Performance Modeling of Softwarized Network Functions Using Discrete-Time Analysis

Steffen Gebert, Thomas Zinner, Stanislav Lange, Christian Schwartz, Phuoc Tran-Gia
University of Würzburg, Institute of Computer Science
{steffen.gebert,zinner,stanislav.lange,christian.schwartz,trangia}@informatik.uni-wuerzburg.de

***Abstract*—The softwarization of networks promises cost savings and better scalability of network functions by moving functionality from specialized devices into commercial off-the-shelf hardware. Generalized computing hardware offers many degrees for adjustment and tuning, which can affect performance and resource utilization. One of these adjustments are the interrupt moderation techniques implemented by modern network interface cards and operating systems. Using these, an administrator can optimize either lower latencies or lower CPU overhead for processing of network traffic. In this work, an analytical model that allows computing relevant performance metrics like the packet processing time and the packet loss for generic virtualized network functions running on commodity hardware is developed. The applicability of the model is shown by comparing its outcome with measurements conducted in a local testbed featuring a VNF that acts as an LTE Serving Gateway (SGW). Based on this model, impact factors like the average packet interarrival time, the interarrival time distribution, and the duration of the interrupt aggregation interval are studied.**

***Keywords*—Discrete-Time Analysis, Performance Modeling, NFV, VNF, Queueing Theory.**

I. Introduction

The trend towards softwarization of networks, especially using Software Defined Networking (SDN) and Network Functions Virtualization (NFV), promises more flexibility and innovation for networks. Network functions running on commercial off-the-shelf (COTS) hardware have many appealing advantages such as easy scale-up or scale-down of computing resources as well as scale-out or scale-in of virtual machines among the available physical hardware. Further, faster release cycles compared to hardware devices are promised.

This high flexibility, however, comes at the expense of performance [4], [13], i.e., a lower packet throughput and longer processing delays of softwarized solutions compared to hardware-based implementations. The usage of particular network functions, for instance within network function chains, however, has stringent performance requirements. Firstly, enough function instances have to be available to handle the corresponding traffic. Secondly, the overall processing delay of a network function should be minimized, particularly in case of large forwarding graphs where such delays sum up.

Forwarding packets from the Network Interface Card (NIC) via the kernel space to a specific network function requires an interrupt to inform the Central Processing Unit (CPU) about the packet arrival. Such an interrupt is costly and prevents normal CPU execution for a couple of µs. To cope with high packet rates of today's links with rates of almost 2 million packets per second (Gigabit Ethernet) to around 20 million packets per second (10 GE), optimization techniques are required to prevent CPU livelocks. Such techniques, e.g., *interrupt moderation* or *interrupt coalescence*, are provided by operating systems and hardware components, i.e., NICs. In order to tune COTS hardware for their particular use case, these techniques allow to adjust the trade-off between the number of interrupts and therewith the overall throughput rate, and the corresponding processing delays. For that, incoming packets are aggregated over a certain time span, resulting in a single transfer of all accumulated packets in a batch from the NIC to the kernel space and then to the corresponding network function. This batch-style packet processing is also used by advanced packet processing mechanisms. Cisco's Vector Packet Processing (VPP) [2] uses Intel DPDK [5] to first apply busy polling of incoming packets from the NIC. Afterwards, it processes the packet headers of the aggregated packets as vectors, i.e., it processes equal headers on a protocol basis (Ethernet, IPv4, IPv6, ARP, etc.) in parallel rather than processing the complete stack step-by-step for each packet. To understand the impact of performance-relevant parameters on these metrics and in order to allow an adequate dimensioning and a proper performance prediction, appropriate performance models are required.

The contribution of this paper is a discrete-time model for *Virtualized Network Functions (VNFs)* running in software on commodity hardware. The presented model takes into account interrupt moderation, a technique used by current operating systems and server hardware to reduce the overall number of interrupts. Based on an exemplary network function, a mobile network Serving Gateway (SGW), we determine an empirical service time distribution. We illustrate the applicability of the model by comparing it to measurements for a fixed aggregation interval and varying interarrival times. After that, the impact of different interarrival times, interarrival distributions, and aggregation interval durations on the processing times and the packet loss are presented. The proposed model also allows computing distributions, i.e., mean values, standard deviations, as well as quantiles of the delay distributions.

The remainder of this work is structured as follows: Background information as well as related work is introduced in Section II. The steps involved in processing packets in a x86 system are described in Section III, before an abstract model is introduced in Section IV. After its applicability based on measurements is shown in Section V, exemplary evaluations of the packet processing time and packet loss behavior under different settings are presented in Section VI. Finally, Section VII draws conclusions and outlines future work.

978-0-9883045-1-2/16 $31.00 © 2016 ITC

II. Background & Related Work

This section discusses related work with respect to the performance of softwarized network functions and corresponding optimization mechanisms. Afterward, interrupt moderation techniques are discussed.

A. Performance of Packet Processing in Software

Applications processing network traffic send and receive data packets through functions provided by the operating system kernel. Accordingly, packets traverse a complex chain of forwarding steps between the NIC, the kernel, and the software application resulting in a specific delay overhead.

One major contributor to these delays are copy operations between the memory of the kernel space and the user space. To reduce this overhead, multiple techniques and frameworks that enable a faster processing of packets in software have been introduced. These approaches, e.g., Netmap [12], ClickOS [10], Intel DPDK [5], or VPP [2] bypass the kernel completely during packet reception, use shared memory buffers to avoid additional copy operations, process packets in batches, or replace the entire network stack. Accordingly, these mechanisms usually speed up specific parts of the stack. An extensive measurement study on the performance of several of the aforementioned mechanisms in case of packet forwarding is conducted in [1].

However, the abovementioned studies have several drawbacks. First, the focus on simple network functions like pure packet forwarding obscures the influence of the processing time spent in the user space on the total processing time. This component, however, might account for the majority of the total processing time. Second, measurements are conducted for very specific use cases and cannot be generalized in order to obtain a holistic evaluation of the proposed mechanisms. Finally, it is impossible to determine the feasibility of an approach without identifying its key performance indicators. Therefore, a model for analyzing the packet processing performance on COTS hardware is required. In addition to providing the capability to derive key performance indicators, model parameters can be tuned in order to represent different acceleration techniques and quantify their effects in the context of different use cases.

Based on such evaluations, it could be decided, which technique offers a good trade-off between complexity of implementation and speedup for a specific network function. As seen in [6], operating modes of network functions exist, in which the overhead of packet handling, and therefore the speedup gained by techniques like DPDK, is negligible.

The model developed in this work is a first step towards a model of packet processing in commodity hardware running a general purpose operating system.

B. Interrupt Moderation

In particular, the previously listed frameworks also help to avoid livelocks [8] that result from the CPU being effectively busy with interrupt handling instead of executing the program that processes incoming data. In order to avoid such livelocks and to reduce the overhead of packet processing in a server, several approaches that apply interrupt moderation have been introduced on operating system side as well as in networking hardware.

The networking stack (*New API*, short NAPI [8]) in the Linux kernel disables interrupt handling for interrupts related to receiving packets, once the first packet is processed. Followed by that, the NIC queue is polled in assumption that multiple packets arrived in a burst. After a certain number of packets have been processed, or a timeout occurs, interrupts are re-enabled and the process restarts once the next packet arrives.

Hardware-based implementations are offered in many server network adapters. The actual feature set varies between different chipsets. For receive as well as transmit directions, the NIC can hold back interrupts until either a pre-configured number of packets is received or sent, or until a pre-configured time since the first packet starting the batch passed by. Further options allow to define a threshold to differentiate between a low and a high traffic load and to specify options for both of these conditions. Finally, some NICs offer adaptive modes, in which they change their behavior based on the current receive rate.

The effects of interrupt moderation and the reduction of end-to-end delay has been subject of several studies already. The influence on passive and active network measurements is investigated in [11]. By identifying packet bursts, effects of interrupt moderation can be considered when running capacity and delay measurements using commodity NICs. A similar methodology is applied in our work in order to estimate the processing time within the application.

By increasing the interrupt rate, more context switches occur in the CPU, when switching between interrupt handling and data processing. Every context switch comes at a certain cost, especially when code and data are evicted from the CPU caches. [14] estimates a time of 3-4.5 µs for a pure context switch without any computation on a multi-core system and 1.3-1.9 µs when the processes are pinned to one CPU core. When the CPU's cache lines are not filled, experimental results show context switch delays of 2.2-2.9 µs, when the process is pinned to a specific core and a simple program that writes memory pages is used. With virtualization, the time for context switches is reported to be increased 2.5-3-fold. As a rule of thumb, the author estimates 30 µs for a context switch in real-world scenarios. In contrast, the delays seen in the following are mostly based on one single program being executed, resulting in much lower overhead, as the contents of CPU caches are usually not evicted by code or data of other applications.

Latencies of network communication between two servers are studied in detail in [7]. The authors investigate the contributing factors to latencies in Ethernet-based TCP/IP connections and try to achieve a minimal end-to-end latency. Using a modified Linux kernel, the authors make use of nanosecond-precision timers offered by CPUs to break down the packet transmit and receive latencies for a 1 Gbps and a 10 Gbps Ethernet NIC. Based on measurements and estimations, this study indicates a total receive latency of 7.747 µs for a 1 Gbps card. The main contributors (more than 1 µs) are *Interrupt cause register read requirement*, *SoftIRQ*, *Wakeup application to process socket information*, as well as the example application identifying and acknowledging the received data (*ACK the pong received by the remote sender*).

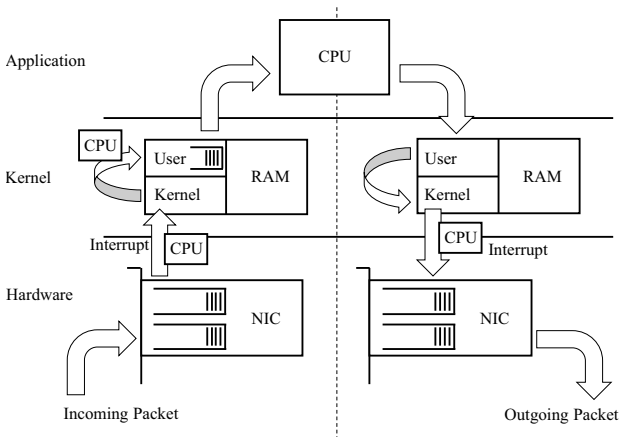


Fig. 1. Packet processing in a server.

III. System Description

In order to understand the process of packet processing within a Linux x86 system, an abstracted description is provided in the following. This process, which starts with receiving a packet on the wire and ends with the processed packet being sent over the wire, is also depicted in Figure 1.

Read from media: The network interface card reads data from the transmission media by interpreting electrical or optical signals within the MAC layer and transforms it into packets.

Store in receive queue: These packets are saved into a receive queue implemented in hardware inside the NIC. Multiple such queues can exist and, based on hashing, packets can be distributed among these queues.

Trigger interrupt: In the most simple case, the NIC triggers an interrupt signal to notify the CPU about the arrival after every received packet. Interrupt moderation techniques, which are under study in this work, aim at reducing the number of interrupts by processing multiple packets at once. Depending on the capabilities and configuration of the network card, this batch processing mechanism can be triggered by a timeout, by accumulating a specified amount of received packets, or a combination of both. Some NICs also offer adaptive modes, which adjust timers and batch sizes according to the current packet rate.

Read packet from NIC: As soon as the interrupt is sent, the CPU stops other work in order to load and execute the interrupt service routine of the NIC driver. This code then fetches the batch of packets from the network card. This process, which results in a *context switch* of the CPU, is rather costly as CPU registers first need to load new code and data. Additionally, this also purges other applications' code/data and thus introduces overhead. This overhead caused by an interrupt can also lead to livelocks, if all CPU time is spent with interrupt handling. It can be reduced by avoiding interrupts for every single packet at the cost of additional delay.

Store packet in buffer: The packet data is stored in a buffer in RAM, until an application requests them for processing. The size of this buffer is limited to a fixed number of bytes[1]. If the application cannot catch up with reading packets, the kernel drops packets. The process of copying packet data from kernel space to user space takes additional time per packet.

Process packet in application: While the application processes the packet, it blocks the CPU.

Send packet: After processing, the packet traverses the same way backwards, until it is finally sent to media. The NIC informs the operating system about this by means of another interrupt.

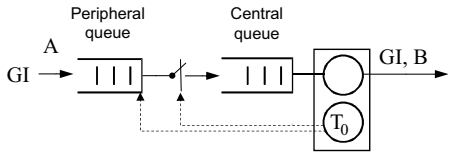


Fig. 2. Queueing Model.

IV. Model

A. Abstract Server Model and Performance Metrics

The queuing model used for the performance analysis of the system outlined in Section III is depicted in Figure 2. It is a generalization of the clocked approach introduced by Manfield et al. [9]. The generation of packets follows an arbitrary distribution A. The packets are stored in a peripheral queue which is assumed to have infinite size. Incoming packets are transferred in a batch to the central queue after a time interval τ initiated by the first packet after a batch transfer. The inner queue is then modeled as a $GI^{[X]}/GI/1-L$ system and evaluated by means of discrete-time analysis. Distributions of the batch sizes and burst interarrival times are derived in the next subsection.

B. Model of the Peripheral Queue (NIC)

In the peripheral queue, which represents the network interface card, packets are aggregated. The resulting batch is then forwarded to the central queue, which represents the CPU/software.

For the remainder of this work, we use the following notation to distinguish between random variables (RVs), their distributions, and their distribution functions. A random variable is represented by an uppercase letter, e.g., X. The distribution of X is denoted by $x(k)$ and is defined as

$$x(k) = P(X = k),\ -\infty < k < \infty.$$

Furthermore, the distribution function of X is written as $X(k)$ and is defined as

$$X(k) = \sum_{i=0}^{k} x(i),\ -\infty < k < \infty.$$

Finally, $\mathrm{E}[X]$ denotes the mean of X and $*$ refers to the discrete convolution operation, i.e.,

$$a_3(k) = a_1(k) * a_2(k) = \sum_{j=-\infty}^{\infty} a_1(k-j) \cdot a_2(j).$$

[1] in Linux, `net.core.rmem_max = 131071 bytes`

The following distributions are used for modeling the peripheral queue:

- $a(k)$: distribution of the packet interarrival time.
- $r_a(k)$: distribution of the packet recurrence time.
- $\tau(k)$: distribution of the duration of the aggregation interval.
- $u_n(k)$: distribution of unfinished work in the system before the arrival of the n-th batch.
- $o(k)$: distribution of the interrupt processing delay.
- $s(k)$: distribution of the interarrival time between batches.
- $x(k)$: distribution of the batch size.
- $f^{(j)}(k)$: distribution of the time between the start of an aggregation interval and the arrival of the j-th packet. Since the aggregation interval starts with the arrival of a packet, this time equals the sum of j interarrival times. The corresponding random variable is referred to as $F^{(j)}$.
- $w_i(k)$: distribution of the waiting time of the i-th packet in the peripheral queue.

The first packet arriving after a burst transferal initiates a new aggregation interval. All packets arriving in this time frame are transferred to the inner queue at the end of this interval. Based on the work in [15] and [3], the batch size distribution $x(k)$ can be computed as follows.

$$x(k) = \tau(0)\delta(k) + \sum_{m=1}^{\infty} \tau(m) \sum_{i=0}^{m-1} \left(f^{(k)}(i) - f^{(k+1)}(i) \right), k = 0, 1, ... \tag{1}$$

The equation allows calculating the number of arrival events in an arbitrarily distributed time interval. The special case, in which no arrivals are observed in an interval of length 0, is covered by the first term. The function δ is defined in Equation 2. For the remaining interval lengths, the law of total probability is used in the second term in order to calculate the conditional probability $x(k|m)$. It can be derived from the relationship shown in Equation 3.

$$\delta(k) = \begin{cases} 1 & k = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\begin{aligned} x(k|m) &= \mathrm{P}\left(F^{(k)} < m \leq F^{(k+1)} \right) \\ &= \mathrm{P}\left(F^{(k)} < m \right) - \mathrm{P}\left(F^{(k+1)} < m \right) \\ &= \sum_{i=0}^{m-1} \left(f^{(k)}(i) - f^{(k+1)}(i) \right), m > 0 \end{aligned} \tag{3}$$

Since the first packet after a transfer initiates the next aggregation interval, the batch interarrival time s can be calculated as the sum of the recurrence time of one packet, i.e., r_a, and the duration of the aggregation interval τ:

$$s(k) = r_a(k) * \tau(k) \tag{4}$$

Since the first packet in a batch triggers the timeout, the waiting time of consecutive packets is reduced. In particular, the waiting time of the i-th packet in the peripheral depends on the arrivals of the $i-1$ packets before it. Hence, the distribution of its waiting time can be computed as follows:

$$w_i(k) = \pi_0 \left[\tau(k) * \underbrace{a(-k) * \cdots * a(-k)}_{(i-1) \text{ times}} \right] \tag{5}$$

C. Model of the Central Queue (CPU/software)

We model the inner queue as a $GI^{[X]}/GI/1-L$ queue, i.e., a system with batch arrivals and bounded delay. The waiting time of packets is limited to a maximum value of L, i.e., customers who arrive and would have to wait longer than $L-1$ are rejected. Our analysis extends the work presented in [16] by introducing batch arrivals. A similar notation, as presented in the following, is used:

- $u_{n,b_i}(k)$: distribution of unfinished work in the system before the arrival of the i-th packet of the n-th batch.
- $B_{n,i}$: RV for the service time of the i-th packet of the n-th batch.
- p_b: average blocking probability per packet.
- $\pi_0(\cdot)$: sweep operator which sums the probability mass of negative unfinished work in the system and appends it to the state for an empty system.

$$\pi_0(x(k)) = \begin{cases} x(k) & k > 0 \\ \sum_{i=-\infty}^{0} x(i) & k = 0 \\ 0 & k < 0 \end{cases}$$

- $\sigma^m(\cdot)$: operator which truncates the upper part of a probability distribution function.

$$\sigma^m(x(k)) = \begin{cases} x(k) & k \leq m \\ 0 & k > m \end{cases}$$

- $\sigma_m(\cdot)$: operator which truncates the lower part of a probability distribution function.

$$\sigma_m(x(k)) = \begin{cases} 0 & k < m \\ x(k) & k \geq m \end{cases}$$

The development of the batch arrival process is illustrated in Figure 3. Observing the packets of the n-th batch arrival, the i-th packet of the burst is accepted if the current unfinished work in the system is less than $L-1$. In case the packet is accepted, the unfinished work is increased by the amount of work $B_{n,i}$ that is required to process the packet. Otherwise, the packet as well as the remaining packets of the current batch are rejected.

The following recursive relationship can be used in order to compute the amount of unfinished work in the system:

$$u_{n,b_1}(k) = u_n(k) \tag{6}$$

$$u_{n,b_{i+1}}(k) = \sigma^{L-1}\left[u_{n,b_i}(k)\right] * b_{n,i}(k) + \sigma_L\left[u_{n,b_i}(k)\right] \tag{7}$$

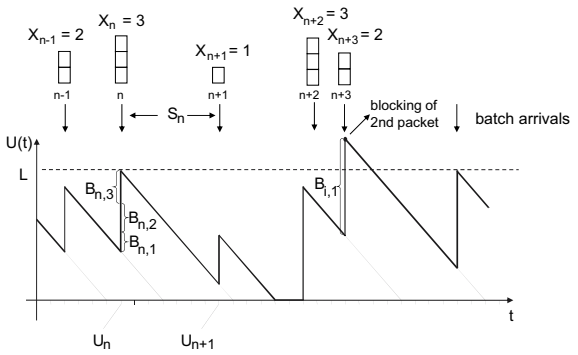


Fig. 3. Exemplary system development for $GI^{[X]}/GI/1-L$ with bounded delay.

Hence, the remaining unfinished work in the system at the arrival of the next batch can be computed as:

$$u_{n+1}(k) = \pi_0 \left[\left(\sum_{i=1}^{\infty} x(i) \cdot u_{n,b_i}(k) \right) * s_n(-k) \right] \tag{8}$$

Using these equations, an algorithm for calculating the workload prior to the i-th arrival can be derived. The algorithm can be used for both stationary and non-stationary traffic conditions. Under stationary conditions, the index n and $(n+1)$ in these equations can be suppressed, cf. Equation 9. Furthermore, we assume that the packet service time is independent of a packet's position within the batch. Hence, the RV B_n refers to the service time for packets in the n-th batch. Similarly to Equation 9, the index n can also be suppressed under stationary conditions, resulting in RV B.

$$\begin{aligned} u(k) &= \lim_{n\to\infty} u_n(k) \\ u_{b_i}(k) &= \lim_{n\to\infty} u_{n,b_i}(k) \end{aligned} \tag{9}$$

The computational diagram of the system is depicted in Figure 4. Depending on the batch size X, the unfinished work after a batch arrival can be determined by following the corresponding path through the diagram. Each of the X phases in such a path represents the relationship from Equation 7. Finally, the batch interarrival time s_n is taken into account and the π_0 sweep operator is used in order to ensure that a proper probability distribution is returned.

It is also possible to quantify the load ρ of the central queue. This is achieved by calculating the ratio between the amount of work that arrives within a given time interval and the amount of work that is processed in this interval. In particular, we observe that the amount of work that arrives within a batch interarrival time depends on the batch size and the packet service time (cf. Equation 10). Note that both the batch size and the batch interarrival time are affected by the packet interarrival time (cf. Equations 1 and 4).

$$\rho = \frac{\mathrm{E}[X]\,\mathrm{E}[B]}{\mathrm{E}[S]} \tag{10}$$

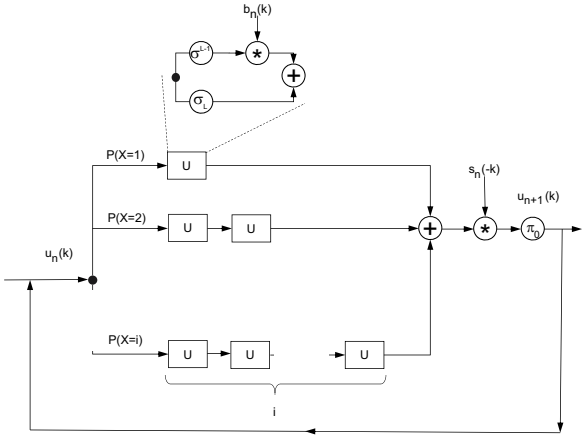


Fig. 4. Computational diagram for $GI^{[X]}/GI/1-L$ with bounded delay.

Finally, the packet loss probability in statistical equilibrium can be computed as follows:

$$p_b = \sum_{i=1}^{\infty} \left(\frac{1}{i} x(i) \cdot \sum_{j=L}^{\infty} u_{b_i}(j) \right) \tag{11}$$

Depending on the batch size and the amount of unfinished work added by each packet within the batch, the blocking probability for the latter packets within the batch increases.

D. Combined Model

Using the two models described in Section IV-B and Section IV-C, it is possible to determine the distribution of the total processing time. It is comprised of the waiting time in the peripheral queue, the waiting time in the central queue, and the service time in the latter. The waiting time in the central queue can be calculated from the unfinished work in the system and a packet's position in its batch. Hence, the following equation can be used to calculate the distribution of the total processing time of the i-th packet in a batch, d_i:

$$d_i(k) = w_i(k) * u(k) * \underbrace{b(k) * \cdots * b(k)}_{i \text{ times}} \tag{12}$$

Consequently, the distribution of the total processing time for all packets can be determined via conditional probabilities:

$$d(k) = \sum_{i=1}^{\infty} \mathrm{P}(X=i) \cdot d_i(k) = \sum_{i=1}^{\infty} x(i) \cdot d_i(k) \tag{13}$$

V. Applicability of the Proposed Model

In order to assess the goodness of fit of the introduced model, measurements are conducted in a test bed and compared with the model's predictions. In the following, the components of this test bed are described alongside the methodology for accurately measuring the CPU processing times as well as the results of the comparison.

A. Testbed Setup

The testbed setup is depicted in Figure 5. The SGW [6] application runs on the Device Under Test (DUT), a server[2] running a recent Linux version[3] equipped with a four-port NIC. Similar to [6], GPRS Tunneling Protocol (GTP) traffic is generated using a hardware traffic generator[4]. In order to evaluate per-packet processing times, wiretaps that duplicate all traffic are placed between the traffic generator and the receiving NIC of the DUT, as well as between the emitting NIC of the server and the traffic sink, which is again the traffic generator. The wiretaps are connected to a hardware capture card[5], which provides nanosecond precision timestamping of received traffic.

The processing time of the server is measured by calculating the time between a packet's arrival at the first wiretap and its arrival at the second wiretap. The packets at the two wiretaps are matched based on a unique 40 byte signature that the traffic generator adds to every packet. In order to verify that the traffic generator emits packets at equidistant times and at the correct rate, the interarrival times seen at the first capture card are inspected.

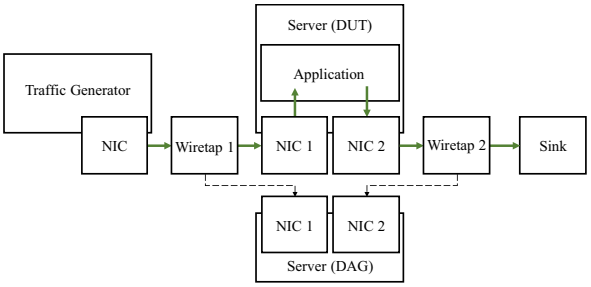


Fig. 5. Testbed setup consisting of the DUT running the SGW application, a traffic generator, and a server with a DAG capture card.

The delay, how long the NIC buffers incoming packets, is adjusted using the `ethtool` command. In this context, `<N>` represents the number of the NIC and `<T>` reflects τ, i.e., the number of microseconds to wait after the first incoming packet:

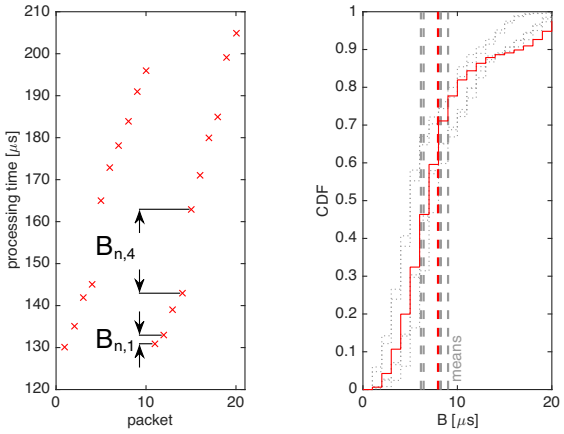
```
# ethtool -C eth<N> rx-usecs <T>
```

B. Estimating CPU Processing Time

In order to determine the processing time of the application code at a per-packet granularity, measurements using `tcpdump` are conducted. The time, when `tcpdump` captures a packet is on the kernel level, right after the interrupt is handled in incoming direction (from NIC 1), i.e., before the packet is copied by the kernel driver code to NIC 2 (cf. Figure 1).

One such exemplary measurement displaying the time between a packet's arrival at the receiving and the sending side is shown in Figure 6(a). The two batches of packets each show an increasing processing time, as the first packet is processed first by the CPU and the last (10th) packet is processed after all others in this batch. Therefore, the difference in the processing

[2]Intel Xeon E5-2620 v2 CPU at 2.10 GHz, Intel I350 NICs, 32 GB of RAM
[3]64 bit version of Debian 7.7 (wheezy, kernel version 3.2.0-4-amd64)
[4]Spirent TestCenter C1
[5]Endace DAG 7.5G2 Gig Ethernet



(a) Function service time of the CPU for packets captured at kernel level.

(b) Distributions of measured function service times and mean values for different runs.

Fig. 6. Differences in processing times of single packets seen in the networking stack (kernel level) allow deriving the CPU service time per packet.

delay between consecutive packets equals $B_{n,i}$, the waiting and processing time in the application.

Given the application used in our experiments, a prototypical VNF implementation of a mobile network Service Gateway, the measurements result in a distribution of processing times with a mean of 8.336 µs. This empirical distribution is used in the following after capping it at the 90% quantile (16 µs) to remove outliers, resulting in a mean of 7.25 µs. This distribution is shown as the red curve in Figure 6(b) and was picked as a representative from multiple measurements. The gray CDFs, as well as the corresponding means (dashed lines) show the CPU processing times of other measurements and highlight the variations between the different runs.

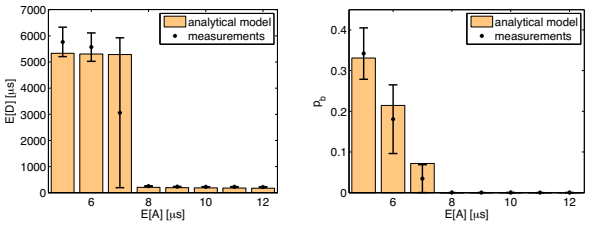
C. Comparison of Model Predictions and Measurements

In order to demonstrate the applicability of the proposed model, we compare its results with measurements. For that, we conduct five independent measurement runs for constant interarrival times between 5 and 12 µs. Each measurement run lasts one minute, and the aggregation interval is set to $\tau = 200\,\mu s$.

The size of the central queue, denoted by L, corresponds to 5,200 µs of unfinished work. Based on the measured mean service time at the CPU $\mathrm{E}[B] = 8.336\,\mu s$, the inner queue size of $L_{byte} = 131,071$ byte as defined by the operating system, and the packet payload of 210 byte, L computes as follows:

$$L = \mathrm{E}[B] \cdot \frac{L_{byte}}{210} = 5200\,\mu s$$

Based on the measurements, we compute the mean processing times and the corresponding confidence intervals on a 95% confidence level, as well as the packet loss probability. Additionally, we compute the mean processing times and the packet loss probability using the analytical model. As service time distribution, we take the empirically measured service



(a) Average processing times. (b) Packet loss.

Fig. 7. Comparison of analytical model and measurements for $\tau = 200\,\mu s$.

time from Figure 6. As interrupt overhead, we use $o = 4\,\mu s$, which is based on the values reported in [7].

Figure 7(a) shows a comparison of the measurements and values obtained from the analytical model. Error bars denote the 95% confidence intervals from five measurement runs. The bars indicate the mean processing time per packet according to the model.

For packet interarrival times below 8 µs, the error bars overlap with the mean values from the model, indicating the applicability of the model. For larger interarrival times, only a slight difference is observed between the model's prediction and the measurements. A possible explanation for this phenomenon is the high degree of variance regarding the empirical function service times shown in Figure 6(b).

In an analogous fashion, Figure 7(b) shows the applicability of the model w.r.t. to the packet loss rate. Except in the case of $\mathrm{E}[A] = 7\,\mu s$, the error bars overlap with the values from the model. Based on the huge error bar seen in the previous figure, this interarrival time roughly corresponds to the maximum rate that the server can handle and the first occurrence of packet loss can be observed. For larger interarrival times, the model and measurements both indicate zero packet loss.

The occurrence of packet loss for an average interarrival time below 8 µs is consistent with the definition of the system load ρ in Equation 10 and the mean service time at the CPU of 7.25 µs that is obtained after removing outliers. Since the average batch size $\mathrm{E}[X]$ can be determined by means of τ and $\mathrm{E}[A]$, and the recurrence time in the context of very low interarrival times is negligible, the system load can be approximated as follows:

$$\rho = \frac{\mathrm{E}[X] \cdot 7.25}{\mathrm{E}[S]} = \frac{\tau \cdot 7.25}{\mathrm{E}[A](\mathrm{E}[R_a] + \tau)} \overset{\mathrm{E}[R_a] \ll \tau}{\approx} \frac{7.25}{\mathrm{E}[A]} \quad (14)$$

Hence, in the context of mean interarrival times below 7.25 µs, the system load is larger than 1, and packet loss occurs. Furthermore, the actual load is slightly higher due to the fact that the same CPU core also handles the interrupts that are caused by outgoing packets. These amount to roughly 20,000 IRQs per second in our scenarios.

VI. Evaluation

In this section, we investigate the behavior of the packet processing server based on the introduced model. In this context, we focus on the total processing time D and the packet loss probability p_b. The influence of the mean packet interarrival time $\mathrm{E}[A]$ and the length of the aggregation interval τ are studied. At first, coarse-grained analyses of the resulting mean processing times and packet loss ratios for different interarrival time distributions and aggregation interval lengths are presented. Afterwards, we investigate the impact of these two influence factors for a particular packet interarrival time distribution on the distribution of processing times.

A. Impact of the Arrival Process

The sensitivity of the modeled system to different distributions of the packet interarrival time A is studied based on four different distributions, namely deterministic (det), Poisson (pois), geometric (geo), and negative binomial (nbin). While for det, pois, and geo, the distributions are characterized solely by $\mathrm{E}[A]$, the parameters p and r of nbin are adjusted so that $\sigma = \mu$ holds true. This ensures a constant coefficient of variation equal to 1.

1) Impact on Mean Processing Times: Figure 8 presents the mean packet processing time D that results from different combinations of the distribution of packet interarrival time and its mean. While the x-axis displays the mean packet interarrival time, the y-axis indicates the average packet processing time. Additionally, line colors represent different values of the aggregation interval length τ and line styles correspond to the four distribution types.

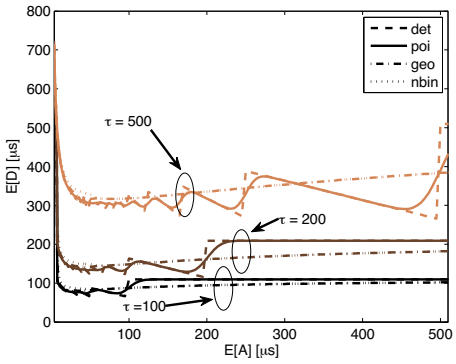
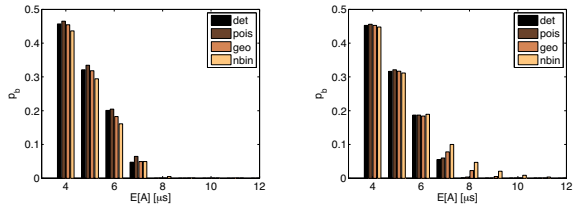


Fig. 8. Effects of different values of $\mathrm{E}[A]$ and different aggregation intervals τ on the mean processing time $\mathrm{E}[D]$.

In most cases, the curve shape is composed of three phases. First, small packet interarrival times result in high processing times that stem from long waiting times in the central queue. As soon as the average interarrival time exceeds τ, in most cases, each batch is comprised of only one packet. As this packet initiated a new aggregation interval, it has to wait until the timer ends after τ. Because of the low rate, the unfinished work at the central queue (the CPU) is low or oftentimes zero, resulting in immediate processing of the packet. Since in this case the processing time in the central queue is relatively low compared to the waiting time in the peripheral queue, the total processing time is mostly influenced by τ. For interarrival times that follow a deterministic or a Poisson distribution, most aggregation intervals contain exactly one packet, resulting in processing times that are slightly higher than τ. In contrast, the negative binomial and geometric distributions lead to bursts



(a) $\tau = 200\,\mu s$. (b) $\tau = 500\,\mu s$.

Fig. 9. Packet loss depending on $\mathrm{E}[A]$ for different τ.

of packet arrivals that result in lower mean processing times. After the first packet of a batch starts the aggregation interval, consecutive packets still arrive within the interval and thus, have a lower waiting time in the peripheral queue.

For interarrival times that are lower than τ, but do not lead to queuing at the central queue, expected batch sizes for all distributions are larger than one. Therefore, the mean waiting time in the peripheral queue decreases and thus, the mean overall processing time $\mathrm{E}[D]$ also decreases.

Although the figure might suggest that decreasing τ, i.e., reducing the interrupt moderation, leads to lower processing times, this is only true until reaching a break-even point. Then, the overhead per packet caused by interrupt handling and context switches accounts for the majority of CPU time.

2) Impact on Packet Loss: As described previously, the processing time increases with the number of packets per second, because packets experience a waiting time at the central queue. As this queue is limited by L (cf. Section III), packet loss occurs once this limit is exceeded as described in Equation 11. In the following, the impact of the mean and distribution of interarrival times on the packet loss probability is evaluated.

Figure 9 depicts the packet loss probability for the four different distributions depending on different mean processing times and lengths of the aggregation interval. It can be observed that the Poisson distributed interarrival times result in the highest packet loss ratio when the system operates at a high load. The assumption behind applying interrupt moderation techniques is a certain burstiness of traffic. Hence, the packet loss ratio is up to 8% lower for nbin than for geometrically distributed arrivals in the case of $\tau = 200\,\mu s$ depicted in Figure 9(a). Due to the higher degree of burstiness of the former, longer idle times after τ finished occur and thus fewer interrupts are triggered.

As described in Section V, the CPU load exceeds 1 when $\mathrm{E}[A]$ falls below 7.25 µs (cf. Equation 14). This fits with the observed packet loss at $\mathrm{E}[A] \leq 7\,\mu s$ for all distributions. In case of nbin, packet loss occurs even at $\mathrm{E}[A] = 8\,\mu s$ due to the higher burstiness of the traffic.

However, it is questionable whether this system can be operated in overload conditions with a packet loss ratio of more than 5 %, which occurs for interarrival times of 7 µs and less, corresponding to more than 142,857 packets per second. Thus, the lower rate with $\mathrm{E}[A] = 8\,\mu s$, when no packet loss occurs for all distributions *except* nbin, is more interesting. The reason for this behavior is, again, its burstiness and higher variation, resulting in short overload situations that lead to packet loss. In contrast, the other distributions result in more equally spaced arrivals.

For the largest aggregation interval of 500 µs, this effect is visible even for higher values of $\mathrm{E}[A]$. Caused by the higher expected number of packets per batch ($\tau / \mathrm{E}[A]$), the probability that packets are dropped in the central queue is increased, resulting in a higher packet loss ratio.

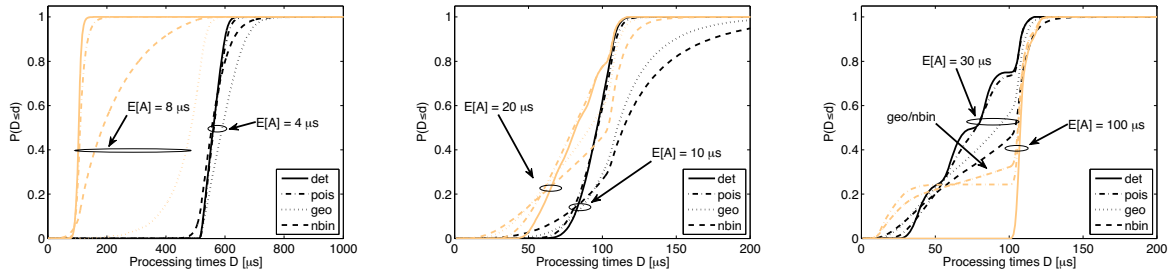
B. Processing Time Distributions for Varying Interarrival Times

In addition to studying the influence of the arrival process on the mean processing time, we also investigate its effect on the distribution of the processing time. Figure 10 shows the CDFs of the processing time D given an aggregation interval of $\tau = 100\,\mu s$ combined with different arrival processes and values for the mean interarrival time $\mathrm{E}[A]$.

For the lowest mean interarrival time of $4\,\mu s$ shown in Figure 10(a), i.e., the scenario with the highest system load, the highest processing times are observed. Furthermore, the distribution of processing times in this scenario has a low variance and similar values independent of the arrival process. This can be explained by the combination of the very high load and the fact that the system drops packets that encounter a full queue. In contrast, the distribution of the processing time in the context of $\mathrm{E}[A] = 8\,\mu s$ differs significantly across different distributions of the interarrival time. On the one hand, the relatively stable det and pois distributions result in a narrow range of processing times which is significantly lower than for $\mathrm{E}[A] = 4\,\mu s$. On the other hand, the higher degree of variation of the geo and nbin distributions result in a larger variety of batch sizes which, in turn, yield wide intervals of different processing times.

A further decrease of the processing times is observed for the medium interarrival times seen in Figure 10(b). In these scenarios, the distributions resulting from the nbin and geo distributions are closer to each other and begin to converge. This phenomenon can be explained by the evolution of the two arrival processes. For higher values of $\mathrm{E}[A]$, the coefficient of variation of geo approaches 1, i.e., that of the nbin distribution used in this work. Simultaneously, the r parameter of the nbin distribution approaches 1. Since the geometric distribution is a special case of the negative binomial distribution with $r = 1$, the aforementioned convergence can be explained.

Finally, Figure 10(c) displays the processing time distributions in case of $\mathrm{E}[A] = 30\,\mu s$ and $\mathrm{E}[A] = 100\,\mu s$, respectively. When the mean interarrival time equals the aggregation interval τ, only size 1 batches are processed in case of a deterministic arrival process. In combination with the fact that arrivals initiate the aggregation intervals, the processing time is dominated by the waiting time in the peripheral queue. For $\mathrm{E}[A] = 30\,\mu s$, batches consist of four packets, hence the distribution consists of four segments with similar shapes corresponding to a packet's position within a batch. The processing time distributions that result from a geometric and a negative binomial distribution converge further when $\mathrm{E}[A]$ is increased and overlap in case of $\mathrm{E}[A] = 100\,\mu s$. Processing times resulting from interarrival times that follow a Poisson distribution are lower and closer to those of det rather than geo and nbin which have a significantly higher degree of variation.



(a) Low interarrival times. (b) Medium interarrival times. (c) High interarrival times.

Fig. 10. Processing time distributions for varying packet interarrival times and different interarrival distributions in case of $\tau = 100\,\mu s$.

VII. Conclusion

NFV has many appealing advantages such as easy scale-up or scale-down of computing resources as well as scale-out or scale-in of virtual machines among the available physical hardware. This high flexibility, however, comes at the expense of performance, i.e., a lower packet throughput and longer processing delays. To understand the impact of performance-relevant parameters on these metrics, and in order to allow an adequate dimensioning and a proper performance prediction, appropriate performance models are required.

The contribution of this paper is an analytical model for *virtualized network functions* running in software on commodity hardware. The model takes into account interrupt moderation, a technique used by current operating systems and server hardware to reduce the overall number of interrupts. Based on an exemplary network function, a mobile network serving gateway, we determine an empirical service time distribution. We illustrate the applicability of the model by comparing it to measurements obtained from a testbed deployment of a VNF for a fixed aggregation interval and varying interarrival times. After that, the impact of different interarrival times, interarrival distributions, and aggregation interval durations on the processing times and the packet loss is presented. The proposed model also allows the computation of distributions, i.e., mean values, standard deviations, and also quantiles of the delay distributions can be computed. Therefore, the presented method can be used by administrators to ensure an appropriate operation of network functions based on their needs.

The model itself may be generalized to take into account acceleration techniques like Intel's DPDK or Cisco's VPP. This allows comparing heterogeneous network function implementations and selecting the appropriate technique for a specific use case. Furthermore, economic trade-offs between operational metrics and corresponding costs can be investigated.

Acknowledgment

This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), and it is partly funded by the German BMBF (Project ID 16KIS0474). The authors alone are responsible for the content of the paper.

References

[1] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast userspace packet processing. In *11th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, pages 5–16, Washington, DC, USA, 2015. IEEE Computer Society.

[2] Cisco Systems and Intel Corporation. NFV Partnership. Joint Whitepaper, http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cisco-nfv-partnership-paper.pdf, 2015.

[3] Steffen Gebert, Thomas Zinner, Stanislav Lange, Christian Schwartz, and Phuoc Tran-Gia. Discrete-Time Analysis: Deriving the Distribution of the Number of Events in an Arbitrarily Distributed Interval. Technical Report 498, June 2016. Available online: https://www3.informatik.uni-wuerzburg.de/TR/tr498.pdf.

[4] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, Feb 2015.

[5] Intel. Intel Data Plane Development Kit (DPDK). http://dpdk.org.

[6] Stanislav Lange, Anh Nguyen-Ngoc, Steffen Gebert, et al. Performance benchmarking of a software-based LTE SGW. In *2nd International Workshop on Management of SDN and NFV Systems*, Barcelona, Spain, November 2015.

[7] Steen Larsen, Parthasarathy Sarangam, Ram Huggahalli, and Siddharth Kulkarni. Architectural breakdown of end-to-end latency in a TCP/IP network. *International Journal of Parallel Programming*, 37(6), 2009.

[8] Robert Love. *Linux Kernel Development*. Addison-Wesley Professional, 3rd edition, 2010.

[9] David Manfield, Phuoc Tran-Gia, and Herbert Jans. Modelling and performance of inter-processor messaging in distributed systems. *Perform. Eval.*, 7, 1987.

[10] Joao Martins, Mohamed Ahmed, Costin Raiciu, et al. ClickOS and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473, Seattle, WA, April 2014. USENIX Association.

[11] Ravi Prasad, Manish Jain, and Constantinos Dovrolis. *Passive and Active Network Measurement: 5th International Workshop, PAM 2004, Antibes Juan-les-Pins, France, April 19-20, 2004. Proceedings*, chapter Effects of Interrupt Coalescence on Network Measurements, pages 247–256. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[12] Luigi Rizzo. netmap: A novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, Bellevue, WA, August 2012. USENIX Association.

[13] Sakir Sezer, Sandra Scott-Hayward, and Pushbinder Chouhan et al. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, July 2013.

[14] Benoit Sigoure. How long does it take to make a context switch? http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html, November 2014.

[15] Phuoc Tran-Gia. Zeitdiskrete Analyse verkehrstheoretischer Modelle in Rechner- und Kommunikationssystemen - 46. Bericht über verkehrstheoretische Arbeiten, 1988.

[16] Phuoc Tran-Gia. Discrete-time analysis technique and application to usage parameter control modelling in ATM systems. In *8th Australian Teletraffic Research Seminar*, Melbourne, Australia, December 1993.