A discrete-time model for optimizing the processing time of virtualized network functions

Thomas Zinner, Stefan Geissler, Stanislav Lange, Steffen Gebert, Michael Seufert, Phuoc Tran-Gia

Angaben zur Veröffentlichung / Publication details:

Zinner, Thomas, Stefan Geissler, Stanislav Lange, Steffen Gebert, Michael Seufert, and Phuoc Tran-Gia. 2017. "A discrete-time model for optimizing the processing time of virtualized network functions." *Computer Networks* 125: 4–14. https://doi.org/10.1016/j.comnet.2017.04.049.

A discrete-time model for optimizing the processing time of virtualized network functions

Thomas Zinner*, Stefan Geissler, Stanislav Lange, Steffen Gebert, Michael Seufert, Phuoc Tran-Gia

Julius-Maximilians-Universität Würzburg, Chair of Communication Networks

1. Introduction

The trend towards softwarization of networks, especially using Software Defined Networking (SDN) [1] and Network Functions Virtualization (NFV) [2], 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 possible.

This high flexibility, however, comes at the expense of performance [3,4], i.e., 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.

The contribution of this article is twofold. On the one hand, we extend the evaluation of the discrete-time model for Virtualized Network Functions (VNFs) running in software on commodity hardware that was developed in [5]. On the other hand, we significantly improve the computational efficiency of its numerical evaluations by utilizing a property regarding its limit behavior. In addition to proving this property, we also show that for realistic input parameters, convergence is reached and thus, the performance improvement does not affect the accuracy of the model.

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 the presented model, the impact of different interarrival times, interarrival distributions, and aggregation interval durations on processing times and packet loss ratios is presented. The proposed model also allows computing distributions, i.e., mean values, standard deviations, as well as quantiles of the delay distributions. In [5], we have already illustrated the applicability of the model by comparing it to mea-

* Corresponding author.
E-mail address: zinner@informatik.uni-wuerzburg.de (T. Zinner).

surements for a fixed aggregation interval and varying interarrival times using a mobile network Serving Gateway (SGW) as an exemplary network function.

Since the parameters of the interrupt moderation mechanism can severely affect the overall network performance, administrators can apply the model in order to find optimal values for their use cases. Additionally, the network conditions might change dynamically over time. Hence, a fast recalculation of optimal settings is required in order to adapt in a timely manner. This aspect is addressed by leveraging the abovementioned behavior regarding convergence, resulting in computations that are up to 22 times faster than in the context of the original model.

The remainder of this work is structured as follows: Background information as well as related work is introduced in Section 2. The steps involved in processing packets in a x86 system are described in Section 3, before an abstract model is introduced in Section 4. Afterwards, exemplary evaluations of the packet processing time and packet loss behavior under different settings are presented in Section 5. Finally, Section 6 draws conclusions and outlines future work.

2. Background & related work

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

2.1. 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 Network Interface Card (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 faster processing of packets in software have been introduced. These approaches, e.g., Netmap [6], ClickOS [7], Intel DPDK [8], or VPP [9] 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 [10].

However, the above mentioned 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 [11], 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 presented in this work is a first step towards enabling analytical evaluation of packet processing performance in commodity hardware running general purpose operating systems.

2.2. Interrupt moderation

One of the key features of the previously listed frameworks consists of avoiding livelocks [12] that result from the Central Processing Unit (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 in the Linux kernel (*New API*, short NAPI [12]) 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 has been processed, or a timeout occurs, interrupts are re-enabled and the process restarts with the next packet arrival.

Hardware-based implementations of interrupt moderation are supported by 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 preconfigured number of packets is received or sent, or until a preconfigured 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.

3. 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 Fig. 1.

Receive incoming packet. The network interface card reads data from the transmission media, transforms it into packets and stores it into a receive queue.

Trigger copy to kernel via interrupt. The NIC triggers an interrupt to notify the CPU about the arrival of a packet. The interrupt starts an expensive copy process that moves the data from the network card into the address space of the kernel.

Store packet in RAM. The data is stored in a buffer until the application requests it for processing. The size of this buffer is limited to a fixed number of bytes. If the application cannot catch up with reading, the kernel drops packets.

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

Send outgoing packet. After processing, the packet traverses the same way backwards until it is finally sent to media.

4. Model

The queuing model used for the performance analysis of the system outlined in Section 3 is depicted in Fig. 2. It is a generalization of the clocked approach introduced by Manfield et al. [13].

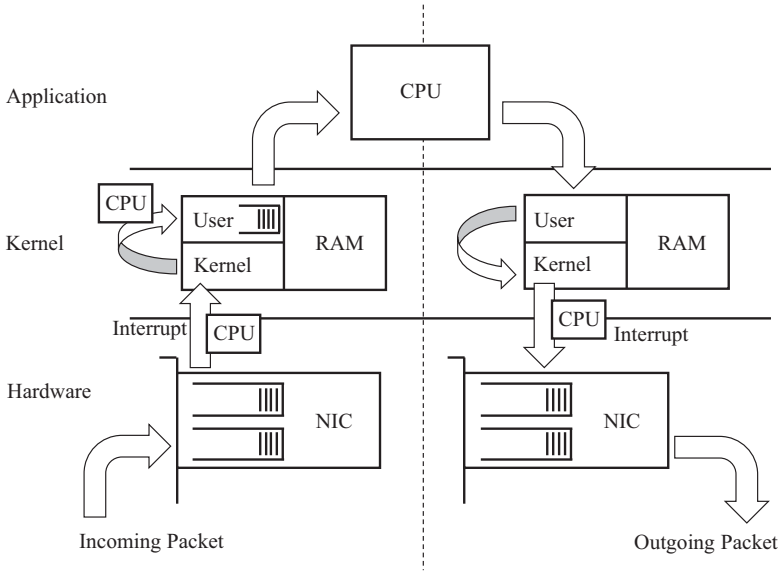


Fig. 1. Packet processing in an x86 server running Linux.

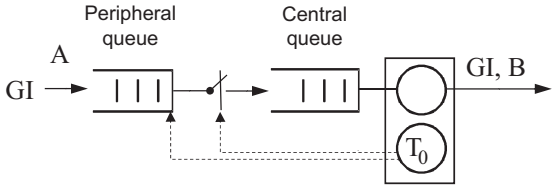


Fig. 2. Queueing model.

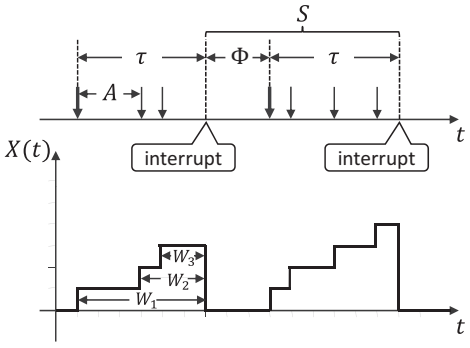


Fig. 3. Exemplary development of the peripheral queue and involved random variables.

The generation of packets follows an arbitrary distribution A. For the model presented in this work, we assume independent packet interarrivals. The packets are stored in a peripheral queue which is assumed to have infinite size. This queue corresponds to the NIC queue displayed in Fig. 1. Incoming packets are transferred in a batch to the central queue (cf. RAM in Fig. 1) 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 following. The next sections introduce the model of the peripheral as well as central queue and finally arrive at a combined model for the whole system.

4.1. 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

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

Although the random variables considered in this work only take on positive values, distributions are defined over the range from $-\infty$ to ∞ in order to enable operations like computing the distribution of the difference of two random variables. Furthermore, the distribution function of X is written as $X(k)$ and is defined as

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

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

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

The following distributions are used for modeling the peripheral queue. Additionally, an exemplary development of the queue is shown in Fig. 3, which highlights the relationships between different RVs.

- $a(k)$: distribution of the packet interarrival time.
- $r_a(k)$: distribution of the packet recurrence time. The recurrence time is defined as the duration between a randomly chosen time and the arrival of the next packet.
- $\tau(k)$: distribution of the duration of the aggregation interval. For the remainder of this work, we use a constant aggregation interval of length τ. That is, $\tau(k) = \delta(\tau)$ where $\delta(\tau)$ denotes the Dirac impulse at τ.
- $f^{(j)}(k)$: distribution of the time between the start of an aggregation interval and the arrival of the jth 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 $F^{(j)}$.

- $x(k)$: distribution of the batch size.
- $\phi(k, \tau)$: distribution of the time between the end of an aggregation interval of length τ and the arrival of the packet that initiates the next aggregation interval.
- $s(k)$: distribution of the interarrival time between batches.
- $o(k)$: distribution of the interrupt processing delay.
- $w_i(k)$: distribution of the waiting time of the ith packet in the peripheral queue.
- $u_n(k)$: distribution of unfinished work in the system before the arrival of the nth batch.

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 [14] and [15], the batch size distribution $x(k)$ can be computed as follows.

$$x(k) = \tau(0)\delta_0(k) + \sum_{m=1}^{\infty} \tau(m) \sum_{i=0}^{m-1} \left(f^{(k)}(i) - f^{(k+1)}(i) \right), \quad k = 0, 1, \ldots. \tag{1}$$

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_0 serves as indicator and is defined formally in Eq. 2. The law of total probability is used in the second term in order to calculate the conditional probability $x(k|m)$ for the remaining interval lengths. It can be derived from the relationship shown in Eq. 3.

$$\delta_0(k) = \begin{cases} 1 & k = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

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

According to the above description of the distribution $\phi(k, \tau)$, it can be computed using Eq. 4. In particular, the equation states that in order to observe $\Phi = k$, two conditions need to be met. First, the sum of n interarrival times needs to be equal to $\tau + k$. Second, the last interarrival time needs to be larger than k in order to ensure that only the last arrival is outside the aggregation interval. Finally, the sum over all possible numbers of events n yields the desired probability.

$$\phi(k, \tau) = \mathrm{P}(A > k) \sum_{n=1}^{\infty} f^{(n)}(\tau + k) \tag{4}$$

Since the first packet after a transfer initiates the next aggregation interval, the batch interarrival time s can be calculated as the sum of Φ and the duration of the aggregation interval τ:

$$s(k) = \phi(k, \tau) * \tau(k). \tag{5}$$

Furthermore, the waiting time of consecutive packets, after the interval is started, is reduced. In particular, the waiting time of the ith packet of a batch in the peripheral queue depends on the arrivals of the $i-1$ packets before it. Hence, the distribution of its waiting time can be computed as follows:

$$w_i(k) = \pi_0 \left[\tau(k) * \underbrace{a(-k) * \cdots * a(-k)}_{\text{(i - 1) times}} \right] \tag{6}$$

In [5], we assumed that Φ follows the same distribution as the packet recurrence time R_a. We now show that this is only true when considering limits. In particular, we prove the following theorem in the appendix. Note that this theorem does not apply to deterministic packet interarrivals with an average interarrival time greater than 1. This stems from the fact that the corresponding distribution is arithmetic and has a span greater than 1.

Theorem 1. *For non-arithmetic packet interarrival times and arithmetic packet interarrival times with a span equal to 1, the distribution of the time between consecutive batches, $\phi(k, \tau)$, converges to the recurrence time of the interarrival time, $r_a(k)$, as τ approaches infinity, i.e.,*

$$\lim_{\tau \to \infty} \phi(k, \tau) = r_a(k).$$

However, evaluations presented in Section 5.3 demonstrate that in the context of realistic input parameters, convergence is reached and the assumption of equality does not impact the accuracy of performance indicators like packet loss or overall processing time. Additionally, in contrast to calculating the distribution $\phi(k, \tau)$, the calculation of the recurrence time does not require any convolution operations and thus allows for significantly faster numerical evaluations of the model.

4.2. 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 by a maximum value of L, i.e., packets that 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 ith packet of the nth batch.
- $B_{n,\,i}$: RV for the service time of the ith packet of the nth 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 Fig. 4. Observing the packets of the nth batch arrival, the ith 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:

$$u_{n,b_1}(k) = u_n(k) \tag{7}$$

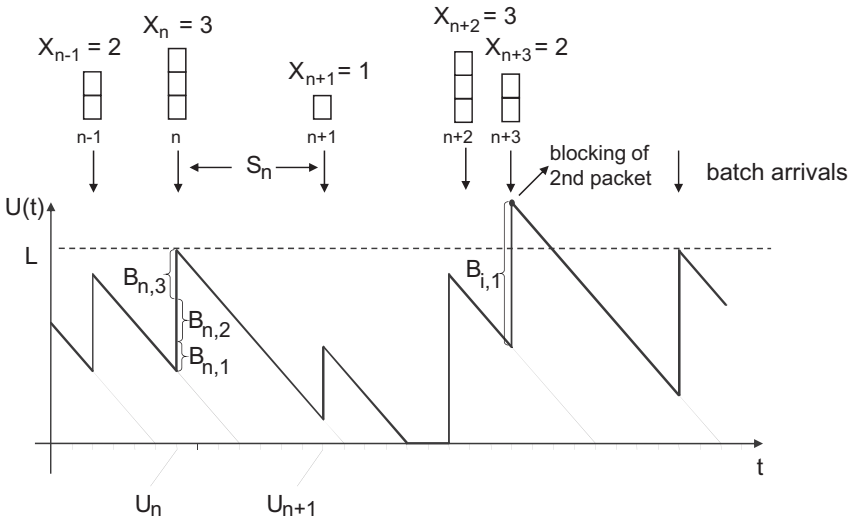


Fig. 4. Exemplary system development for $Gl^{[X]}/GI/1-L$ with bounded delay.

$$u_{n,b_{i+1}}(k) = \sigma^{L-1}\left[u_{n,b_i}(k)\right] * b_{n,i}(k) + \sigma_L\left[u_{n,b_i}(k)\right] \tag{8}$$

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

$$u_{n+1}(k) = \pi_0\left[\left(\sum_{i=1}^{\infty} x(i)\cdot u_{n,b_i}(k)\right) * s_n(-k) * o(k)\right]. \tag{9}$$

In this calculation, the interrupt overhead o is added to the batch interarrival time s due to the fact that for each batch, the CPU has to devote time to handle this interrupt instead of processing packets.

Using the above equations, an algorithm for calculating the workload prior to the ith 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. Eq. 10. 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 nth batch. Similarly to Eq. 10, the index n can also be suppressed under stationary conditions, resulting in RV B.

$$\begin{aligned} u(k) &= \lim_{n\to\infty} u_n(k) \\ u_{b_i}(k) &= \lim_{n\to\infty} u_{n,b_i}(k) \end{aligned} \tag{10}$$

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 processed in this interval. In particular, we observe that the amount of work arriving within a batch interarrival time depends on the batch size and the packet service time (cf. Eq. 11). Note that both the batch size and the batch interarrival time are affected by the packet interarrival time (cf. Eqs. 1 and 5).

$$\rho = \frac{\mathrm{E}[X]\mathrm{E}[B]}{\mathrm{E}[S]} \tag{11}$$

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

$$p_b = \sum_{i=1}^{\infty}\left(\frac{1}{i}x(i)\cdot\sum_{j=L}^{\infty}u_{b_i}(j)\right) \tag{12}$$

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.

4.3. Combined model

Using the two models described in Sections 4.1 and 4.2, 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 ith packet in a batch d_i:

$$d_i(k) = w_i(k) * u(k) * \underbrace{b(k) * \cdots * b(k)}_{\text{i times}} \tag{13}$$

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

$$d(k) = \sum_{i=1}^{\infty}\mathrm{P}(X=i)\cdot d_i(k) = \sum_{i=1}^{\infty}x(i)\cdot d_i(k) \tag{14}$$

The applicability of the proposed model has already been shown in [5].

5. Evaluation

In this section, we investigate the behavior of a 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 length of the aggregation interval τ is studied in the context of different amounts of load that is offered to the system. Since the load of the entire system can not be calculated easily in advance, we vary the normalized arrival rate $\alpha = \frac{\mathrm{E}[B]}{\mathrm{E}[A]}$ instead. This quantity contains the main contributors to the system load ρ and thus, provides a good estimate.

At first, coarse-grained analyses of the resulting mean processing times and packet loss ratios for different parameter combinations are presented. Afterwards, we evaluate the impact of using the assumption that the time between the end of an aggregation interval and the beginning of the next interval Φ, is distributed according to the recurrence time of packet interarrivals, R_a. On the one hand, we quantify the difference between the two distributions directly. On the other hand, we investigate the resulting impact on the total processing time and the packet loss, which are calculated based on these distributions.

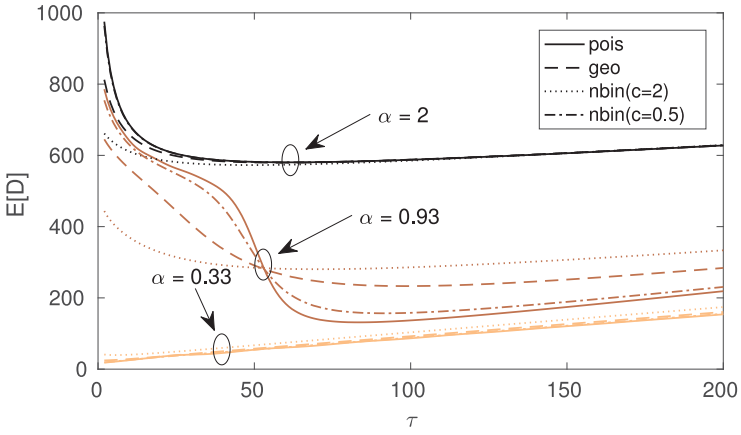


Fig. 5. Effects of different aggregation interval lengths τ and normalized arrival rates α on the mean processing time E[D].

5.1. Impact of the aggregation interval

The sensitivity of the modeled system to different aggregation intervals τ is studied based on four different distributions, namely Poisson (pois), geometric (geo), and negative binomial (nbin). While for pois and geo, the distributions are characterized solely by E[A], the parameters p and r of nbin are adjusted in such a manner that coefficients of variation $c_A = 0.5$ and $c_A = 2$ are achieved.

5.1.1. Impact on mean processing times

Fig. 5 presents the mean packet processing time D that results from different combinations of the aggregation interval τ, normalized arrival rate α as well as different interarrival distributions. In the calculations, the processing time at the CPU follows a Poisson distribution and we use a fixed value for its mean, i.e., E[B] = 10 μs. Then, the different values for α are achieved by setting the mean interarrival time E[A] to values in {5, 10.75, 30}, respectively. While the x-axis displays the length of the aggregation interval, the y-axis indicates the average packet processing time. Additionally, line colors represent different values of the normalized arrival rate α and line styles correspond to the four distribution types.

In the context of a low normalized arrival rate, an almost strictly linear relationship between the length of the aggregation interval and the total processing time can be observed for all distributions. In these cases, the CPU operates at a low load and thus, can finish work faster than it arrives. Therefore, the waiting time at the peripheral queue constitutes the main influence factor on the total processing time. This waiting time is directly affected by τ, hence the observed relationship.

When exposed to a high value of $\alpha = 2$, the processing time is also almost independent of the underlying distribution. However, the resulting processing times are significantly higher and the effect of τ is not linear. Due to the fact that the CPU is overloaded in this scenario, the total processing time is also affected by the increased waiting time in the central queue. For low values of τ, the overhead associated with frequent interrupts results in the highest total processing time. When τ is increased, the overhead is reduced and multiple packets are handled with a single interrupt. In the presented case, the lowest processing time is achieved with values of τ at around 80 μs. When τ is increased further, the processing time increases due to the growing waiting time at the peripheral queue.

Finally, the processing time for $\alpha = 0.93$ depends on the underlying distribution of packet interarrival times and attains values between those for $\alpha = 0.33$ and $\alpha = 2$. As in the previous case, low values of τ result in a high overhead and a high total processing time while high values of τ increase the waiting time in the peripheral queue. Between these extremes, the shape of the curves is mainly determined by the variability of the packet arrival distribution. For distributions with a low coefficient of variation, i.e., pois and geo, bends can be identified when τ attains values that are integer multiples of E[A]. At these points, the expected number of events in an aggregation interval increases by one. Thus, the last packet of each batch has a low waiting time in the peripheral queue while there are no significant changes for the remaining packets. Therefore, the average total processing time is decreased. In contrast, distributions with a higher coefficient of variation lead to smoother curves and attain values that lie between the aforementioned bends. For each distribution, there is an optimum value for τ which minimizes the resulting mean processing time. Hence, the model can be used by network operators in order to optimize the processing times of their VNFs based on traffic characteristics observed in their particular network.

5.1.2. 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 the central queue is limited, packet loss occurs once this limit is exceeded. In the following, we evaluate the average packet loss probability for different sizes of the aggregation interval τ and distributions of the interarrival time.

In Fig. 6, the x-axis represents the length of the aggregation interval τ and the y-axis displays the average packet loss probability. Different packet arrival processes are highlighted by means of different line shapes and different values for the normalized arrival rate α are denoted by their color. The presented results are based on the same parameter combinations as in Fig. 5.

Similar to the observations regarding the average processing time, the case of $\alpha = 0.33$ does not stress the system to a large enough extent, so that packet loss does not occur. When the system is in a situation of overload with $\alpha = 2$, packet loss always occurs, but decreases for larger values of τ. As stated before, choosing short aggregation intervals results in a higher interrupt rate and thus, in more overhead. Hence, the CPU has even less time to actually process packets, which leads to more congestion at the inner queue and finally, a larger packet loss rate.

Furthermore, the packet loss probability depends on the arrival distribution. In particular, the distribution with the lowest coefficient of variation leads to the highest packet loss probability and vice versa. This is also true for $\alpha = 0.93$. This phenomenon can be explained with the effect of the arrival distribution on the service time and the resulting overhead per packet. In case of high vari-

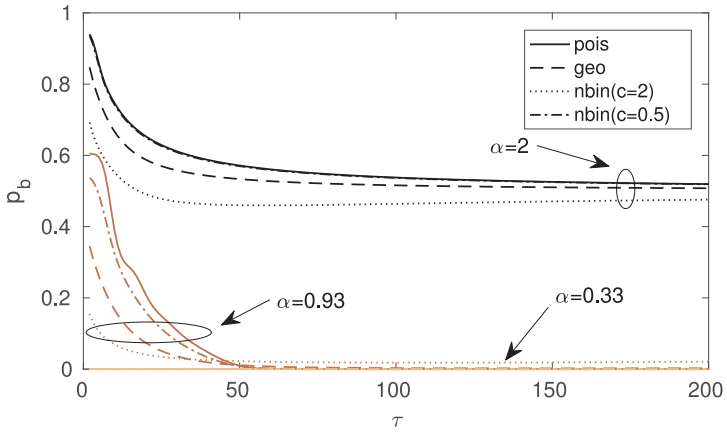
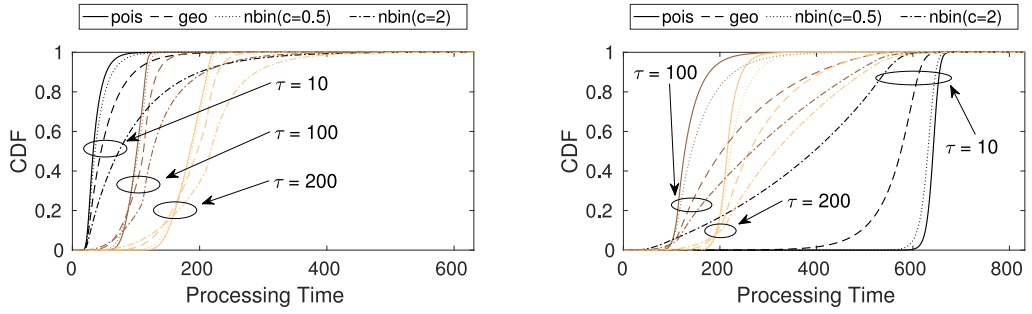


Fig. 6. Effects of different aggregation interval lengths τ and normalized arrival rates α on the packet loss probability p_b.



(a) Low normalized arrival rate ($\alpha = 0.66$). (b) High normalized arrival rate ($\alpha = 0.93$).

Fig. 7. Processing time distributions for different aggregation interval lengths and different interarrival distributions.

ance, the arrival process is burstier, resulting in more packets per batch. However, the heavy tail of the corresponding distributions also leads to longer service times due to a higher probability of very long interarrival times between batches. This, in turn, results in longer times between consecutive batches Φ. Consequently, only a single interrupt is triggered for such a batch. If, on the other hand, the variance of the packet interarrival time is low, batches contain fewer packets and service times are shorter. Thus, more interrupts are triggered during the same time, resulting in more overhead per processed packet.

5.2. Processing time distributions for varying aggregation intervals

In addition to studying the influence of the length of the aggregation interval τ on the mean processing time, we also investigate its effect on the distribution of the processing time. Fig. 7 shows the distribution of the processing time D for different τ and normalized arrival rates α.

Regarding low expected system load, Fig. 7(a) shows the processing time distribution for three different aggregation intervals, i.e, $\tau = 10, 100, 200\,\mu s$, and four different arrival distributions. In this case it can be seen that the processing time distribution is clustered by the selected aggregation interval length. This indicates that, due to spare computational resources, the system can handle the variation of the number of incoming packets up to a certain point. This is supported by the fact that the two distributions with low coefficients of variation, i.e. pois and nbin($c = 0.5$), result in similar processing time distributions with a small variance in values. The two distributions with a higher coefficient of variation, geo and nbin($c = 2$), on the other hand result in processing time distributions with a larger range of values. In particular, the significantly higher variance observed for nbin($c = 2$) can lead to increased processing times. This can be explained by the high variation of the batch size.

Finally, the figure confirms the continuous increment of the mean processing time with growing τ in low load scenarios.

Observing the processing time distribution in a scenario with high expected system load, i.e., a normalized arrival rate of $\alpha = 0.93$ yields entirely different results, as shown in Fig. 7(b). First, the previously observed clusters are, although still present, less significant. Furthermore, it can be seen that an aggregation interval of $\tau = 10$ now results in the highest processing times. This can be explained by the overhead resulting from short aggregation intervals that, especially in the high load scenario, leads to prolonged processing times. As the CPU is already busy handling the actual workload, the overhead hits harder in this scenario and thereby the processing times are significantly increased. Moreover, it can again be observed that the distributions featuring low coefficients of variation also result in a processing time distribution showing lower variance. The same holds true for the two distributions with a high coefficient of variation, as these result in processing time distributions with a high range of values.

5.3. Impact of the recurrence time assumption

In the proof of Theorem 1, we show that the time between the end of an aggregation interval and the next packet arrival converges to the recurrence time of the packet interarrival time as the length of the aggregation interval τ approaches infinity. Hence, for large enough τ, the expected error that is caused by assuming $\phi(k, \tau) = r_a(k)$ converges to zero. In this section, we quantify the errors that are caused by using this assumption in the context of realistic parameter combinations.

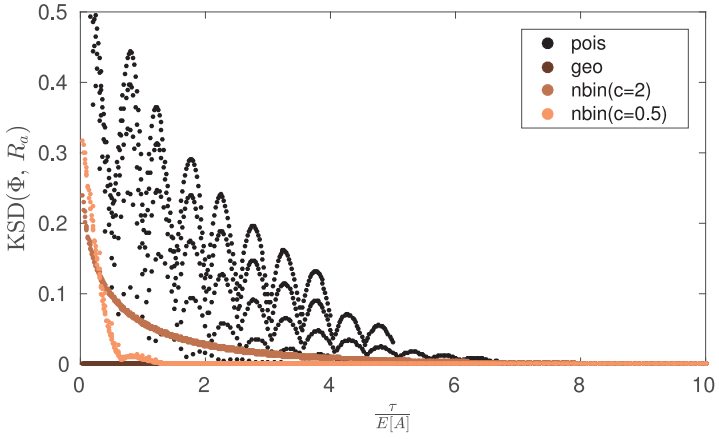


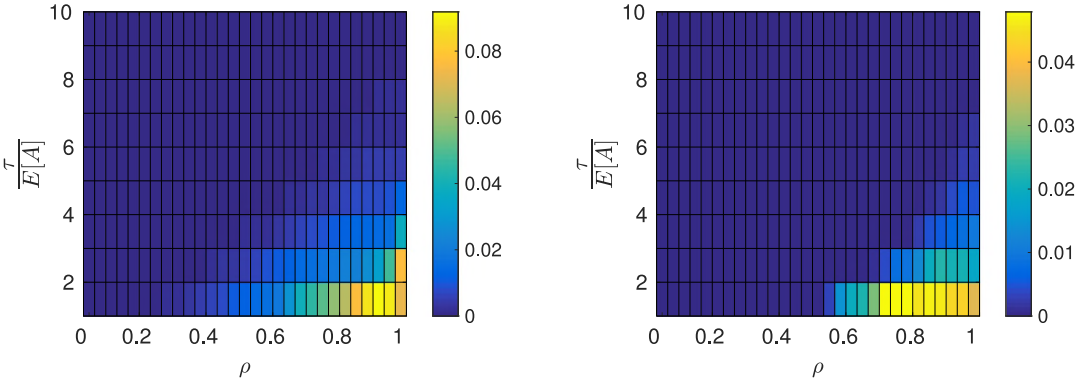
Fig. 8. Kolmogorov–Smirnov distance between the distributions of the recurrence time of packet interarrivals, R_a, and the actual distribution of the time between the end of an aggregation and the next packet arrival, Φ.

Fig. 8 displays the difference between the distributions $\phi(k, \tau)$ and $r_a(k)$ for different combinations of τ, E[A], and packet interarrival distributions. In this case, the difference is expressed in terms of the Kolmogorov-Smirnov distance (KSD) between the corresponding cumulative distribution functions and is shown on the y-axis. On the x-axis, the ratio between τ and E[A] represents the average number of events during an aggregation interval. For these evaluations, the mean interarrival time was chosen so that E[A] $\in$ {10, 20, 30, 40, 50} and the length of the aggregation interval τ ranged from 2 to 200.

Due to the property of memorylessness of the geometric distribution, the equality $\phi(k, \tau) = r_a(k)$ is always true for this distribution, resulting in a constant KSD of zero. For the remaining distributions, the KSD equals zero as soon as more than an average of seven arrivals fit into an aggregation interval. Since the negative binomial distribution has a higher variance than the Poisson distribution, the former converges faster as the end of an aggregation interval becomes equivalent to an independent observer more quickly. On the other hand, it can be seen that the negative exponential distribution with a coefficient of variance of $c = 2$ takes longer to converge as the same distribution with a coefficient of variance of $c = 0.5$. This indicates that the shape of the distribution also influences the convergence process. The different allocation of probability mass among the possible values of the distribution also influences the speed of convergence. Hence, the variance of a distribution in general is not sufficient to make an absolute statement about the speed of the convergence, but can be used as a reliable indicator towards the behavior of the process. Finally, interarrival times that follow a Poisson distribution cause the slowest convergence due to their low variance and narrow range of attained values. Additionally, an alternating behavior of the KSD can be observed in case of the Poisson distribution when τ is an integer multiple of E[A]. This causes $\phi(k, \tau)$ to be more similar to $a(k)$ rather than $r_a(k)$.

Fig. 8 shows the difference between the distributions $\phi(k, \tau)$ and $r_a(k)$. However, we are more interested in the impact on the accuracy of the model's prediction of performance indicators that is caused by using the above mentioned assumption. Therefore, we show the effects regarding the mean processing time and the packet loss probability in Fig. 9. In particular, it displays an overview of the difference between the actual values and the values when using the assumption that the time between the end of an aggregation interval and the recurrence time of packet arrivals follow the same distribution. In case of the mean total processing time, the difference is normalized by dividing it by the actual mean value, whereas the difference between the two packet loss probabilities is guaranteed to range between zero and one. Again, different parameter combinations regarding τ, E[A], and packet interarrival distributions are used. Additionally, we vary the mean packet processing time at the CPU, E[B], in order to achieve different levels of system load ρ (cf. Eq. 11). In particular, the following parameter combinations are used.

- The length of the aggregation interval τ is set to values in {100, 150, . . . , 500}.



(a) Error w.r.t. average processing time.

(b) Error w.r.t. packet loss probability.

Fig. 9. Impact of the recurrence time assumption on E[D] and p_b.

- The mean interarrival time E[A] is chosen so that the ratio $\frac{\tau}{\mathrm{E}[A]}$ ranges from 1 to 10. Like in all presented evaluations, four interarrival distributions are used, i.e., pois, geo, nbin($c = 0.5$), and nbin($c = 2$).
- The processing time at the CPU follows a Poisson distribution whose mean value E[B] is varied so that the normalized arrival rate $\frac{\mathrm{E}[B]}{\mathrm{E}[A]}$ attains values in $\{0.03, 0.06, \ldots, 1.02\}$.

Both, Fig. 9(a) and (b) present the corresponding maximum error as a function of the system load ρ, which is represented by the x-axis, and the ratio between τ and E[A] on the y-axis. The color of each cell denotes the error, with dark blue cells indicating an error of zero and light yellow colored cells indicating the largest observed error. Qualitatively, both figures show a similar behavior. That is, the observed error is zero in most cases except for those where ρ is high, i.e., 70% and larger, and the expected number of arrivals during an aggregation interval is low, i.e., below four. In the case of the mean total processing time, the largest relative error is around 9%, while the largest deviation in terms of the packet loss probability is around 5%.

Since the goal of interrupt mitigation mechanisms is to avoid frequent interrupts, the ratio between τ and E[A] is usually high. Furthermore, network operators tend to dimension their infrastructure for average load levels far below 90% [17]. Hence, the results shown in Fig. 9(a) and (b) confirm that in the context of realistic parameter combinations, using the significantly more efficient calculation of $r_a(k)$ instead of $\phi(k, \tau)$ does not impact the accuracy of the model's predictions. At the same time, the computation time can be reduced by a factor of up to 22 in the context of the investigated cases.

6. Conclusion

NFV has many appealing advantages such as easy scale-up or scale-down of compute 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., 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 primary contribution of this article is an analytical model for *virtualized network functions* running in software on commodity hardware. Given the characteristics of the network traffic and the utilized VNFs, this model allows network administrators to identify optimal parameters for the interrupt moderation mechanisms that are used by modern operating systems and network interface cards. A central benefit is that the model supports arbitrary distributions for the packet interarrival and packet service times. Hence, administrators only need to determine empirical distributions and do not need to fit observed values to a predefined set of known distributions.

Furthermore, we show that approximating the time between consecutive batches with the recurrence time of the interarrival process only introduces a negligible error while providing a significant speedup when performing numerical evaluations of the model. This provides network administrators with the ability to quickly optimize their system parameters in a dynamic fashion when network conditions change over time.

We perform an in-depth evaluation of the model by investigating the impact of different load levels, aggregation interval durations, and interarrival distributions on processing times and packet loss ratios. The proposed model also allows the computation of distributions of these performance indicators, i.e., it is possible to determine mean values, standard deviations, as well as quantiles. Therefore, the presented model 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 Vector Packet Processing (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. Additionally, the authors would like to thank Michael Schönlein for assisting with the validation of the presented proofs.

Appendix A. Proofs

Lemma 1. *Let A be independent and identically distributed (i.i.d.) interarrival times with* E[A] > 0. *Then, the expected number of renewal events during a finite interval $t < \infty$, defined by the renewal function $m(t)$, is also finite, i.e.,*

$$m(t) < \infty.$$

Proof. Let A be the RV that describes the interarrival time between consecutive events with E[A] > 0. The corresponding interarrival times are denoted as A_i for $i \in \mathbb{N}$. Let now A' generate the following interarrival times.

$$A'_i = \begin{cases} 0 \text{ if } A_i = 0 \\ 1 \text{ otherwise} \end{cases}.$$

We then get

$$\forall i \;\; A'_i \leq A_i.$$

Consequently, we can define the number of events during t with arrivals according to A' as

$$N'(t) = \max\{n : A'_1 + A'_2 + \ldots + A'_n \leq t\}.$$

Equally, the same can be defined for the original distribution of interarrival times that are based on A.

$$N(t) = \max\{n : A_1 + A_2 + \ldots + A_n \leq t\}.$$

It follows that

$$N(t) \leq N'(t)$$

and consequently

$$\mathrm{E}[N(t)] \leq \mathrm{E}[N'(t)]. \tag{A.1}$$

Let now $X = N'(1)$ be the number of events in an interval of length 1. We get

$$\begin{aligned} \mathrm{E}[X] &= \sum_{k=1}^{\infty} k\mathrm{P}\big(A' < 1\big)^{k-1}\mathrm{P}\big(A' \geq 1\big) \\ &= \mathrm{P}\big(A' \geq 1\big)\sum_{k=1}^{\infty} k\mathrm{P}\big(A' < 1\big)^{k-1} \\ &= \frac{\mathrm{P}\big(A' \geq 1\big)}{\mathrm{P}(A' < 1)}\sum_{k=0}^{\infty} k\mathrm{P}\big(A' < 1\big)^{k}. \end{aligned}$$

Since $\mathrm{P}(A' < 1)$ is a probability, $\mathrm{P}(A' < 1) \leq 1$ holds and thus, we can apply the formula for the geometric progression, arriving at

$$\frac{\mathrm{P}(A' \geq 1)}{\mathrm{P}(A' < 1)}\frac{\mathrm{P}(A' < 1)}{(1 - \mathrm{P}(A' < 1))^2} = \frac{\mathrm{P}(A' \geq 1)}{\mathrm{P}(A' \geq 1)^2} = \frac{1}{\mathrm{P}(A' \geq 1)}.$$

Since $P(A' \geq 1) > 0$, it follows that $E[X] < \infty$. We can now extend the interval to an arbitrary length $t < \infty$ and get

$$E[N'(t)] = tE[X].$$

We can thus conclude that $E[N'(t)] < \infty$. Using A.1 we finally arrive at

$$m(t) = E[N(t)] \leq E[N'(t)] < \infty$$

□

Lemma 2. *We consider the renewal function $m(t)$ which is used to determine the average number of renewals during time t. If the mean interarrival time, $E[A]$, is greater than 0, the following relationship holds for the renewal function $m(t)$:*

$$m(\tau + k) = \sum_{n=1}^{\infty} f^{(n)}(\tau + k) + m(\tau + k - 1).$$

Proof. For positive arrivals that are i.i.d., the renewal function $m(t)$ at $t = \tau + k$ is defined as

$$m(\tau + k) := \sum_{n=1}^{\infty} F^{(n)}(\tau + k).$$

Using the relationship between the cumulative distribution function F and the probability density function f, we get

$$\sum_{n=1}^{\infty} F^{(n)}(\tau + k) = \sum_{n=1}^{\infty} \sum_{i=0}^{\tau+k} f^{(n)}(i).$$

From the proof of Lemma 1, we conclude that $m(\tau + k) < \infty$. Since $F^{(n)}(t)$ is a cumulative distribution function, $F^{(n)}(\tau + k) \geq 0$ holds as well. Therefore, the convergence of $m(\tau + k)$ is absolute. Hence, the order of the summation can be switched, i.e.,

$$\sum_{n=1}^{\infty} \sum_{i=0}^{\tau+k} f^{(n)}(i) = \sum_{i=0}^{\tau+k} \sum_{n=1}^{\infty} f^{(n)}(i).$$

Finally, the first element of the outer sum can be separated while the remaining elements form $m(\tau + k - 1)$:

$$\begin{aligned} \sum_{i=0}^{\tau+k} \sum_{n=1}^{\infty} f^{(n)}(i) &= \sum_{n=1}^{\infty} f^{(n)}(\tau + k) + \sum_{i=0}^{\tau+k-1} \sum_{n=1}^{\infty} f^{(n)}(i) \\ &= \sum_{n=1}^{\infty} f^{(n)}(\tau + k) + m(\tau + k - 1). \end{aligned}$$

□

Proof. Using the definition of $\phi(k, \tau)$, we obtain

$$\lim_{\tau \to \infty} \phi(k, \tau) = \lim_{\tau \to \infty} \left(P(A > k) \sum_{n=1}^{\infty} f^{(n)}(\tau + k) \right).$$

Furthermore, the relationship shown in the proof of Lemma 2 allows representing the sum in terms of the renewal function $m(t)$.

$$\lim_{\tau \to \infty} \left(P(A > k) \sum_{n=1}^{\infty} f^{(n)}(\tau + k) \right) = P(A > k) \lim_{\tau \to \infty} (m(\tau + k) - m(\tau + k - 1)).$$

Finally, using Blackwell's renewal theorem, we can show the desired equality for non-arithmetic inter-renewal distributions as well as arithmetic distributions with a span equal to 1:

$$P(A > k) \lim_{\tau \to \infty} (m(\tau + k) - m(\tau + k - 1)) = P(A > k) \frac{1}{E[A]} = r_a(k).$$

□

References

[1] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, W. Kellerer, Interfaces, attributes, and use cases: acompass for SDN, IEEE Commun. Mag. (2014).

[2] M. Chiosi, et al., Network functions virtualisation - introductory white paper, 2012, (http://portal.etsi.org/NFV/NFV_White_Paper.pdf).

[3] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: challenges and opportunities for innovations, IEEE Commun. Mag. 53 (2) (2015) 90–97, doi:10.1109/MCOM.2015.7045396.

[4] S. Sezer, S. Scott-Hayward, P. Chouhan, et al., Are we ready for SDN? implementation challenges for software-defined networks, IEEE Commun. Mag. 51 (7) (2013) 36–43, doi:10.1109/MCOM.2013.6553676.

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

[6] L. Rizzo, Netmap: a novel framework for fast packet I/O, in: 21st USENIX Security Symposium (USENIX Security 12), USENIX Association, Bellevue, WA, 2012, pp. 101–112. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/rizzo.

[7] J. Martins, M. Ahmed, C. Raiciu, et al., ClickOS and the art of network function virtualization, in: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), USENIX Association, Seattle, WA, 2014, pp. 459–473. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins.

[8] Intel, Intel data plane development kit (DPDK), (http://dpdk.org).

[9] Cisco Systems and Intel Corporation, NFV partnership, joint whitepaper, 2015, (http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cisco-nfv-partnership-paper.pdf).

[10] T. Barbette, C. Soldani, L. Mathy, Fast userspace packet processing, in: 11th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, in: ANCS '15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 5–16. http://dl.acm.org/citation.cfm?id=2772722.2772727.

[11] S. Lange, A. Nguyen-Ngoc, S. Gebert, T. Zinner, M. Jarschel, A. Köpsel, M. Sune, D. Raumer, S. Gallenmüller, G. Carle, et al., Performance benchmarking of a software-based LTE SGW, in: Network and Service Management (CNSM), 2015 11th International Conference on, IEEE, 2015, pp. 378–383.

[12] R. Love, Linux Kernel Development, 3rd, Addison-Wesley Professional, 2010.

[13] D. Manfield, P. Tran-Gia, H. Jans, Modelling and performance of inter-processor messaging in distributed systems, Perform. Eval. 7 (1987).

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

[15] S. Gebert, T. Zinner, S. Lange, C. Schwartz, P. Tran-Gia, Discrete-time analysis: deriving the distribution of the number of events in an arbitrarily distributed interval, Technical Report 498, 2016. Available online: https://www3.informatik.uni-wuerzburg.de/TR/tr498.pdf.

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

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: experience with a globally-deployed software defined WAN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14.

Thomas Zinner studied computer science at the University of Würzburg, Germany. He finished his Ph.D on performance modeling of QoE-aware multipath video transmission in the future Internet in 2012. He is heading now the "Next Generation Networks" research group at the Chair of Communication Networks in Würzburg. His main research interests cover video streaming techniques, implementation of QoE-awareness within networks, Software-Defined Networking (SDN) and network virtualization, network function virtualization and the benefits of cloudification, as well as the performance assessment of these technologies and architetures.

Stefan Geissler is working towards his Ph.D at the University of Würzburg, Germany, where he also completed his Master's degree in 2016. His research topics include software defined networking and network function virtualization with focus on performance evaluation.

Stanislav Lange studied computer science at the University of Würzburg, Germany, where he received his M.Sc. degree in 2014. Currently, he is a researcher in the "Next Generation Networks" research group at the Chair of Communication Networks in Würzburg and is pursuing his PhD. His research is focused on software defined networking, performance analysis, system modeling, as well as multiobjective optimization.

Steffen Gebert is working towards his Ph.D at the University of Würzburg Germany, where he also received his Diploma degree in 2011. His research interests include softwarized networks and agile network operations.

Michael Seufert studied computer science, mathematics, and education at the University of Würzburg, Germany. In 2011, he received his Diploma degree in computer science, and additionally passed the first state examinations for teaching mathematics and computer science in secondary schools. From 2012–2013, he was with FTW Telecommunication Research Center, Vienna, Austria, working in the area of user-centered interaction and communication economics. He is currently a Researcher at the Chair of Communication Networks, University of Würzburg. His research mainly focuses on QoE of Internet applications, traffic management solutions, monitoring and orchestration of edge cloud services, and performance analysis and modeling of communication systems.

Phuoc Tran-Gia is professor and director of the Chair of Communication Networks, University of Würzburg, Germany. He is also Member of the Advisory Board of Infosim (Germany) specialized in IP network management products and services. Prof. Tran-Gia is also cofounder and board member of Weblabcenter Inc. (Dallas, Texas), specialized in Crowdsourcing technologies. Previously he was at academia in Stuttgart, Siegen (Germany) as well as at industries at Alcatel (SEL) and IBM Zurich Research Laboratory. He is active in several EU framework projects and COST actions. Prof. Tran-Gia was coordinator of the German-wide G-Lab Project 'National Platform for Future Internet Studies' aiming to foster experimentally driven research to exploit future Internet technologies. His research activities focus on performance analysis of the following major topics: Future Internet & Smartphone Applications; QoE Modeling & Resource Management; Software Defined Networking & Cloud Networks; Network Dynamics & Control; Crowdsourcing. He has published more than 100 research papers in major conferences and journals and received the Fred W. Ellersick Prize 2013 (IEEE Communications Society).