Discrete-Time Modeling of NFV Accelerators that Exploit Batched Processing

Authors omitted due to double blind review

***Abstract*—Network Functions Virtualization (NFV) is among the latest network revolutions, bringing flexibility and avoiding network ossification. At the same time, all-software NFV implementations on commodity hardware raise performance issues with respect to ASIC solutions. To address these issues, numerous software acceleration frameworks for packet processing have appeared in the last few years. Common among these frameworks is the use of *batching techniques*. In this context, packets are processed in groups as opposed to individually, which is required at high-speed to minimize the framework overhead, reduce interrupt pressure, and leverage instruction-level cache hits.**

Whereas several system implementations have been proposed and experimentally benchmarked, 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 fill this gap by proposing a simple generic model for such batching-based mechanisms, which allows a very detailed prediction of highly relevant performance indicators. These include the distribution of the processed batch size as well as queue size, which can be used to identify loss-less operational regimes or quantify the packet loss probability in high-load scenarios. 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.

***Index Terms*—Discrete-Time Model, Queueing Theory, NFV, DPDK, netmap, FD.io, VPP.**

I. 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 [1] 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 [2], [3], 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) [4] and Vector Packet Processor (VPP) [5] are two examples.

To achieve high-speed processing, these software frameworks share commonalities [6] such as the use of lock-free multi-threading as well as the use of *poll-mode batched processing*. If 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 [2], [3], as well as for processing batches of packets in higher-level applications to amortize framework overhead [5]–[8].

Yet, while a large number of system *implementations* exist, and while some work recently started undertaking an experimental comparison of these implementations [6], [9], [10], to the best of our knowledge a system *model* that can explain and accurately predict the measurable system performance of such batch-based packet processors has yet to appear. Although a model for VNF processing times is proposed in [11], 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 [12]–[15], both the use case and the particular processing schemes differ significantly.

This paper presents the first simple yet accurate model of high-speed software routers using batching acceleration. We present a general model that is able to accurately characterize the most distinctive parameters of new-generation software routers, including the packet loss probability and processed batch size. In particular, the model allows deriving the full distribution of the batch size and not just the first few moments. Experiments with a real software router confirm the model to be very accurate in realistic scenarios where a mixture of network functions with different levels of complexity is present and each requires a different number of CPU cycles.

In the remainder of this paper, we first introduce the architecture of a modern NFV software router in Sec. II. We develop a discrete-time queuing model in Sec. III, after which we describe the experimental setup that is used for the model validation in Sec. IV. Results of the validation are presented in Sec. V. Finally, we put this work in the context of related efforts in Sec. VI and summarize our findings in Sec. VII.

II. 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. IV. The Device Under Test (DUT) consists in 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, intrusion detection, 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.

A. 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 the Direct Memory Access (DMA) technique, 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 the newcomers. 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 the Receive Side Scaling (RSS) technique [16] to bind different CPUs 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 with a separate instance of the software router, managing its own specific RSS queue with its own packet ring. Since the RSS techniques 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 in the number of cores. Hence, for modeling purposes, it is sufficient to focus on a single RSS queue.

B. Polling and I/O Batching

Traditionally, the networking stack generated an *interrupt* every time a new packet was 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 has been proven to be very inefficient and to overload the CPU, for which different interrupt mitigation mechanisms have been introduced. One such mechanism is *polling* [17]: at very high traffic rates, the CPU continuously checks for packets stored in the packet ring without raising any interrupt.

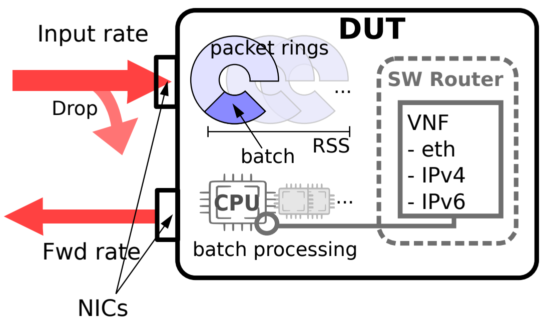

Fig. 1: Synopsis of the device under test considered in this manuscript.

Polling mechanisms are typically coupled with *batching*: 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 the overall processing, as it amortizes the fixed costs of the I/O over multiple packets [7], [18] and is as such supported by all modern networking stacks [2], [3].

A maximum batch size β is usually defined to fix a 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 $[0, \beta]$. Thus, defining a *simple* model capable of faithfully representing batched operations is a relevant goal.

C. Compute Batching

Furthermore, the use of batching is not limited to packet I/O. Indeed, network function computation can similarly benefit from grouped processing, which is known as *compute batching*.

Shortly, when a VNF is executed over a batch, this allows sharing the overhead of the packet processing frameworks on 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 [8], DoubleClick [7], FastClick [6] and VPP [5]), 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 another relevant goal.

III. 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

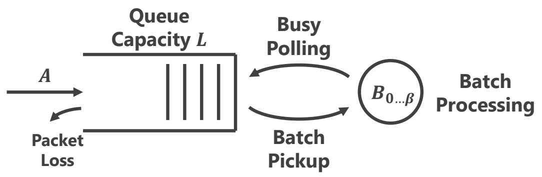

Fig. 2: Model overview.

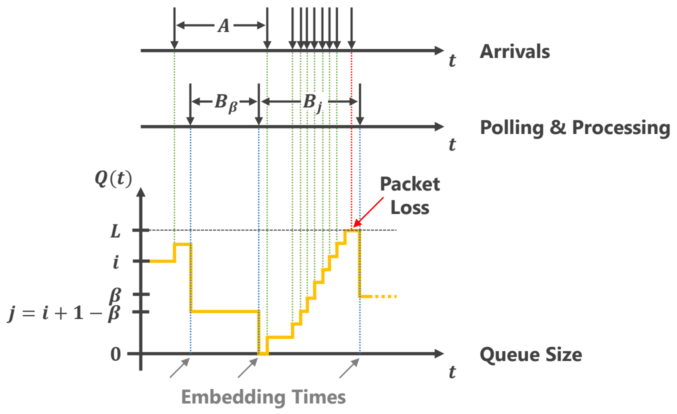

Fig. 3: Exemplary state development of the model.

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. V). 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.

A. Discrete-Time Model

For the sake of readability, we provide an overview of the notation used in this manuscript in Tab. I. 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

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

and the corresponding distribution function is defined as

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

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 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* $Q(k)$ can be used to derive all relevant performance indicators for the modeled system, e.g., the batch size distribution as well as the packet loss probability.

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 Q 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 $q(k)$. 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.

TABLE I: 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.
$x_{\tau,a}(k)$	Number of arrivals whose interarrival time is distributed according to a during an interval whose length is distributed according to τ.
Q_n, $q_n(k)$	Queue size immediately before the n-th batch is picked up.
Q, $q(k)$	Queue size at embedding times.
V_n, $v_n(k)$	Batch size immediately before the n-th batch is picked up.
V, $v(k)$	Batch size at embedding times.
p_{loss}	Packet loss probability.

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

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 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 - which is distributed according to $b_{\min(i,\beta)}$ - by means of $x_{b_{\min(i,\beta)},a}$. 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.

Finally, we remark that under stationary conditions, the indexes n and $(n + 1)$ in (1) can be suppressed, i.e.,

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

B. 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.

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.

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

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}:

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

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 j, 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.

IV. Experimental Setup

To validate our model, we instrument a testbed operating a real NFV software router following the IETF benchmarking guidelines [19]. This section describes our hardware and software setups as well as the scenarios we use to assess the accuracy of our model.

A. Hardware Setup

We reproduce the experimental setup that is illustrated early in Fig. 1. Our hardware consists of two COTS servers, equipped with two Intel X520 dual port NICs operating at 10 Gbps. Each server has 2 Intel Xeon E52690 processors, with 12 physical cores per processor, running at 2.60 GHz. Each processor has 3 levels of cache hierarchy, ranging from 576 KB for the L1 to 30 MB for the L3. The RAM consists of two Non-uniform Memory Access (NUMA) nodes for a total size of 128 GB.

We use one server as Device Under Test (DUT) and another server 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, packet loss at the NIC, and packet loss at the DUT.

Given that traffic comes from a single 10 Gbps line-card, the hardware setup would be vastly over-provisioned w.r.t. CPU and memory in case all cores would be used. Therefore, we run the DUT on a single CPU core attached to a single RSS queue, as typically done in stress-test conditions.

B. Software Setup

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) [5]. In a nutshell, VPP implements VNFs as software components (aka *nodes*) that can be linked together in a specific configuration (aka *forwarding graph*). A specific input node (aka *dpdk-input*) polls the line-card for new packets, grabbing a batch (aka *vector*) from the ring for processing. Notice from Tab. II 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: 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.

TABLE II: Experimental configuration parameters.

	Parameter	Value
HW	NIC	Intel X520 dual-port 10 Gbps
	CPU	2× Intel Xeon E52690 @ 2.6 GHz
	Caches L1/L2/L3	32 KB/256 KB/30 MB
DUT	Software router	VPP 17.04
	Number of CPU cores	1
	Number of RSS queues	1
	Memory allocated	4 GB
	Size of input queue (pkts)	$L = 4096$
	Max DPDK batch size (pkts)	32
	Max VPP batch size (pkts)	$\beta = 256$
TX/RX	Traffic Generator	MoonGen
	Rate span [min..inc..max]	[0.5..0.5..10] Gbps
	Hi/Lo rates	10 Gbps / 2.5 Gbps
	Arrival rate process	Constant bit-rate (CBR)
	Data points per rate (pkts)	138k
	Functions	{ XC, Eth, IPv4, IPv6 }
	Scenarios	Homogeneous vs Heterogeneous

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.

2) TX/RX: For traffic generation and reception, we use MoonGen [20], a state-of-the-art scriptable tool capable of sustaining 10 Gbps line-rate. MoonGen also provides APIs to perform basic measurements from 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 [3], [6]. 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-load (10 Gbps) and a low-rate (2.5 Gbps) regimes.

C. Scenarios

We consider two VNF cases, where 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 introducing different types of lookup and data structures to provide instances of compute-batching with different complexity.

1) Homogeneous Cross-Connect Function: In this scenario a single simple VNF, usually referred to as *cross-connection* (XC), is applied to all packets before the immediate forwarding, 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 traffic. 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.

2) Heterogeneous Eth/IPv4/IPv6 Functions: As pointed out in [10], 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 have $\frac{1}{3}$ of the bandwidth, so that each function activates with probability $\frac{1}{3}$, resulting in different function breakdowns across batches. We leave the investigation of even more complex scenarios, e.g., featuring an uneven split, a larger set of functions, or longer chains of functions for future work. In this scenario, both the function and the vector split are heterogeneous, which already makes it a quite challenging use case for our model.

V. 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.

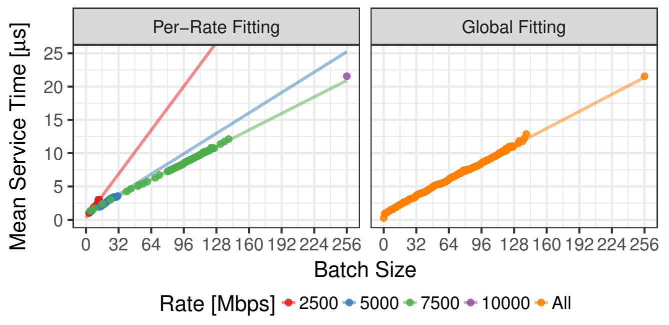

Fig. 4: Size-dependent batch service times. Points indicate mean values from measurements, lines denote linear fits.

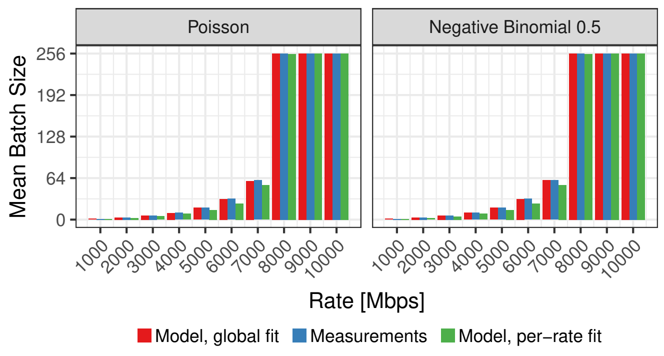

Fig. 5: Mean batch sizes for different rates, arrival processes, and fitting strategies.

A. *Model Tuning Options*

As detailed in Sec. III, 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 $b_i(k)$. Due to our choice of hardware and software components, the values for L and β are fixed 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 $1/\sqrt{\mathrm{E}[A]}$, (ii) the geometric distribution with a coefficient of variation that equals 1, 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 suitable distributions for these batch sizes, linear fitting of the mean per-batch service times is performed. The Poisson distributions for the service time are then generated with measurement-based means where available and with fitted means otherwise. Finally, the mean batch service time $\mathrm{E}[B_i]$ can depend on the applied rate due to internal specifics of the packet processing framework. Hence, the aforementioned fitting can be done either globally or on a per-rate basis. These choices represent trade-offs 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.

B. *Homogeneous Scenario*

1) Per-rate vs Global Fit: We illustrate the immediate effects of the chosen fitting strategy in Fig. 4. 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 significantly change depending on the considered arrival 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. In particular, at low rate 2.5 Gbps the size of the processed batch is small, which is not as efficient to process as a larger batch, resulting in longer mean service times. Starting at 7.5 Gbps, it can be seen that batch size spans a much larger range, whereas for 10 Gbps the batch size is consistently maximal and the system likely operates in a lossy regime.

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. 5 display the mean batch size on their y-axes.

The two subplots correspond to evaluations that use Poisson (left) and negative binomial distributions (right) and 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. In the former case mean values differ by only up to 1 packet, whereas differences of up to 8 packets are observed for the latter. This effect can be explained by the fact that the per-rate fitting strategy can suffer from performance issues when the

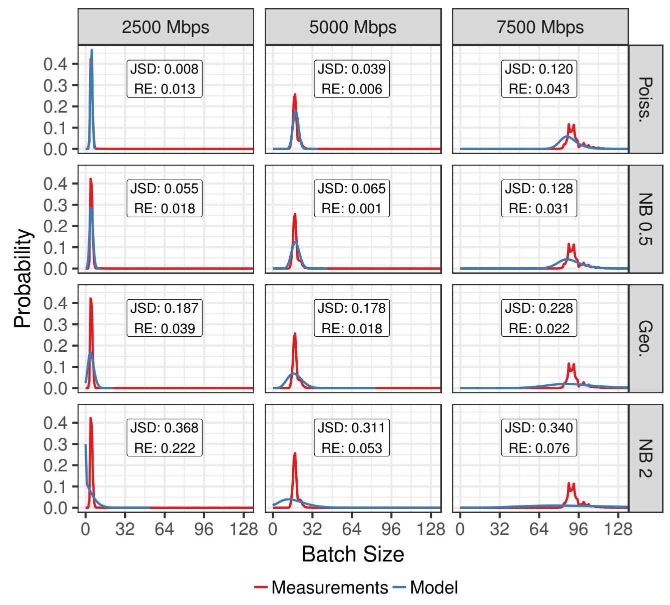

Fig. 6: Batch size distributions for different rates and arrival processes.

model requires service time information for batches that have not been observed in the corresponding measurements. Hence, for the remainder of our evaluation, we show results that are obtained with the global fitting strategy.

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. We quantify the difference between the distribution that is returned by the measurements, $p(k)$, and the model, $q(k)$, by means of the Jensen-Shannon divergence (JSD) which is symmetric and bounded, allows to equally weight differences among $p(k)$ 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. 6 displays the batch size distribution obtained by means of measurements and our model. Given the batch size on the x-axis, the y-axis represents the corresponding probability and annotations provide the JSD and RE values. 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. 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 sub-microsecond interarrival times are to be expected. Therefore, we use interarrival times that follow a Poisson distribution for the remainder of this work.

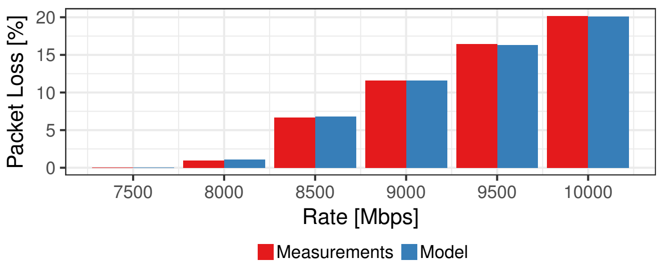

Fig. 7: Packet loss probability for different rates.

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 Fig. 7, the actual packet loss that is reported in the measurements is compared to the model's predictions. Given the rate on the x-axis, the height of the bars denotes the packet loss percentage. Rates below 7.5 Gbps are omitted since they are equal to 0 for both the measurements and the model. 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 key performance indicators 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.

C. Heterogeneous Scenario

We continue our validation with the heterogeneous scenario which features three types of packets that receive different

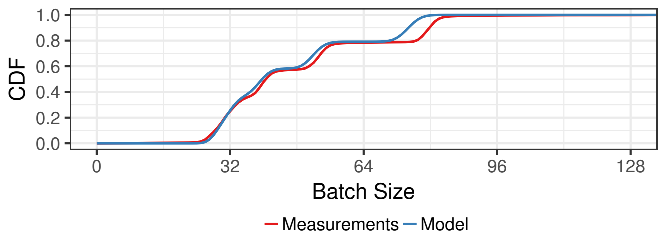

Fig. 8: Distribution function of the batch size for the mixed traffic scenario at 2500 Mbps.

treatment by the network function. Based on the insights from the cross-connect case, we use packet interarrival times that follow a Poisson distribution and employ Poisson distributions for the batch service time. The means of the latter are based on the means obtained in our measurements for observed batch sizes, or on a global linear fit otherwise.

For the high load scenario of 10 Gbps, the model accurately predicts a batch size distribution that has 100 % of its probability at the maximum value of 256. Furthermore, it also reports a packet loss probability of 56.61 % which very closely matches the 56.65 % that are obtained via testbed measurements.

For the low load scenario of 2.5 Gbps, Fig. 8 shows the cumulative distribution function (CDF) of the batch size obtained by means of measurements and the model. Despite applying the unmodified generic model to a significantly more complex scenario, the model achieves a very high accuracy regarding the batch size distribution. However, a small shift near the batch size of 75 indicates a systematic mismatch. Hence, we perform an in-depth analysis of the measurements in order to further investigate the cause of this behavior.

To this end, Fig. 9 portrays a time series view of consecutive batch sizes during an experimental run, reporting just 50 batches for the sake of illustration. A repeating pattern of batch sizes can be observed: the pattern begins with a batch of around 80 packets, that is followed by batches whose size steadily declines until it falls below 32, after which the pattern starts again. The monotonous decrease of batch sizes can be explained by the more efficient processing of large batches, during which fewer packets arrive than are processed. In contrast, during the service time of batches whose size is below 32, the overhead of the framework can be amortized over fewer packets, causing the system to enter a less efficient regime during which significantly more packets arrive.

This alternating behavior suggests the presence of (at least two) different processing regimes. We further analyze this phenomenon by checking the mean service time for different batch sizes in the mixed traffic case. In particular, for different batch sizes on the x-axis, Fig. 10 displays the mean service time on the y-axis. Differently colored dots correspond to values obtained in the low and high load context while the grey line corresponds to the global linear fit. As suggested by the previously observed traffic pattern, a significant change in terms of the mean batch service time occurs for batches

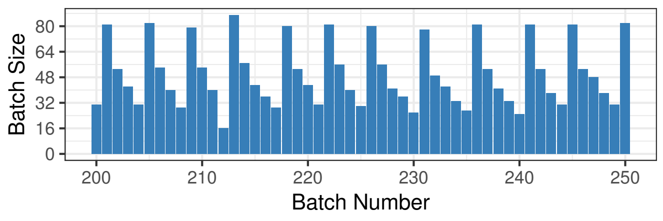

Fig. 9: Batch size development over time for the mixed traffic scenario at 2500 Mbps.

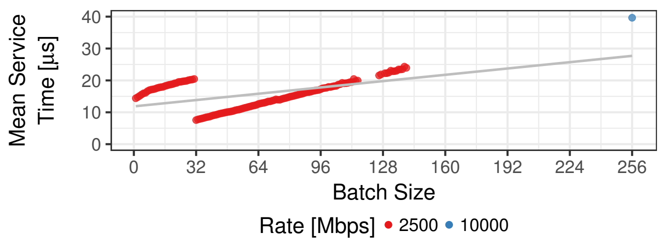

Fig. 10: Size-dependent batch service times for the mixed traffic scenario. The grey line represents a linear fit for all rates.

having size 32. Incidentally, notice that this drop is related to DPDK's internal batch size of 32 (cf. Tab. II), which causes a drastic change of slope for the linear fit (service time drops by a factor of 2 around that DPDK batch value) which is the likely cause of the observed mismatch.

In summary, even when not taking into account these highly implementation-specific details and autocorrelations, our model generalizes very well and provides accurate predictions for both performance indicators, i.e., the batch size distribution and the packet loss probability.

VI. Related Work

Related to our work is either *experimental work* [2]–[10] of NFV systems or *modeling work* [11]–[15], [21]–[26] that either shares a similar technique or NFV focus.

1) Experimental Viewpoint: As introduced earlier, 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 [3] and DPDK [2]), as well as high-level full-blown stacks that apply NFV functions with a compute batching paradigm [5]–[8]. 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.

More recently, work started to appear that extends the comparison to a limited subset of the aforementioned tools [6], [9], [10]. For example, [9] 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 [6]. Finally, [10] experimentally compares NFV throughput with chains of heterogeneous functions using OVS-DPDK, SR-IOV, and FD.io VPP. Given findings in [6], [9], [10], it is reasonable to assume that the model presented in this paper should also be fit to express the performance of other frameworks, which we aim at tackling as future work.

2) Modeling Viewpoint: The theory of *bulk* queueing systems has long been studied [21]. For Markovian bulk input $M^{[X]}/M/1$ and service $M/M^{[X]}/1$ systems, [22] provides closed form solutions under Poisson arrivals and exponentially distributed service times. Particularly, *bulk-input* Batch Markovian Arrival Processes (BMAP) have been well studied [12], [13], and applied to study long lived TCP connections [14], [23] or to model aggregated IP traffic [15]. Similar studies for *bulk-service* systems, which would be more relevant w.r.t. our batch-processing perspective, are missing to date.

Models of Network Functions Virtualization (NFV) have also recently appeared [11], [24]–[26]. In particular, queueing models are used in [24] and [25] to describe software-based networks. Both these models adopt a global network view and strongly abstract the mechanisms of specific network elements, as opposed to this work. Under this perspective, studies closer to ours are [11], [26], which both aim at predicting virtual function performance on multi-core systems. Yet, [26] does not take into account mechanisms like batch arrival or batch processing of packets, which both are crucial characteristics of nowadays NFV routers. In contrast, the authors of [11] assume fixed processing times, which we show not to hold true in practice, and omit a proper experimental validation.

VII. 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. 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 our future work, we plan to validate the generality of the model beyond the experimental results gathered in this paper by considering a larger set of NFV routers such as FastClick and G-opt as well as more realistic traffic patterns, e.g., with chains of functions of different lengths.

References

[1] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The Click Modular Router," *Operating Systems Review*, 1999.

[2] Intel, "Data Plane Development Kit," http://dpdk.org.

[3] L. Rizzo, "netmap: a novel framework for fast packet I/O," in *USENIX ATC*, 2012.

[4] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *USENIX NSDI*, 2015.

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

[6] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *ANCS*, 2015.

[7] J. Kim, S. Huh, K. Jang, K. Park, and S. Moon, "The power of batching in the click modular router," in *Asia-Pacific Workshop on Systems*, 2012.

[8] A. Kalia, D. Zhou, M. Kaminsky, and D. G. Andersen, "Raising the bar for using gpus in software packet processing." in *NSDI*, 2015.

[9] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet io," in *ANCS*, 2015.

[10] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris, "Characterizing the performance of concurrent virtualized network functions with OVS-DPDK, FD.IO VPP and SR-IOV," in *ACM/SPEC International Conference on Performance Engineering*, 2018.

[11] S. Gebert, T. Zinner, S. Lange, C. Schwartz, and P. Tran-Gia, "Performance modeling of softwarized network functions using discrete-time analysis," in *28th International Teletraffic Congress (ITC)*, 2016.

[12] D. Manfield and P. Tran-Gia, "Analysis of a finite storage system with batch input arising out of message packetization," *IEEE Transactions on Communications*, 1982.

[13] D. M. Lucantoni, "New results on the single server queue with a batch Markovian arrival process," *Communications in Statistics. Stochastic Models*, 1991.

[14] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," *ACM SIGCOMM Computer Communication Review*, 2000.

[15] A. Klemm, C. Lindemann, and M. Lohmann, "Modeling IP traffic using the batch Markovian arrival process," *Performance Evaluation*, 2003.

[16] T. Herbert and W. de Bruijn, "Scaling in the linux networking stack," https://www.kernel.org/doc/Documentation/networking/scaling.txt, 2011.

[17] L. Rizzo, "Device polling support for freebsd," in *BSDConEurope Conference*, 2001.

[18] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in *SIGCOMM*, 2010.

[19] S. Bradner and J. McQuaid, "RFC2544 Benchmarking Methodology for Network Interconnect Devicesgoo," https://www.ietf.org/rfc/rfc2544.txt, 1999.

[20] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *IMC*, 2015.

[21] I. W. Kabak, "Blocking and delays in m (x)/m/c bulk arrival queueing systems," *Management Science*, 1970.

[22] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2018.

[23] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *ACM SIGCOMM Computer Communication Review*, 1998.

[24] G. Faraci, A. Lombardo, and G. Schembra, "A building block to model an SDN/NFV network," in *IEEE International Conference on Communications (ICC)*, 2017.

[25] A. Lombardo, A. Manzalini, V. Riccobene, and G. Schembra, "An analytical tool for performance evaluation of software defined networking services," in *IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[26] K. Suksomboon, M. Fukushima, S. Okamoto, and M. Hayashi, "A dilated-CPU-consumption-based performance prediction for multi-core software routers," in *IEEE NetSoft Conference and Workshops (NetSoft)*, 2016.