\sqrt{q}}$, and (iii)-(iv) negative binomial distributions whose parameters are set to achieve coefficients of variation equal to 0.5 and 2, respectively.

Furthermore, we use our measurements to obtain $\mathrm{E}[B_i]$, the mean size-dependent batch service times. Similarly to the packet interarrival time, we can use different distributions to model the behavior of the processor. However, all conducted measurements yielded a very low degree of variation when considering a particular combination of applied rate and the corresponding per-size batch service time. Hence, we use Poisson distributions for the service time.

Additionally, the model might require service time distributions for batch sizes that did not occur in the measurements. In order to provide these missing distributions, the mean service times for the remaining values of the batch size are required. We obtain these by means of linear fitting. In particular, we interpolate the missing mean service times based on the means of observed values. The Poisson distributions for the service time are then generated with measurement-based means where available and with fitted means otherwise. This fitting procedure can be done either globally or on a per-rate basis. This choice represents a trade-off between the overhead for per-rate measurements of the service time, risking overfitting the model to a particular scenario, and a possible improvement w.r.t. the resulting accuracy.

5.1.1 Per-rate vs Global Fit. We illustrate the immediate effects of the chosen fitting strategy in Fig. 5. While the x-axes denote the batch size, the y-axes represent the mean service time in microseconds, and different colors represent different packet arrival rates. Each dot represents a mean service time that is obtained from measurements and lines correspond to linear fits. In particular, we observe that the slope of the linear fit can vary depending on the considered arrival

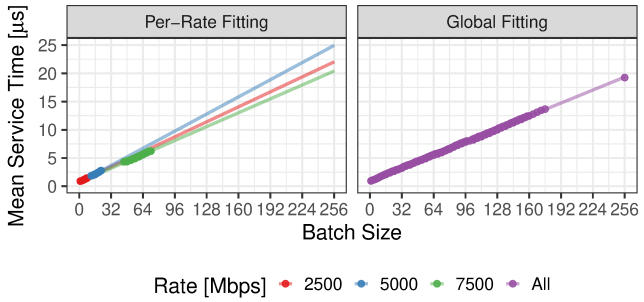


Fig. 5. Size-dependent batch service times. Points indicate mean values from measurements, lines denote linear fits. Cross-connect scenario with $\beta = 256$, $L = 4096$, and 64 B packets.

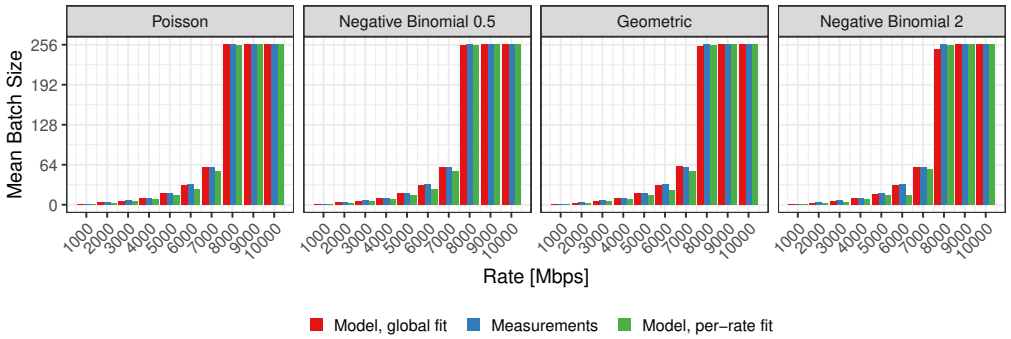


Fig. 6. Mean batch sizes for different rates, arrival processes, and fitting strategies. Cross-connect scenario with $\beta = 256$, $L = 4096$, and 64 B packets.

rate and therefore might lead to a larger error when the model has to take into account batch sizes that did not appear in a measurement run. Furthermore, the range of observed batch sizes widens when increasing the rate. This effect can be explained by shorter packet interarrival times for which the same amount of service time fluctuation leads to a bigger variation in terms of the resulting batch size. At 7.5 Gbps, we observe the widest range of batch sizes, whereas for 10 Gbps the batch size is consistently maximal and the system likely operates in a lossy regime. Note that due to this behavior, only a single data point for 10 Gbps is available and as no linear fit is possible in that case, the data point is not included in the per rate fitting, but has been taken into account during global fitting.

In order to evaluate the impact of the fitting strategy as well as the distribution of the packet interarrival time, we apply our model to the XC scenario and compare the resulting mean batch size with our measurements. For different rates on the x-axes, the graphs in Fig. 6 display the mean batch size on their y-axes. The four subplots correspond to evaluations that use Poisson, negative binomial, geometric, and negative binomial distributions with corresponding coefficients of variation equal to $1/\sqrt{\mathrm{E}[A]}$, 0.5, 1, and 2, respectively. Bars of different colors represent the measurement data (middle, blue) surrounded by results from the model with the two fitting strategies: the global fit (red bars to the left of measurement data) vs per-rate fit (green bars to the right).

As evidenced by the similar development of the mean batch size and the correct identification of the saturation for rates greater than 7 Gbps, all four considered model variants lead to a high degree of agreement with the measurements. However, the models using the global fitting strategy

consistently outperform those that rely on per-rate fitting of the service time. For instance, in the case of the Poisson distribution, mean values differ by only up to 1 packet when using a global fit, whereas differences of up to 8 packets are observed with a per-rate fit. This effect can be explained by the fact that the per-rate fitting strategy can suffer from performance issues when the model requires service time information for batches that have not been observed in the corresponding measurements. Hence, there is a trade-off between the amount of measurements that are used for fitting and the resulting accuracy. For the remainder of our evaluation, we show results that are obtained with the global fitting strategy.

5.1.2 Arrival Process Distribution. In contrast to the fitting strategy, the chosen distribution of the arrival process does not have a significant impact on the *mean batch size* returned by the model, which we can capture with the relative error (RE) of the normalized difference of means and is defined as

$$\frac{|\mathrm{E}[P] - \mathrm{E}[Q]|}{\mathrm{E}[P]}.$$

Therefore, we extend our evaluation and compare the *batch size distributions* that are returned for different arrival processes. To this end, we compare the batch size distribution returned by the model under different arrival processes to distributions observed in the testbed. Note that we do not modify the arrival process in the measurement setup, but are interested in identifying an appropriate interarrival time distribution for the model. We quantify the difference between the distribution that is returned by the measurements and the model by means of the Jensen-Shannon divergence (JSD) which is symmetric and bounded, allows to equally weight differences among two distributions $p(k)$ and $q(k)$ over their full support, and is defined as

$$\sum_{k=0}^{\infty} \left(\frac{1}{2} p(k) \ln \frac{p(k)}{\frac{1}{2}p(k) + \frac{1}{2}q(k)} + \frac{1}{2} q(k) \ln \frac{q(k)}{\frac{1}{2}q(k) + \frac{1}{2}p(k)} \right).$$

For three exemplary rates that represent a low, a medium, and a high load as well as our four arrival distributions in increasing order of coefficient of variation, Fig. 7 displays the batch size distribution obtained by means of measurements and our model. Given the batch size on the x-axes, the y-axes represent the corresponding probability while annotations provide the JSD and RE values. Note that all rows show the same values in case of the measurement-based distributions, since we only vary the interarrival time distribution used in the model.

When inspecting the distributions obtained by the measurements, we can observe that there is usually one peak around which the main portion of the probability mass is centered. This can be explained by the fact that there is an equilibrium between the per-packet service time that is achieved in the context of a particular batch size and the mean packet interarrival time. Hence, the number of arrivals during the service time of a batch is nearly constant. In the case of higher rates, shorter interarrival times lead to larger mean batch sizes which, in turn, allow for larger fluctuations in terms of the number of arrivals during the corresponding service time.

When comparing the subfigures column-wise, we observe that while these peaks are also reconstructed by all model variants, their dispersion increases significantly with the coefficient of variation of the chosen arrival distribution. Similarly to the previous argument, the higher variance of packet interarrivals leads to a wider range in terms of the number of arrivals during a service period. Finally, the best match regarding both the shape of the resulting distributions as well as the achieved JSD measure is achieved when using arrivals that follow a Poisson distribution[2]. This is also in line with the settings of the MoonGen traffic generator that is set to send packets at a constant rate. Since it is a software-based generator, minor fluctuations of the corresponding

[2]Note that arrivals do not follow a Poisson process, but exhibit interarrival times according to a Poisson distribution.

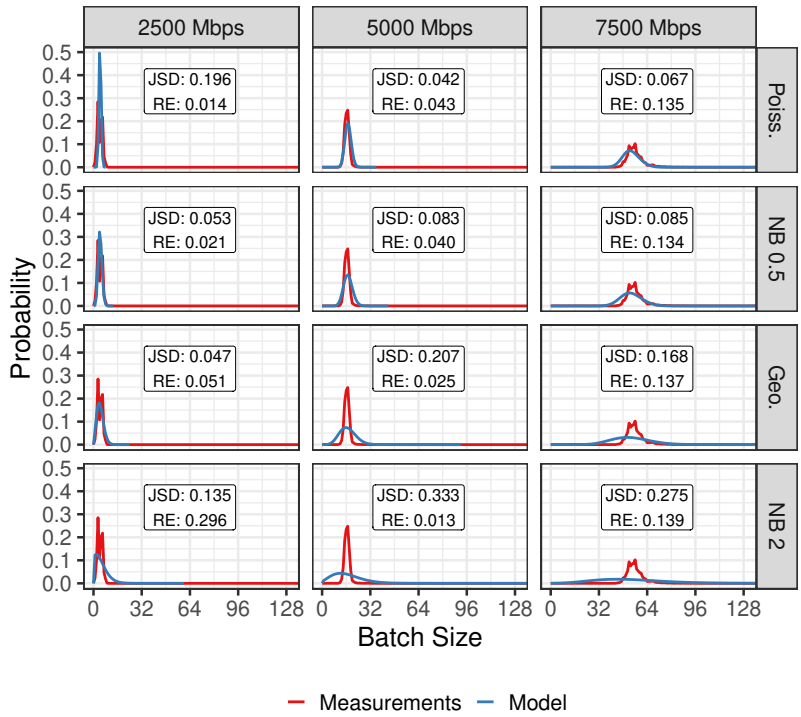


Fig. 7. Batch size distributions for different rates and arrival processes. Cross-connect scenario with $\beta = 256$, $L = 4096$, and 64 B packets.

Table 3. Packet loss probability for different rates. Cross-connect scenario with $\beta = 256$, $L = 4096$, and 64 B packets.

Rate [Mbps]	8000	8500	9000	9500	10000
Measurements	0.97%	6.69%	11.58%	16.44%	20.17%
Model	1.09%	6.78%	11.56%	16.30%	20.13%

sub-microsecond interarrival times are to be expected. Therefore, we use interarrival times that follow a Poisson distribution for the remainder of this work.

As already noted, the mean batch size takes on a constant value of 256 for rates of 8 Gbps and above. In these high-load regimes, packet loss begins to occur since the number of arrivals during the batch service time exceeds 256 and the queue fills up steadily. In Tab. 3, the actual packet loss that is reported in the measurements is compared to the model's predictions. Rates below 8 Gbps are omitted since they are equal to 0 in both cases. For the remaining rates, the model accurately predicts the occurrence and quantity of packet loss which increases linearly with the applied load.

In summary, our model achieves a very high accuracy for both batch size and packet loss in the cross-connect scenario, faithfully modeling I/O batching over a wide range of arrival rates, including overload scenarios that result in packet loss.

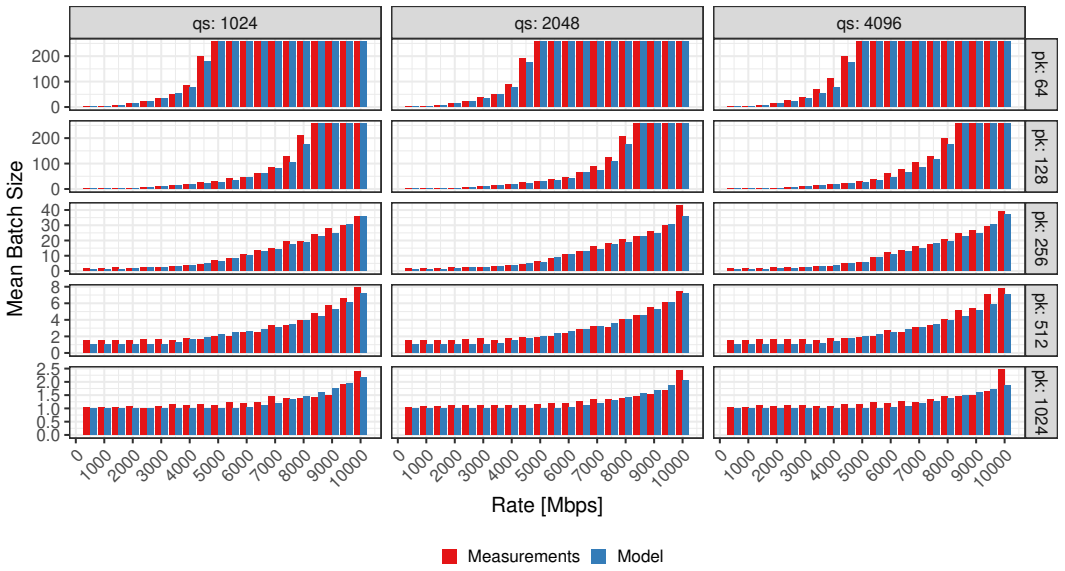


Fig. 8. Average batch size obtained via model and measurements given different packet (pk) and queue sizes (qs). Maximum batch size $\beta = 256$ and mixed traffic.

5.2 Prediction of Key Performance Indicators

Having demonstrated the model accuracy in the context of the simple cross-connect scenario and having identified appropriate model settings, we present the results of evaluations with the more complex scenario featuring mixed traffic in this section. Additionally, we investigate the impact of parameters such as the maximum batch size, the queue size, and the packet size on the system behavior as well as the accuracy of the model.

5.2.1 Batch Size Distribution. Since the overhead for processing a batch of packets is shared between packets within a batch, the batch size distribution constitutes an important measure of the system efficiency. We present a comparison between the mean batch sizes reported in our measurements and those predicted by our model in Fig. 8. Each subplot corresponds to a combination of queue and packet size, and displays the applied rate on the x-axis and the corresponding mean batch size on the y-axis.

First, we can extract insights regarding the system behavior. We can observe that the maximum batch size is attained earlier in the context of the more complex mixed traffic scenario than in the simple cross-connect scenario. While the top right subplot of Fig. 8 shows that this already happens at a rate of 5 Gbps with mixed traffic, a queue size of 4096, and 64 B packets, the maximum batch size is reached starting at a rate of 8 Gbps when using the same parameters in the cross-connect case (cf. Fig. 6. Furthermore, increasing the packet size leads to a decrease of batch sizes as evidenced by the development of batch sizes along vertical sequences of subplots. This can be explained by the fact that processing happens per header rather than per byte and therefore an increase in packet size results in a decrease of the packet-rate at the same bitrate. From the vertical sequences of subplots, we can derive that the queue size does not have a significant impact on the batch size. To explain this, we can consider the two extreme operational regimes the system can be in. If it is in the loss-free regime, the queue is emptied on each batch pickup event and never runs full. In conditions with packet loss, the queue tends to fill up regardless of its size, leading to exclusively

full batches. The behavior inbetween these two scenarios is, in general, depending on the arrival process and, more specifically, its burstiness. Hence, while the maximum queue size does not have an immediate impact on batch sizes, it should not be ignored since it does affect waiting times and can allow the system to withstand packet bursts.

Finally, comparing measurement values with model-based estimates demonstrates that despite the increased complexity of the scenario, the model is still capable of reliably predicting the mean batch size. In 90% of scenarios, the difference w.r.t. mean batch size is 2 or less. Nevertheless, some outliers with larger differences are present. These tend to occur on two occasions. First, during the transition from the loss-free regime to the lossy regime. Second, when the applied rate equals the maximum rate of the link. Both constellations represent conditions with an increased sensitivity where deviations are amplified.

The transition between the loss-free and the lossy regime can be observed in Fig. 9, which shows the difference between the experimentally measured batch sizes and the predictions of the model. Considering the first row (obtained with mixed traffic of 64 B packets) we observe that at high-rate, the difference is zero: the model correctly predicts the saturation of the system which will always retrieve batches of the maximum size β. At low rate, the difference between the model and the measurements is very low, and it increases as we approach the state change between a loss-less and a lossy regime. When the system load reaches this state change (between 3.5 Gbps and 5.0 Gbps) we observe that the model underestimates the actual size of the batches. This is due to non-linear effects introduced by the implementation of the VNF router, as the program tries to privilege larger batches in order to minimize the overhead of the framework, thus causing a discrepancy of up to 30 packets for the batch size. Similarly, this also explains the behavior observed in the second row of Fig. 9. Interestingly, we observe that when the packet size is 64 B (first row), the interval of the state change is [3.5, 5.0] Gbps, which translates into an interval of [5.2, 7.4] Mpps (millions of packets per second). When the packet size is 128 B, the state-change interval is [6.0, 8.5] Gbps, which translates into an interval of [5.1, 7.2] Mpps. Therefore, although the second interval is larger in bitrate, it is comparable in terms of processed packets per second. Finally, as the scenarios with packet size greater than 256 B never show a change of regime, we do not observe a significant difference between the measurements and the model.

While the mean batch size provides aggregated information on the system efficiency, our model is also capable of returning the batch size distribution which allows an even more in-depth analysis. We use the Jensen-Shannon divergence (JSD) to quantify the difference between the batch size distribution that is obtained via the model with the measurement-based reference and present the results in Fig. 10. Based on a total of 1200 parameter combinations of queue size, packet size, maximum batch size, and applied rate for each of the two scenarios, empirical cumulative distribution functions of attained JSD values are shown. We can identify a gap between the curves representing the cross-connect (XC) and mixed traffic (MIX) scenarios, with lower JSD values observed in the case of the former. This is in line with previous observations, i.e., the mixed traffic scenario introduces additional complexity which increases the difficulty of the prediction task.

Since the JSD captures differences over the entire range of the distribution, even slight changes in the mean value can have a large impact on the resulting JSD value despite similarities in terms of the shape of distributions. Nonetheless, in over 15% of cases, a perfect match between distributions is achieved. Furthermore, JSD values lower than 0.1 are observed in 67.4% and 63.2% of parameter combinations of the XC and MIX scenarios, respectively. Higher JSD values stem from parameter constellations that also caused the largest deviations regarding the mean batch size, i.e., those near the transition towards the lossy regime and rates equal to the capacity of the physical link.

In summary, the model accurately captures the mean batch size as well as the batch size distribution, even when the scenario complexity is increased by changing VNF behavior, traffic mix, and parameters

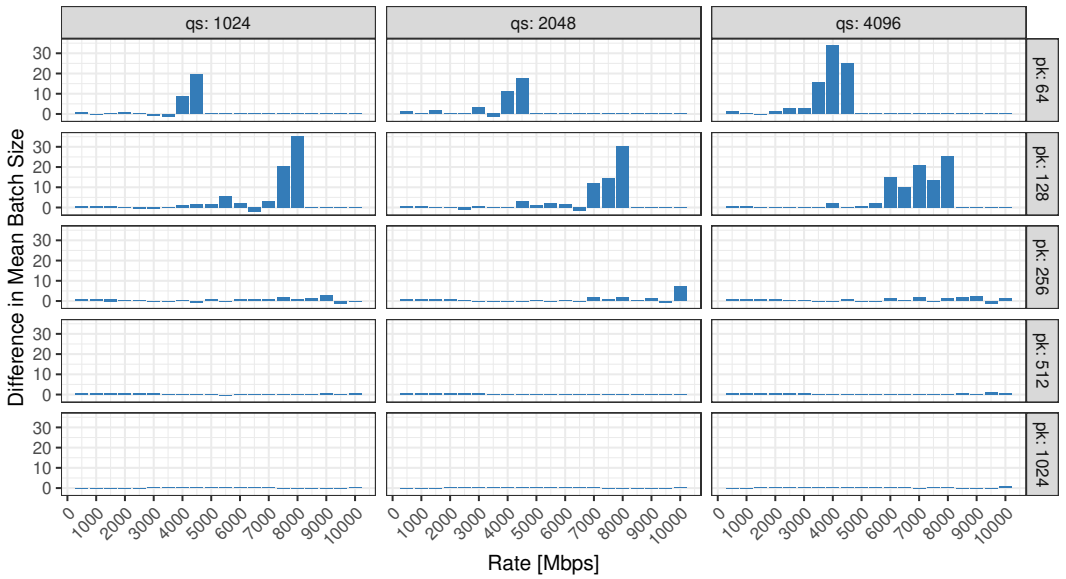


Fig. 9. Difference in mean batch sizes obtained via model and measurements given different packet (pk) and queue sizes (qs). Maximum batch size $\beta = 256$ and mixed traffic. Values greater than 0 correspond to the mean batch size in the measurements being larger than the one returned by the model.

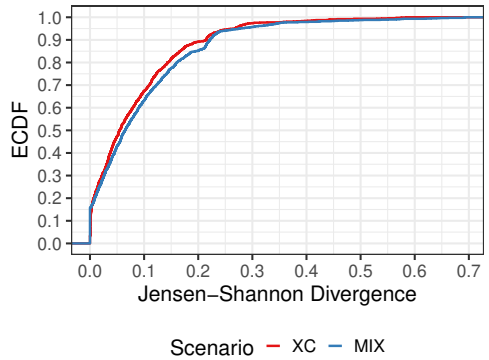


Fig. 10. Distribution of the Jensen-Shannon divergence regarding the batch size distributions obtained from measurements and via the model. All scenarios are included, i.e., queue size $L \in \{1024, 2048, 4096\}$, maximum batch size $\beta \in \{64, 256, 512, 1024\}$, packet size $\in \{64, 128, 256, 512, 1024\}$ B, and applied rate $\in \{500, 1000, \ldots, 10000\}$ Mbps.

such as the packet size, queue size, and maximum batch size. The compatibility with these scenarios is maintained without modifications to the model, highlighting its general applicability.

5.2.2 Waiting and Processing Time Distributions. While the batch size can serve as an efficiency indicator, large batches can also adversely affect the waiting and processing time of packets. Hence, these metrics should also be considered when evaluating the performance of VNFs. With our model, we can derive the waiting time as well as the processing time distributions and we validate the results in this section.

Since performing per-packet delay measurements in the device under test would interfere with VNF performance - especially at high packet rates - it is not a feasible strategy for obtaining the ground-truth latency data. Instead, we measure the total per-packet latency between the egress and ingress of our MoonGen traffic generator and compare it to the processing time that is determined via our model. Due to this measurement setup, we expect two main sources of mismatch between the experimental results and our model. Whereas the model targets internal DUT processing, measurements are taken externally. Hence, the measurements include DUT processing as well as additional delays induced by traffic generator processing and other overheads. Similarly, propagation and transmission delay between the generator and the DUT are not explicitly accounted for in the model.

As such, we expect the model to underestimate delay-related KPIs when compared to measurement values. However, due to the nature of the aforementioned overhead, this difference should result in a constant, scenario-specific *fixed delay offset* which can be addressed with appropriate calibration. In our case, this fixed delay offset encompasses the MoonGen processing time for packet handling, the round-trip propagation, and transmission delays on wire. We quantify this offset by means of a simple cross-connect setup in which the DUT simply executes DPDK L2 forwarding to minimize processing. In this scenario, we once again use MoonGen to generate a continuous stream of packets, that get forwarded back to the source by the L2 forwarding VNF. In our specific testbed configuration, this overhead amounts to roughly 5 microseconds and is later used in Figure 13 to present adjusted model predictions.

For different rates on the x-axis, Fig. 11 shows a comparison of the mean latency in microseconds as obtained from measurements and our model. Horizontally and vertically arranged subplots illustrate the effects of changes to the queue size and packet size, respectively. We limit the y-axis to a maximum of 10 μs in order to show details during the loss-free operational regime. As soon as the load increases and the system transitions into the lossy regime, the latency increases significantly due to congestion at the queue.

We make three main observations. First, for all shown scenarios, both the measurement- and the model-based values follow the same trend, i.e., latency increases happen at the same locations and with a similar slope. Furthermore, the offset between the two curves remains in a narrow range around 5 μs. Second, there is a clear effect of the packet size on the waiting time. Following subfigures along the vertical axis, we can observe that the latency starts increasing earlier in the case of small packets. This is in line with previous observations about higher packet rates when using smaller packets at the same bitrate. Third, almost no difference is observed regarding the mean latency in scenarios that differ only in the queue size. This effect is caused by limiting the y-axis and showing only the loss-free portion of the scenarios. In those, no congestion at the queues takes place and the queue is emptied on each batch pick-up. Beyond these rates, the queue size does play a role and limits the maximum waiting time, i.e., rather than having to wait longer, packets are dropped in the case of a smaller queue size.

In order to quantify the difference between latency values obtained from the measurements and the model as well as to investigate the behavior in high-load scenarios, Fig. 12 displays the cumulative distribution function of the latency differences observed in all scenarios, i.e., all combinations of queue and packet size, maximum batch size, and XC / MIX scenarios. The logarithmically scaled x-axis shows the difference between measurement and model values, whereas the y-axis denotes the percentage of experiments in which the difference was below of equal to the corresponding value.

In the case of the XC scenario, the difference is in the range of 3-5.8 μs in 92% of experiments, and in the range of 3-6.4 μs in 75% of MIX experiments. These correspond to the loss-free cases shown in the previous figure. In cases of overload, both the absolute values of the latency and the

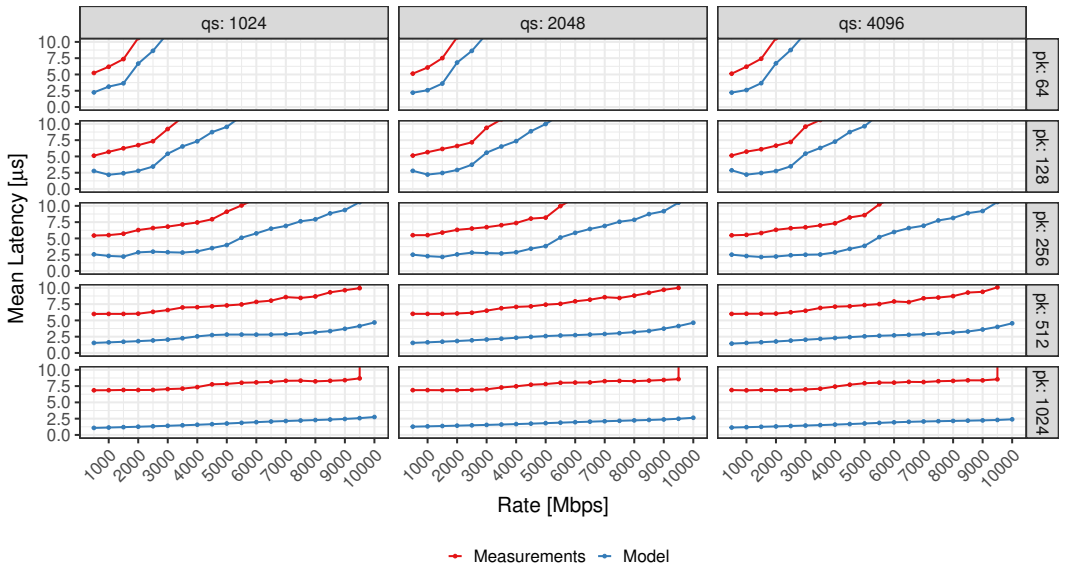


Fig. 11. Mean latency measured in the technical system and reported by the model under different packet (pk) and queue sizes (qs). Maximum batch size $\beta = 256$ and mixed traffic.

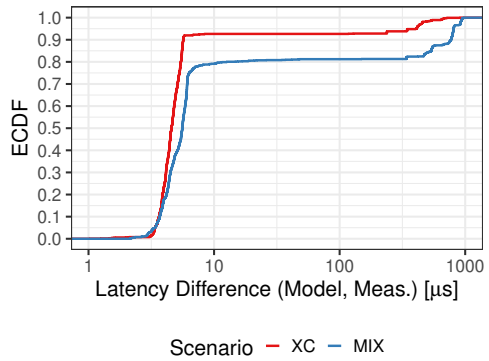
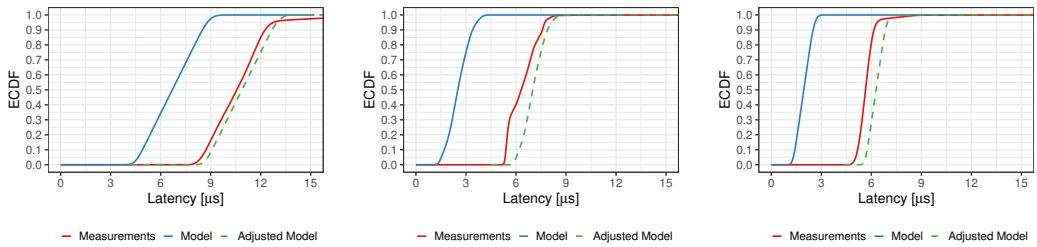


Fig. 12. Distribution of the latency difference between measurements and the model. All scenarios are included, i.e., queue size $L \in \{1024, 2048, 4096\}$, maximum batch size $\beta \in \{64, 256, 512, 1024\}$, and packet size $\in \{64, 128, 256, 512, 1024\}$ B.

latency difference increase significantly and are in the range of 400-900 μs. However, as evidenced by the vertical sections in the ECDF curve, the difference remains stable within scenarios with identical parameters and could be partially mitigated by calibration. Additionally, in the overload case, congestion and packet drops at the queue play a role and our simplified modeling of ring buffers as FIFO queues starts having an adverse effect on the accuracy of latency predictions. However, when the system is in a lossy-regime, the waiting time is not the right KPI to consider, as the system operator should be more concerned with the packet loss ratio, which the model can correctly predict. The reason for the better overall performance in the XC experiments is caused not only by the lower complexity of the system, but also by the fact that XC has a higher non-drop rate and therefore a larger proportion of loss-free scenarios.



(a) Mixed traffic, $\beta = 256$, 64 B packets, $L = 4096$, 2000 Mbps.

(b) Mixed traffic, $\beta = 256$, 256 B packets, $L = 2048$, 2000 Mbps.

(c) Cross-connect, $\beta = 256$, 64 B packets, $L = 4096$, 3000 Mbps.

Fig. 13. Latency distributions in different scenarios obtained via measurements and model. Dashed green curves are obtained by shifting the original model curves by a constant offset that is obtained via calibration measurements.

Despite the offset regarding the mean latency, we investigate the latency distribution to check whether the model can faithfully reproduce the general system behavior, e.g., regarding the shape of the distribution. To this end, we present cumulative distribution functions for three different parameter combinations in Fig. 13. Each subfigure corresponds to one parameter combination and we vary the traffic conditions, queue size, packet size, and applied packet rate. The measurement values are shown in red, whereas the green and blue lines indicate the model values with and without calibration, respectively. In particular, the adjusted model curves are obtained by using the mean offset from the calibration measurements to shift the original distributions that are returned by the model.

In case of all three combinations, we can observe that the shape of the distribution is retained and that in case of both the measurements and the model, the overall processing time roughly follows a uniform distribution. This is consistent with the fact that packets experience different waiting times depending on their time of arrival relative to the next batch pick-up event and arrive at the system at a near-constant rate.

When comparing the first two subplots, we notice that the latency ranges differ despite the respective scenarios having the same applied rate of 2000 Mbps. This phenomenon is explained by the different packet size which in turn affects the batch size and therefore batch service time, leading to a lower total processing time in case of the scenario with larger packets in Fig. 13b. Furthermore, the latency values shown in Fig. 13c are even lower and lie within a narrower range. This stems from the simple cross-connect scenario in which a higher rate can be sustained due to more heterogeneous packets and absence of table look-ups during packet handling.

Finally, when looking at the adjusted model values obtained by adding the constant offset obtained during calibration measurements, the close fit of the model can be seen.

In summary, the results presented in this subsection highlight that the model is also capable of accurately reproducing the behavior of VNFs in terms of the latency that is experienced by packets. Furthermore, this capability persists throughout numerous parameters and therefore shows that the model generalizes well.

6 RELATED WORK

Related to our work is both *modeling work* [1, 17, 22, 27, 30, 36, 38, 39, 41, 51, 53] that either shares a similar technique or NFV focus and *experimental work* [7, 18, 20, 26, 28, 29, 42, 44, 49] of NFV systems. Our work is, to the best of our knowledge, the first attempt at bridging the gaps between

these two worlds by proposing a strongly grounded theoretical model, that is directly compared to experimental results of state of the art NFV software with batch processing.

6.1 Modeling Viewpoint

The theory of *bulk* queueing systems has long been studied [27]. For Markovian bulk input $M^{[X]}/M/1$ and service $M/M^{[X]}/1$ systems, [51] provides closed form solutions under Poisson arrivals and exponentially distributed service times. Particularly, *bulk-input* Batch Markovian Arrival Processes (BMAP) have been well studied [38, 39, 50], and applied to study long lived TCP connections [1, 41], model aggregated IP traffic [30] or describe parallel processing in cloud environments [23]. Similarly, models featuring batch arrivals and general independent arrival distributions have been proposed as well [12, 13].

Furthermore, several studies regarding *bulk-service* systems have been conducted in the past [2, 9, 11, 45]. Similarly, previous work that takes batch-size dependent service times into account does exist as well [3, 14, 40]. However, the complexity of the relation between batch-size and service time is limited in these studies. This relation between batch-size and service time can be arbitrarily complex for the model proposed in this work. Finally, [4, 10] both investigate $M/Gi_r^{(a,b)}/1$ queues and compute the queue size distribution at departure events. Note here that all of these studies contain at least one Markov component or are limited to basic performance indicators, as opposed to the $Gi/Gi^{[X]}/1-L$ study presented in this work.

Models of Network Functions Virtualization (NFV) have also recently appeared [17, 22, 36, 53]. In particular, queueing models are used in [17] and [36] to describe software-based networks. Similarly, the authors of [43, 46, 47] investigate the impact of autoscaling on 5G networks with both legacy equipment and VNFs as an $M/M/n$ system with variable n. All of these models adopt a global network view and strongly abstract the mechanisms of specific network elements by simply assuming a certain service rate, as opposed to this work in which we provide a detailed model of a single VNF component. Under this perspective, studies closer to ours are [22, 53], which both aim at predicting virtual function performance on multi-core systems. Yet, [53] does not take into account mechanisms like batch arrival or batch processing of packets, which both are crucial characteristics of modern NFV routers. In contrast, the authors of [22] assume fixed processing times, which we show not to hold true in practice, and omit a proper experimental validation.

In synthesis, while several models exist that take bulk arrival as well as batch service processes into account, evaluations of real world systems are missing. In addition, most solutions are based on the Markovian property of a system, which does not necessarily hold true in the real world. In addition, related approaches in the area of NFV often exploit a high level of abstraction by ignoring details of the software stack like batch processing, interrupt mitigation and busy polling mechanisms. Finally, proper validation of the model outputs based on a comparison to experimental results of a real NFV system is lacking so far.

6.2 Experimental Viewpoint

The ecosystem of high-speed all-software packet processing has flourished in the last decade with both low-level building-blocks that use I/O batching (e.g., netmap [49] and DPDK [26]), as well as high-level full-blown stacks that apply NFV functions with a compute batching paradigm [7, 18, 28, 29]. Whereas such frameworks offer a similar set of features, comparison is difficult so that most related work relies on extensive evaluation campaigns of a single tool – as we do in this work using VPP over DPDK.

Previous efforts aimed to evaluate a limited subset of the aforementioned tools [7, 20, 44]. For example, [20] focuses on accelerated low-level frameworks, namely netmap, DPDK, and `PF_RING`.

The authors perform an experimental campaign assessing not only throughput, measured in Mpps, but also consider the impact of factors such as batch size or misses in CPU caches. Similarly, FastClick performance is evaluated over both DPDK and netmap in [7]. Finally, [44] experimentally compares NFV throughput with chains of heterogeneous functions using OVS-DPDK, SR-IOV, and FD.io VPP. Given findings in [7, 20, 44], it is reasonable to assume that the model presented in this paper should also be fit to express the performance of other frameworks – whose application however requires involving experts in each of the above tools, since the engineering effort to put in place measurement collection in a state of the art manner is delicate process requiring a considerable effort.

A recent direction advocates for a more general approach at the evaluation of software routers, and for the availability for open and honest quantification of novel tools' performance. Authors in [54] propose a methodology to fairly assess the performance of several state-of-the-art software routers in different settings, while the online reports of [19] show the results of several throughput/latency measurements for the latest versions of VPP. Finally, pointed out in [16], the topic of fairly measure the performance of software routers is delicate and difficult, which further proves the need for analytical approaches such as the one proposed in this manuscript, alongside the classical experimental benchmarking. As such, to the best of our knowledge, our work is the first to bridge the gap between experimental and analytical work of NFV by proposing a simple yet accurate model, that we studied and benchmarked for a particular software tool, but that can be easily fit to any other tool in the NFV family.

7 CONCLUSION

This paper presents the first discrete-time NFV model that takes into account the most recent and relevant aspects of modern NFV routers. These include the use of batching for both low-level I/O data transfer as well as for high-level data transformation and computation. We validate the model with experimental results that are gathered in a testbed with state-of-the art NFV routers. The experimental scenarios include a simple cross-connect case as well as a realistic setting in which traffic triggers heterogeneous functions with different processing complexity.

While our proposed model is simple and general, as it only needs few aggregated measurements from a real NFV router, it is very accurate in reporting detailed performance indicators even in complex scenarios with multiple functions. The performance indicators include not only the packet loss probability and mean batch size, but also the distribution of the batch size as well as the processing time distribution. On the one hand, this allows to precisely characterize the router's performance, e.g., in terms of batching delay. On the other hand, it can be used as an operational tool to dimension the router hardware, e.g., the number of CPU cores required to sustain mixed traffic with a classic 5-nines reliability.

As part of a larger effort, we release our measurement at [33] in the hope that, at a community level, we can build a large benchmark for NFV models including a larger set of NFV routers (such as FastClick and G-opt) as well as more realistic traffic patterns (e.g., different traffic mixtures, chains of functions of different lengths).

REFERENCES

[1] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. 2000. A stochastic model of TCP/IP with stationary random losses. *ACM SIGCOMM Computer Communication Review* (2000).

[2] Norman TJ Bailey. 1954. On queueing processes with bulk service. *Journal of the Royal Statistical Society: Series B (Methodological)* 16, 1 (1954), 80–87.

[3] Anuradha Banerjee, Umesh Chandra Gupta, and Srinivas R Chakravarthy. 2015. Analysis of a finite-buffer bulk-service queue under Markovian arrival process with batch-size-dependent service. *Computers & Operations Research* 60 (2015), 138–149.

[4] Shaul K Bar-Lev, Mahmut Parlar, David Perry, Wolfgang Stadje, and Frank A Van der Duyn Schouten. 2007. Applications of bulk queues to group testing models with incomplete identification. *European Journal of Operational Research* 183, 1 (2007), 226–237.

[5] David Barach, Leonardo Linguaglossa, Damjan Marion, Pierre Pfister, Salvatore Pontarelli, and Dario Rossi. 2018. High-Speed Software Data Plane via Vectorized Packet Processing. *IEEE Communications Magazine* 56, 12 (December 2018), 97–103. https://doi.org/10.1109/MCOM.2018.1800069

[6] Tom Barbette, Georgios P Katsikas, Gerald Q Maguire Jr, and Dejan Kostić. 2019. RSS++ load and state-aware receive side scaling. In *Proceedings of the ACM International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*. 318–333.

[7] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast userspace packet processing. In *Proceedings of the Symposium on Architectures for Networking and Communication Systems (ANCS)*.

[8] Scott Bradner and Jim McQuaid. 1999. RFC2544 Benchmarking Methodology for Network Interconnect Devices. https://www.ietf.org/rfc/rfc2544.txt. (1999).

[9] Mohan L Chaudhry and Seok Ho Chang. 2004. Analysis of the discrete-time bulk-service queue Geo/GY/1/N+ B. *Operations Research Letters* 32, 4 (2004), 355–363.

[10] Mohan L Chaudhry and Jing Gai. 2012. A Simple and Extended Computational Analysis of M/G j (a, b)/1 and M/G j (a, b)/1/(B+ b) Queues Using Roots. *INFOR: Information Systems and Operational Research* 50, 2 (2012), 72–79.

[11] Mohan L Chaudhry and UC Gupta. 2003. Analysis of a finite-buffer bulk-service queue with discrete-Markovian arrival process: D-MAP/Ga, b/1/N. *Naval Research Logistics (NRL)* 50, 4 (2003), 345–363.

[12] Mohan L Chaudhry and Umesh Chandra Gupta. 1997. Queue-length and waiting-time distributions of discrete-time GI X/Geom/1 queueing systems with early and late arrivals. *Queueing Systems* 25, 1 (1997), 307–324.

[13] Mohan L Chaudhry, Umesh Chandra Gupta, and Veena Goswami. 2001. Modeling and analysis of discrete-time multiserver queues with batch arrivals: GIX/Geom/m. *INFORMS Journal on Computing* 13, 3 (2001), 172–180.

[14] Guy L Curry and Richard M Feldman. 1985. An M/M/1 queue with a general bulk service rule. *Naval research logistics quarterly* 32, 4 (1985), 595–603.

[15] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A scriptable high-speed packet generator. In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

[16] Vivian Fang, Tamás Lvai, Sangjin Han, Sylvia Ratnasamy, Barath Raghavan, and Justine Sherry. 2018. Evaluating Software Switches: Hard or Hopeless? *Berkeley Technical Report No. UCB/EECS-2018-136* (2018).

[17] Giuseppe Faraci, Alfio Lombardo, and Giovanni Schembra. 2017. A building block to model an SDN/NFV network. In *Proceedings of the IEEE International Conference on Communications (ICC)*.

[18] fd.io. 2016. VPP whitepaper. https://fd.io/wp-content/uploads/sites/34/2017/07/FDioVPPwhitepaperJuly2017.pdf. (2016).

[19] fd.io. 2020. CSIT-2005 - Technical report. (2020). https://docs.fd.io/csit/rls2005/report/vpp_performance_tests/ Last accessed: 2021-05-25.

[20] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. 2015. Comparison of frameworks for high-performance packet IO. In *Proceedings of the Symposium on Architectures for Networking and Communication Systems (ANCS)*.

[21] Steffen Gebert, Thomas Zinner, Stanislav Lange, Christian Schwartz, and Phuoc Tran-Gia. 2016. *Discrete-Time Analysis: Deriving the Distribution of the Number of Events in an Arbitrarily Distributed Interval.* Technical Report 498. University of Wuerzburg.

[22] Steffen Gebert, Thomas Zinner, Stanislav Lange, Christian Schwartz, and Phuoc Tran-Gia. 2016. Performance Modeling of Softwarized Network Functions Using Discrete-Time Analysis. In *Proceedings of the 28th International Teletraffic Congress (ITC)*.

[23] Fabrice Guillemin, Veronica Quintuna Rodriguez, and Alain Simonian. 2019. A Processor-Sharing model for the Performance of Virtualized Network Functions. In *Proceedings of the 31st International Teletraffic Congress (ITC)*.

[24] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: a GPU-accelerated software router. (2010).

[25] Tom Herbert and Willem de Bruijn. 2011. Scaling in the linux networking stack. https://www.kernel.org/doc/Documentation/networking/scaling.txt. (2011).

[26] Intel. 2021. Data Plane Development Kit. http://dpdk.org. (2021).

[27] Irwin W Kabak. 1970. Blocking and delays in M (x)/M/c bulk arrival queueing systems. *Management Science* (1970).

[28] Anuj Kalia, Dong Zhou, Michael Kaminsky, and David G Andersen. 2015. Raising the Bar for Using GPUs in Software Packet Processing.. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[29] Joongi Kim, Seonggu Huh, Keon Jang, KyoungSoo Park, and Sue Moon. 2012. The power of batching in the click modular router. In *Proceedings of the Asia-Pacific Workshop on Systems*.

[30] Alexander Klemm, Christoph Lindemann, and Marco Lohmann. 2003. Modeling IP traffic using the batch Markovian arrival process. *Performance Evaluation* (2003).
[31] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and Frans Kaashoek. 1999. The Click Modular Router. *Operating Systems Review* (1999).
[32] Stanislav Lange, Leonardo Linguaglossa, Stefan Geissler, Dario Rossi, and Thomas Zinner. 2019. Discrete-Time Modeling of NFV Accelerators that Exploit Batched Processing. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*.
[33] Stanislav Lange, Leonardo Linguaglossa, Stefan Geissler, Dario Rossi, and Thomas Zinner. 2021. Dataset repository. https://github.com/lsinfo3/2020-tompecs-vpp-data. (2021).
[34] Leonardo Linguaglossa, Stanislav Lange, Salvatore Pontarelli, Gabor Retvari, Dario Rossi, Thomas Zinner, Roberto Bifulco, Michael Jarschel, and Giuseppe Bianchi. 2019. Survey of Performance Acceleration Techniques for Network Function Virtualization. In *Proceedings of the IEEE*.
[35] Leonardo Linguaglossa, Dario Rossi, Salvatore Pontarelli, Dave Barach, Damjan Marjon, and Pierre Pfister. 2019. High-speed data plane and network functions virtualization by vectorizing packet processing. *Computer Networks* 149 (2019), 187–199.
[36] Alfio Lombardo, Antonio Manzalini, Vincenzo Riccobene, and Giovanni Schembra. 2014. An analytical tool for performance evaluation of software defined networking services. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*.
[37] Robert Love. 2010. *Linux kernel development*. Pearson Education.
[38] David M Lucantoni. 1991. New results on the single server queue with a batch Markovian arrival process. *Communications in Statistics. Stochastic Models* (1991).
[39] David R Manfield and P Tran-Gia. 1982. Analysis of a finite storage system with batch input arising out of message packetization. *IEEE Transactions on Communications* (1982).
[40] Marcel F Neuts. 1987. Transform-free equations for the stationary waiting time distributions in the queue with Poisson arrivals and bulk services. *Annals of Operations Research* 8, 1 (1987), 1–26.
[41] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. 1998. Modeling TCP throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review* (1998).
[42] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.
[43] Tuan Phung-Duc, Yi Ren, Jyh-Cheng Chen, and Zheng-Wei Yu. 2016. Design and analysis of deadline and budget constrained autoscaling (DBCA) algorithm for 5G mobile networks. In *Proceedings of the IEEE international conference on cloud computing technology and science (CloudCom)*. IEEE, 94–101.
[44] Nikolai Pitaev, Matthias Falkner, Aris Leivadeas, and Ioannis Lambadaris. 2018. Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*.
[45] Warren B Powell and Pierre Humblet. 1986. The bulk service queue with a general control strategy: theoretical analysis and a new computational procedure. *Operations Research* 34, 2 (1986), 267–275.
[46] Yi Ren, Tuan Phung-Duc, Jyh-Cheng Chen, and Zheng-Wei Yu. 2016. Dynamic auto scaling algorithm (dasa) for 5g mobile networks. In *Proceedings of the IEEE global communications conference (GLOBECOM)*. IEEE, 1–6.
[47] Yi Ren, Tuan Phung-Duc, Yi-Kuan Liu, Jyh-Cheng Chen, and Yi-Hao Lin. 2018. ASA: Adaptive VNF scaling algorithm for 5G mobile networks. In *Proceedings of the IEEE International Conference on Cloud Networking (CloudNet)*. IEEE, 1–4.
[48] Luigi Rizzo. 2001. Device polling support for FreeBSD. In *BSDConEurope Conference*.
[49] Luigi Rizzo. 2012. netmap: a novel framework for fast packet I/O. In *Proceedings of the USENIX Annual Technical Congress (ATC)*.
[50] Veronica Quintuna Rodriguez and Fabrice Guillemin. 2018. Cloud-RAN modeling based on parallel processing. *IEEE Journal on Selected Areas in Communications* 36, 3 (2018), 457–468.
[51] John F Shortle, James M Thompson, Donald Gross, and Carl M Harris. 2018. *Fundamentals of queueing theory*. John Wiley & Sons.
[52] William J Stewart. 2009. *Probability, Markov chains, queues, and simulation*. Princeton university press.
[53] Kalika Suksomboon, Masaki Fukushima, Shuichi Okamoto, and Michiaki Hayashi. 2016. A dilated-CPU-consumption-based performance prediction for multi-core software routers. In *IEEE NetSoft Conference and Workshops (NetSoft)*.
[54] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, Luigi Iannone, and James Roberts. 2019. Comparing the performance of state-of-the-art software switches for NFV. In *Proceedings of the ACM International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*.