Journal of Network and Computer Applications 136 (2019) 22–37

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Analytical Modelling of Software and Hardware Switches with Internal Buffer in Software-Defined Networks

Deepak Singh[a,*], Bryan Ng[a,**], Yuan-Cheng Lai[b], Ying-Dar Lin[c], Winston K.G. Seah[a]

[a] School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
[b] Dept of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan
[c] Dept of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

ARTICLE INFO

Keywords:
Software-Defined Networks
Modeling
Software Switch
Hardware Switch
Internal buffer
Queuing Theory

ABSTRACT

OpenFlow supports internal buffering of data packets in Software-Defined Networking (SDN) switch whereby a fraction of data packet header is sent to the controller instead of an entire data packet. This internal buffering increases the robustness and the utilization of the link between SDN switches and the controller by absorbing temporary burst of packets which may overwhelm the controller. Existing queuing models for an SDN have focused on the switches that immediately sends packets to the controller for decisioning, with no existing models investigating the impact of the internal buffer in SDN software and hardware switches. In this paper, we propose a unified queueing model to characterise the performance of SDN software and hardware switches with the internal buffer. This unified queueing model is an analytical tool for network engineers to predict a delay and loss during SDN deployments in delay and loss sensitive environments. Our results show that a hardware switch achieves up to 80% lower average packet transfer delay and 99% lower packet loss rate at the cost of requiring up to 50% more queue capacity than a software switch. The proposed models are validated with a discrete event simulation, where the error between 0.6% and 2.8% was observed for both average packet transfer delay and average packet loss rate. Moreover, a hardware switch outperforms a software switch with increasing number of hosts per switch suggesting that a hardware switch has better scalability. We use the insights from the model to develop guidelines that help network engineers decide between a software and hardware switch in their SDN deployments.

1. Introduction

Software-Defined Networking (SDN) is a new networking architecture that simplifies the switch by moving the forwarding decisions away from a switch to a centralised system which is typically realised as a software-based controller. The concept of an SDN is realised with OpenFlow which is among the first (and most widely used) specification to define the communication between the controller and switch in an SDN architecture (Goransson and Black, 2014). At present, OpenFlow is the dominant protocol for programming SDN switches (McKeown et al., 2008). OpenFlow handles different types of messages from the controller-to-switch or conversely from the switch-to-controller. The switch-to-controller messages are called *asynchronous messages* as they are sent without controller solicitation (ONF, 2013).

In the OpenFlow specifications (ONF, 2013), an OpenFlow switch maintains one or more flow tables to make decisions on packet forwarding behaviour. Flow tables are linked together to form a pipeline, where each flow table has flow table entries (FTEs) that consist of match fields and actions. Incoming data packets are matched against the match fields and if there is no matching FTE, an asynchronous message called a "packet-in" is generated and sent to the controller.

As SDN deployments move away from traditional data centers to wide area SDNs, wireless access network (called SDWAN - software defined wireless access network) and mobile SDNs, the usual assumption of a reliable and highly available control channel no longer holds. Fortunately, the OpenFlow specifications have provisions for switches to internally buffer packet-in messages destined for the controller.

* Corresponding author.
** Corresponding author.
E-mail addresses: deepak.singh@ecs.vuw.ac.nz (D. Singh), bryan.ng@ecs.vuw.ac.nz (B. Ng), laiyc@cs.ntust.edu.tw (Y.-C. Lai), ydlin@cs.nctu.edu.tw (Y.-D. Lin), winston.seah@ecs.vuw.ac.nz (W.K.G. Seah).

https://doi.org/10.1016/j.jnca.2019.03.006
Received 6 July 2018; Received in revised form 14 December 2018; Accepted 18 March 2019
Available online 22 March 2019

Packet-in messages are sent by the switch when there is no matching flow information for an arriving packet to the switch. Packet-in messages are sent either with the arriving data packet or only with a fraction of the data packet header based on the availability of memory in a switch for internal buffering. The data packet header contains routing information which is used by the controller to make forwarding decisions. If a switch has sufficient memory to buffer packets, then the packet header along with a buffer ID (identifier of the buffered packet) is sent with the packet-in message. Similarly, some switches do not support internal buffering and require full data packet (not just the header) to be sent with the packet-in message.

1.1. Internal buffer

The internal buffer in a switch plays an important role in packet forwarding behaviour. Data packets are internally buffered in an SDN switch to avoid congestion and improve the throughput of the network. The concept of internal buffering is not new to SDN switches and has been traditionally used in Banyan switches (Ye et al., 2002). These are a network of complex crossover switches designed to avoid blocking between packets at the input ports. In other areas of networking such as ATM (asynchronous transfer mode), internal buffering has been used in ATM switches to reduce the packet loss rate due to the asynchronous nature of ATM traffic (Hui, 2012).

In an SDN, the internal buffer helps in addressing the impact of lossy and unreliable control plane behaviour, a scenario of increasing importance. The study in (Osman et al., 2017) showed that a lossy control channel significantly degrades a data plane throughput and latency. Some of the benefits of internal buffering in SDN switches are: the forwarding delay of data packets can be decreased (Sun et al., 2015), Quality of Service can be improved with reduced packet loss (Appelman and de Boer, 2012), and bandwidth of the control channel can be optimized (Mao et al., 2016).

In an OpenFlow-based SDN switch, if a packet-in event is configured to internal buffering and the switch has sufficient memory to buffer a data packet, then the fraction of a data packet header and buffer ID is encapsulated with a packet-in message. Otherwise, an entire data packet is encapsulated with a packet-in message. The controller processes a packet-in message and generates a packet-out message to the switch updating flow information (ONF, 2013).

Most existing research in the literature analyses the performance of SDN switches with no internal buffering (Jarschel et al., 2011; Mahmood et al., 2014; Mahmood et al., 2015; Miao et al., 2016; Goto et al., 2016; Ogasawara and Takahashi, 2016; Shang and Wolter; Sood et al., 2016; Javed et al., 2017; Singh et al., 2017; Singh et al., 2018a; Lai et al., 2017; Fahmin et al., 2018; Azodolmolky et al., 2013; Koohanestani et al., 2017; Huang et al., 2017). This is perhaps attributed to the evolving nature of the OpenFlow specifications which in their current incarnation leaves the buffering of a data packet as an optional feature. However, it will be increasingly important for the next generation of SDN switches to support internal buffering with increasing diversification of SDN deployments. In these new diversified SDN deployments, there may be intermittent connectivity between the SDN switch and the controller during SDN deployments in domains such as SDWANs, mobile SDN and IoT.

1.2. Hardware vs. software switch

In this paper, we are concerned with the modelling of both physical SDN switches (i.e. hardware switches) and virtual switches (i.e. software switches), both with the internal buffer. Both software and hardware switches have strengths and weaknesses, and internal buffering may affect their performance in an SDN. To identify the potential bottlenecks that could hinder the performance of an SDN, the trade offs between choosing a hardware versus software switch with the internal buffer need to be studied and investigated to improve the performance of SDN.

An SDN-based software switch with the internal buffer maintains the flow table in SDRAM (synchronous dynamic random access memory) where the incoming packet is matched against the FTE using a CPU (central processing unit) (Rygielski et al., 2016). If there is no matching FTE, a data packet is internally buffered and a packet-in message is sent to the controller which feeds back forwarding information to the switch and updates the software flow table. The packet processing logic in a software switch is implemented in software (Goransson and Black, 2014) usually with the help of optimized software libraries. Open vSwitch (OVS) (Open vSwitch), Pantou/OpenWRT (Pantou), ofsoftswitch13, Indigo running on commodity hardware (e.g. desktops with several network interface cards) are few examples of SDN software switches.

Similarly, in an SDN-based hardware switch with the internal buffer, a packet processing function is embedded in the specialised hardware. This specialised hardware includes layer two forwarding tables implemented using content-addressable memories (CAMs), layer three forwarding tables using ternary content-addressable memories (TCAMs) (Goransson and Black, 2014) and application specific integrate circuits (ASICs). In a hardware switch, FTEs are stored in CAMs and TCAMs of the specialised hardware and packets are processed by ASICs. Hardware switches are also equipped with SDRAM and CPU allowing a hardware switch to maintain flow tables in both TCAM and SDRAM (Rygielski et al., 2016). Similar to software switches in an SDN, the CPU in a hardware switch internally buffers data packets when there is no matching FTE.

In this paper, we use queuing theory to derive a first order estimate of an OpenFlow switch's performance and to identify potential trade-offs between an SDN-based software and hardware switch with the internal buffer. Queueing models are useful in predicting switch performance trends as parameterized functions and link the cause to effect relationships of the switch performance. The main contributions of this paper are as follows:

- It proposes a unified queueing model to characterise the performance of hardware and software switches with the internal buffer in an SDN.
- It identifies the benefits and trade-offs of hardware switch vs. software switch with the internal buffer in an SDN.
- It validates a unified queueing model with a discrete event simulation.
- It investigates the performance of software and hardware switches for a scalable SDN with increasing number of hosts connected to the switch.

The remainder of this paper is structured as follows. Section 2 discusses the related work and background theory of SDN-based software and hardware switches, and internal buffering. Section 3 presents the queueing model for an SDN-based software switch with the internal buffer which is followed by the queuing model for an SDN-based hardware switch with the internal buffer in Section 4. Section 5 discusses buffer dimensioning. Section 6 discusses the analytical and validation results in detail. Finally, Section 7 concludes the paper with a discussion of the results and conclusion.

2. Related work & theory

While internal buffering has been well studied in a traditional switch, the buffering of asynchronous messages over a separated control-data plane remains unexplored. The separation of the data plane and control plane in SDN brings a different set of challenges for switch designers working with SDN switches. For example the control decisions from the controller may take up to 1 ms to reach the switch.

The internal buffering for software-based SDN switches can be easily realised by configuring packet-in events to support buffering of packets. However, for hardware-based SDN switches, there are very few commodity switches that support internal buffering. Pica8 switches are among the few that support OpenFlow's feature to configure temporary buffering of packets (PicOS Support), while other commodity switch manufacturers like Cisco (Cisco Plug-in), HP enterprise (HPE Switch Software), Juniper (OpenFlow Support), Arista network (Arista EOS OpenFlow), and Extreme network (ExtremeXOS) still do not support internal buffering. The reason behind fewer commodity switches supporting internal buffering is due to hardware limitations in hardware switches. This is also the reason why there is almost no experimental research conducted on SDN commodity switches to analyse internal buffering.

In (Mizuyama et al., 2017), the authors adopted an SDN for wireless mesh networks and show that the delay variability and limited bandwidth over the wireless induces throughput and packet losses. However, no internal buffering was considered. The use of internal buffering in (Mizuyama et al., 2017) could have improved the channel utilization in the SDN-enabled wireless networks for increasing control traffic. Earlier studies (Sun et al., 2015; Mao et al., 2016) suggest that the smoothing of control traffic via the internal buffer would reduce the losses during periods of poor wireless connectivity or sudden burst of new flows to a mesh router. However, these studies have not explored the drawbacks of internal buffering in an SDN.

For SDWAN applications, a multi-path OpenFlow channel for resilience and scalability in wireless environments was proposed in (Nguyen et al., 2017). In SDWANs, the control path may incur failure due to many reasons, such as deep fading, mobility, etc. In such cases, buffering packets in the internal buffer of the switch allows the switch to continue operating momentarily while the control channel recovers back to its stable state.

Hu et al. (2017) take a radically different approach whereby the control packets are neither buffered nor sent to the controller immediately but sent through a looping path - inducing delay to allow the control messages to be processed and the feedback from the controller. The internal buffering in (Hu et al., 2017) could have reduced the delay at the cost of extra memory.

From a performance modelling perspective, queueing theory has been widely used to model and predict the performance of an SDN (Jarschel et al., 2011; Mahmood et al., 2014; Mahmood et al., 2015; Miao et al., 2016; Goto et al., 2016; Ogasawara and Takahashi, 2016; Shang and Wolter; Sood et al., 2016; Javed et al., 2017; Singh et al., 2017; Singh et al., 2018a; Lai et al., 2017; Fahmin et al., 2018; Singh et al., 2018b). Most of these studies have modelled a software switch except for (Sood et al., 2016; Singh et al., 2018a) which are among the first to model a hardware switch in an SDN. Similarly, the model presented in (Singh et al., 2018b) is among the first to model an SDN switch with the internal buffer.

The above mentioned models use the generic models as shown in Fig. 1 for a software switch and Fig. 2 for a hardware switch where the input buffer of the CPU is modelled either as a single shared queue or two-priority queue. In the single shared queue model (Jarschel et al., 2011; Mahmood et al., 2014; Mahmood et al., 2015; Shang and Wolter; Sood et al., 2016; Javed et al., 2017; Lai et al., 2017; Fahmin et al., 2018), the data traffic and control traffic shares a single queue with FIFO service discipline. While in a two-priority queue model (Miao et al., 2016; Goto et al., 2016; Ogasawara and Takahashi, 2016; Singh et al., 2017, 2018a, 2018b), control traffic goes to a high priority class queue and data traffic goes to a low priority class queue where data traffic is served without preemption. The single shared queue model is not suitable for modelling internal buffers because there is no packet level distinction between data and control traffic. This differentiation is easily modelled in the two-priority queue model and thus is the most relevant starting point for our work presented in this paper.

Moreover, a key finding from previous modelling work on SDN switches shows that the use of two-priority queue in the output buffer of a switch better reflects the SDN behaviour. Analytical and simulation studies in (Singh et al., 2017) show that the time to install FTE

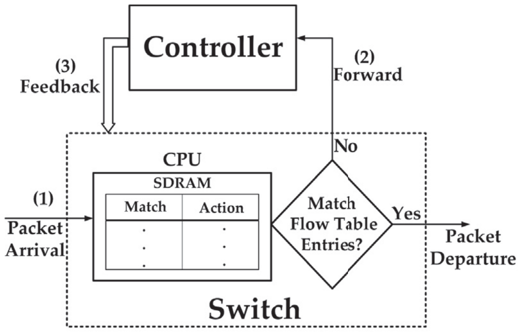


Fig. 1. Generic model for an SDN with Software Switch.

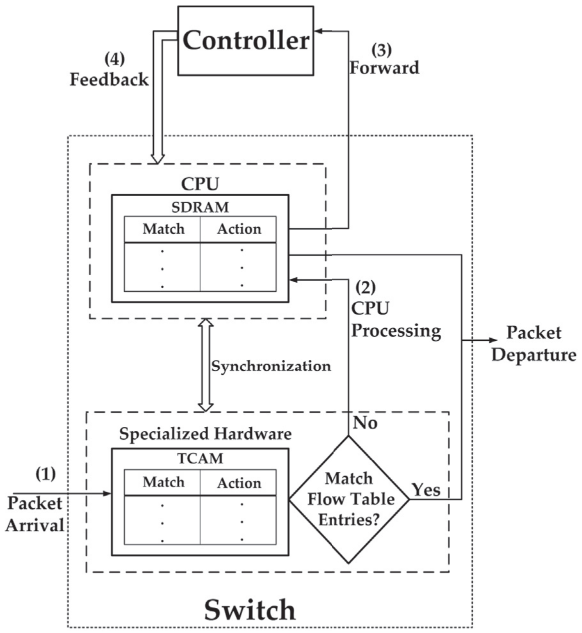


Fig. 2. Generic model for an SDN with Hardware Switch.

is significantly lower in a priority queue compared to a single shared queue.

The model presented in (Singh et al., 2018b) shows the benefit of the internal buffer which significantly reduces the delay and loss rate but at the cost of the higher memory required by the CPU for internal buffering. However, this model considers internal buffering in a software switch and cannot model the dynamics of a faster hardware switch that has dual service rates (i.e. specialised hardware service rate and CPU service rate) – thus producing estimates that are less accurate for hardware switches.

The model presented in (Sood et al., 2016) assumes that the input buffer of a switch as a single shared buffer but have not accounted the switch-controller interaction. The analysis in this work does not map the workings of a hardware switch such as the flow matching and dedicated packet processing to the queueing model as shown in Fig. 2. These limitations of (Sood et al., 2016) have been addressed in (Singh et al., 2018a) through a unified queueing model with both software and hardware switches. However, a unified queueing model in (Singh et al., 2018a) has not considered the internal buffering capabilities of an SDN switch.

The models in (Singh et al., 2018b) and (Singh et al., 2018a) have paved the way for building a new unified queueing model for internal buffering within SDN switches. A summary of existing queueing models for SDN switches with and without the internal buffer is shown

Table 1
Summary of queueing models for SDN switches with and without the internal buffer.

Model	Internal Buffering		CPU	Analysis		Switch Type	
	Yes	No	model	Exact	Approximate	Software	Hardware
Jarschel et al. (2011)		✓	M/M/1		✓	✓	
Mahmood et al. (2014)		✓	M/M/1		✓	✓	
Miao et al. (2015)		✓	M/M/1		✓	✓	
Shang and Wolter,		✓	M/H_2/1		✓	✓	
Sood et al. (2016)		✓	M/Geo/1		✓		✓
Miao et al. (2016)		✓	MMAP		✓	✓	
Goto et al. (2016)		✓	GI/M/1/K	✓		✓	
Javed et al. (2017)		✓	M/G/1		✓	✓	
Singh et al. (2017)		✓	GI/M/1/K	✓		✓	
Lai et al. (2017)		✓	MMPP/M/1	✓		✓	
Singh et al. (2018b)	✓	✓	GI/M/1/K	✓		✓	
Fahmin et al. (2018)		✓	M/M/1		✓	✓	
Singh et al. (2018a)		✓	GI/M/1/K	✓		✓	✓
Our Analysis	✓	✓	GI/M/1/K	✓		✓	✓

Table 2
Notations used for performance analysis.

Parameters	Description
λ	External arrival rate at the switch
μ_c	Service rate of the controller processor
μ_{sp}	Service rate of the CPU processor
μ_{sh}	Service rate of the hardware processor
β	Table miss probability
BER	Bit Error Rate
PER	Packet Error Ratio
n	Number of bits in the packet
N	Number of hosts connected to the switch
ρ	Server utilization at the queue
T	Throughput of the queue
t	Mean sojourn time of packets at the queue
$E[L]$	Total number of data packets in the system
PL	Average packet loss rates
$K_{\min}$	Minimum queue capacity for a switch
m_r	Controller to CPU Processing Ratio (μ_c/μ_{sp})
m_s	Specialised hardware to CPU Processing Ratio (μ_{sh}/μ_{sp})
ϵ_K	Relative minimum capacity
ϵ_d	Relative average delay
ϵ_l	Relative packet loss rate

in Table 1. Similarly, Table 2 lists the notations used for performance analysis in this paper.

In the following subsection, generic models for SDN software and hardware switches are discussed.

2.1. Packet flow in software and hardware SDN switches

A generic block diagram of an SDN-based software switch where the external data packet arrives at the switch which is connected to the controller is shown in Fig. 1. There are three important phases an SDN model of a software switch must capture. Phase (1), the first packet of a flow arrives at the switch and there is no matching FTE for the packet in SDRAM. Phase (2), the packet without a matching flow entry is forwarded to the controller **or** a packet with the matching FTE is serviced by the switch and forwarded to the destination. All packet processing and forwarding in the switch is executed on the CPU and the SDRAM. Finally, Phase (3), the controller feeds the forwarding information back to the switch and updates the flow table. Software switches have been studied and analysed in (Jarschel et al., 2011; Mahmood et al., 2014; Mahmood et al., 2015; Miao et al., 2016; Goto et al., 2016; Ogasawara and Takahashi, 2016; Shang and Wolter; Javed et al., 2017; Singh et al., 2017; Singh et al., 2018a; Lai et al., 2017; Fahmin et al., 2018; Singh et al., 2018b) based on the generic block model shown in Fig. 1.

Fig. 2 shows the block diagram of an SDN-based hardware switch where the switch maintains flow tables in both hardware and software. The hardware and software flow tables are synchronised through a middleware layer on the switch (Kuźniar et al., 2015; Pan et al., 2013) to avoid duplicate entries and to ensure consistent forwarding behaviour.

There are four important phases an SDN model of a hardware switch must capture. Phase (1), the first packet of a flow arrives at the specialised hardware in the switch that maintains hardware FTEs and there is no matching FTE for the packet. Phase (2), a packet with the matching FTE in the TCAM is serviced by the ASIC and forwarded to the destination, otherwise a packet without a matching FTE in TCAM is matched against the FTE in SDRAM and processed by the CPU for forwarding to the destination. In phase (3), a packet without any matching FTE in the TCAM or SDRAM is forwarded to the controller. In phase (4), the controller feeds the forwarding information back to the switch, updates the flow tables in both TCAM and SDRAM. Finally, the packet is serviced by the CPU and forwarded to the destination. A hardware switch has been studied and analysed in (Singh et al., 2018a) based on the generic block model shown in Fig. 2.

Based on these generic models, this paper investigates SDN software and hardware switches that support internal buffering with the help of queuing theory. Additionally, a priority queueing structure is used for the CPU that handles data and control packets, and buffer dimensioning is performed to calculate the minimum queue capacity for the switch which is discussed in the following subsection.

2.2. Buffer dimensioning

The concept of buffer dimensioning in queueing network is to determine the buffer size (K) for a given desired loss probability, hence to ensure losses due to queueing are below desired loss probability. In an SDN queuing network, it is of prime importance to provide no losses to control packets that carry the updated flow table information. The desired loss probability for the outgoing link is given in bit error rate (BER), which is 10^{-12} for 1 Gbps link according to IEEE 802.3 standard (ISO/IEC/IEEE International Standard for Ethernet, 2014). In this paper, we use this value of BER for buffer dimensioning.

For buffer dimensioning, the buffers are first assumed to be an infinite queue, and the queue is truncated at some finite integer K, such that the desired loss probability is achieved (Simcoe and Pei, 1995; Liew, 1994). The required buffer space is measured in packets.

The minimum queue capacity for a switch (denoted by $K_{\min}$) can be derived using an infinite queue model (i.e. M/M/1 queue). However, losses in queues are typically expressed as Packet Error Ratio (PER) while losses in outgoing links are expressed by BER. The relationship between BER and PER is given as:

$$PER = 1 - (1 - BER)^n, \qquad (1)$$

where n is the number of bits in the packet. In an M/M/1 queue, the probability the queue length (L) exceeds $K_{\min}$ is given by $P_r\{L > K_{\min}\} = \rho^{K_{\min}}$, where $\rho^{K_{\min}}$ is the server utilization at the queue for given $K_{\min}$. The value of $K_{\min}$ is calculated as

$$K_{\min} \geq \frac{\log[PER]}{\log[\rho^{K_{\min}}]}. \tag{2}$$

In this paper, $K_{\min}$ determines the minimum queue capacity of the switch in the queueing model.

2.3. Quasi-Birth-Death process

In queueing theory, Quasi-Birth-Death (QBD) process has been widely used to model a computer network due to the flexibility it provides to account a larger amount of details (Ost, 2013; Takagi, 1990). For this reason, the modelling approach in this paper is based on QBD processes. Hence, this subsection is devoted to describe the notation and concepts behind QBD processes.

A QBD process is a continuous-time Markov chain with multidimensional state spaces that can be partitioned into disjoint levels (Dayar et al., 2011). A continuous-time QBD process is a two-dimensional Markov chain represented as $\{(X_t, Y_t), t \geq 0\}$ with the state space $\mathbb{S} = \{(i,j) \in \{0,1,\dots,K\} \times \{0,1,\dots,L\}\}$ where i and j denotes level and phase variables of the process, respectively (Motyer, 2011). Similarly, K and L determines the queue capacities of level and phase variables respectively which can be finite or infinite.

In queueing networks, a QBD process can be multi-dimensional with one level variable and multi-dimension phase variables, whereby the phase variables denote the number of the nodes or queues in the network. For N number of queues, the state of the network can be represented by the vector $n = (n_1, n_2, \dots, n_N)$ where n_l is the number of packets in queue l. If queue 1 is the queue of interest for analysis, then packets at queue 1 are represented by the level variable and packets at queues other than queue 1 are represented by phase variables as the vector $r = (n_2, n_3, \dots, n_N)$ (Latouche and Ramaswami, 1999).

In QBD process, the transitions between the state are limited within the level or between two adjacent levels. If the transitions of QBD process are independent of the level, then such type of QBD process is homogenous or level-independent. Similarly, if the transitions are dependent of the level, then QBD process is nonhomogenous or level-dependent (Kharoufeh).

For an SDN switch with the internal buffer, the level variable tracks the number of packets in the internal buffer of the switch and the phase variable tracks the number of packets in the switch (excluding the internal buffer) and the controller. Due to the dependency of packets in the controller and the switch with temporarily buffered packets in the internal buffer, QBD process for an SDN switch with the internal buffer is non-homogeneous.

Using standard QBD notation (Neuts, 1994), the transition rate matrix of non-homogenous QBD process is given by infinitesimal generator matrix denoted as G with a repetitive block structure expressed as:

$$G = \begin{pmatrix} B_1 & A_0^{(0)} & 0 & \dots & \\ A_2^{(1)} & A_1^{(1)} & A_0^{(1)} & \ddots & \\ 0 & A_2^{(2)} & A_1^{(2)} & A_0^{(2)} & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix},$$

where $A_0^{(i)}$, $A_1^{(i)}$, and $A_2^{(i)}$ are non-negative sub-matrices for $i \geq 0$. The sub-matrices $A_0^{(i)}$, $A_1^{(i)}$, and $A_2^{(i)}$ represent phase variable distribution when the level variable increases by 1 (i.e. $i \to i+1$), remains unchanged (i.e. $i \to i$), and decreases by 1 (i.e. $i \to i-1$ for $i > 0$), respectively. Note that the use of A_0, A_1, A_2, and B_1 are standard QBD notations (Goto et al., 2016; Latouche and Ramaswami, 1999; Kharoufeh). The sub-matrices B_1 and A_1 represent phase distribution when level variable remains unchanged. The sub-matrix B_1 or $A_1^{(0)}$ represents the state of the network when the level variable is "0" (i.e. the internal buffer has no packets in its queue), while A_1 represents the network with the internal buffer having at least one packet in its queue.

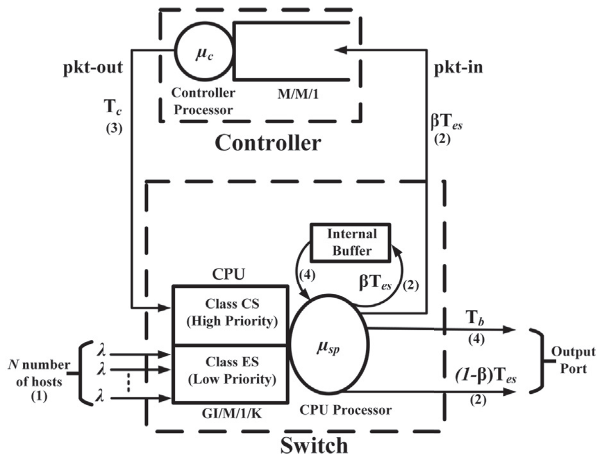


Fig. 3. SPQ–An SDN software switch with the internal buffer.

Similarly, A_0 and A_2 represent phase distribution when number of packets in the internal buffer increases or decreases by 1 respectively.

The exact distribution probabilities or the stationary state distribution (π) for QBD process is obtained by solving the system, $\pi G = 0$ and $\pi e = 0$, where e is the column vector with all elements as one. These distribution probabilities can be used to determine various performance metrics of the queueing network like average delay and throughput.

Throughout this paper, we assume the controller has an infinite capacity queue with M/M/1 distribution, and the CPU of a switch has a finite capacity GI/M/1/K queue to represent independent arrivals with general distribution (Serfozo, 2009). The total number of hosts connected to the switch is denoted as N. External packets arriving at the switch from each connected hosts are assumed to arrive according to a Poisson distribution with parameter λ. If there is no matching FTE in the switch, an external packet is temporarily buffered in the internal buffer and packet-in message is generated and sent to the controller with a probability β. The service rates of the CPU and the specialised hardware in the switch, and the controller are denoted by μ_{sp}, μ_{sh}, and μ_c, respectively. The average packet transfer delay and loss rate are primary performance metrics to compare SDN software and hardware switches with the internal buffer.

3. Software switch with the internal buffer: SPQ

We have named our queueing model for a software switch with the internal buffer as Model SPQ, where "S" refers to the switch with a software data plane, "P" refers to use of a two-priority queueing structure, and "Q" refers to queueing of data packets in the internal buffer.

As seen in Fig. 3, the switch supports internal buffering and the input buffer of the switch is modelled as a finite capacity with non-preemptive two-priority class queues, Class ES (low priority class for data packets) and Class CS (high priority class for control packets) like "SPE" in (Singh et al., 2017).

The packet processing in SPQ can be explained in four steps as shown in Fig. 3: (1) external data packets arrive at Class ES of the switch from N number of hosts connected to the switch, (2) data packets are temporarily buffered in the internal memory and a fraction of data packets are forwarded to the controller encapsulated with "packet-in" control messages if the switch does not have matching FTE **or** successfully forwarded to the destination through an output port, (3) the controller feedback the forwarding information with packet-out message to Class CS of the switch, (4) switch process the control packets in Class CS, update the flow table with forwarding information, tem-

Table 3
Permissible transitions for Model SPQ.

Event	From	To	Rate
One packet arrives to Class ES.	$(n_b, n_c, n_{cs}, n_{es})$	$(n_b, n_c, n_{cs}, n_{es}+1)$	$N\lambda$
One packet departs from Class ES to out of the system (SPQ).	$(n_b, n_c, 0, n_{es} > 0)$	$(n_b, n_c, 0, n_{es}-1)$	$\mu_{sp}(1-\beta)$
One packet departs from Class ES to the internal buffer and subsequently one packet-in message is sent to controller.	$(n_b, n_c, 0, n_{es} > 0)$	$(n_b+1, n_c+1, 0, n_{es}-1)$	$\mu_{sp}\beta$
One packet-out serviced by Controller to Class CS.	$(n_b, n_c > 0, n_{cs}, n_{es})$	$(n_b, n_c-1, n_{cs}+1, n_{es})$	μ_c
One packet in Class CS is processed and subsequently one packet departs from the internal buffer to out of the system (SPQ).	$(n_b > 0, n_c, n_{cs} > 0, n_{es})$	$(n_b-1, n_c, n_{cs}-1, n_{es})$	μ_{sp}

porarily buffered data packets are extracted from the internal buffer and forwarded to the destination through an output port.

SPQ is modelled as a continuous time Markov process with four state variables, $\{(n_b(t), n_c(t), n_{cs}(t), n_{es}(t)), t \geq 0\}$. The state variables denoted by $n_b(t)$, $n_c(t)$, $n_{cs}(t)$, and $n_{es}(t)$ represent the number of packets in the internal buffer, controller, Class CS, and Class ES respectively. Let the Markov process at time t be defined as:

$$\{n_b(t), n_c(t), n_{cs}(t), n_{es}(t)\} = \{w, x, y, z\} \tag{3}$$

where $w \in \mathbb{Z}_+^{\leq K_3}$, $x \in \mathbb{Z}_+$, $y \in \mathbb{Z}_+^{\leq K_1}$ and $z \in \mathbb{Z}_+^{\leq K_2}$. The number of packets in the controller and Class CS is dependent on the number of packets in the internal buffer. Therefore, the state space of the controller and Class CS can be rewritten as, $x \in \mathbb{Z}_+^{\leq w}$, and $y = (w - x)$ subject to $(w - x) \leq K_1$.

For example, if the number of packets in the internal buffer at some instant t is 1, i.e. $n_b(t) = 1$, then the permissible state space for controller and Class CS are $n_c(t) = \{0, 1\}$ and $n_{cs}(t) = \{1, 0\}$ respectively. Due to this dependency, Markov process in SPQ is nonhomogenous QBD process (Latouche and Ramaswami, 1999) with the internal buffer as a level variable; while the number of packets in the controller, Class CS and Class ES are phase variables. The permissible transitions for the Markov chain $\{(n_b(t), n_c(t), n_{cs}(t), n_{es}(t))\}$ are shown in Table 3 and these help us to derive sub-matrices (denoted by A_0, A_1, B_1 and A_2) of transition generator matrix (G) for SPQ. These sub-matrices are inputs to the matrix geometric solution for computing the stationary distribution probability (π) which is used to determine performance metrics for SPQ.

3.1. Elements of matrix A_0

The sub-matrix A_0 for SPQ represents the phase distribution of the controller, Class CS, and Class ES when the number of packets in the internal buffer (i.e. $n_b(t)$ or w in Eq. (3)) increases by 1:

$$A_{0(x,x')} = \begin{cases} A_{00}{}^{(x)}, & x' = x+1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{00}{}^{(x)}{}_{(y,y')} = \begin{cases} A_{001}{}^{(y)}, & y' = y = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{001}{}^{(0)}{}_{(z,z')} = \begin{cases} \mu_{sp}\beta, & z' = z-1, \\ 0, & \text{otherwise.} \end{cases}$$

3.2. Elements of matrix A_1

The sub-matrix A_1 for SPQ represents the phase distribution of the controller, Class CS, and Class ES when the number of packets in the internal buffer remain unchanged and there are some packets in the internal buffer (i.e. $n_b(t)$ or w in Eq. (3) is a positive integer that remain unchanged):

$$A_{1(x,x')} = \begin{cases} A_{11}{}^{(x)}, & x' = x, \\ A_{12}{}^{(x)}, & x' = x-1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{11}{}^{(x)}{}_{(y,y')} = \begin{cases} A_{111}{}^{(y)}, & y' = y, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$A_{12}{}^{(x)}{}_{(y,y')} = \begin{cases} A_{120}{}^{(y)}, & y' = y+1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{111}{}^{(y)}{}_{(z,z')} = \begin{cases} N\lambda, & z' = z+1, \\ \mu_{sp}(1-\beta), & y = 0, z' = z-1, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$A_{120}{}^{(y)}{}_{(z,z')} = \begin{cases} \mu_c, & z' = z, \\ 0, & \text{otherwise.} \end{cases}$$

The diagonal elements of $A_{111}{}^{(y)}{}_{(z,z')}$ where z is equal to z' has four distinct cases:

(i) When there is no packet in the controller (i.e. $n_c(t)$ or x in Eq. (3) is equal to 0),

$$A_{111}{}^{(y)}_{(z,z')} = \begin{cases} -N\lambda - \mu_{sp}, & 0 \leq z < K_2; \\ -\mu_{sp}, & z = K_2; \\ 0, & \text{otherwise,} \end{cases}$$

(ii) When the number of packets in the controller is less that the number of packets in the internal buffer i.e. $0 < x < w$ and $w < K_3$,

$$A_{111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_{sp} - \mu_c, & 0 \leq z < K_2; \\ -\mu_{sp} - \mu_c, & z = K_2; \\ 0, & \text{otherwise,} \end{cases}$$

(iii) When the number of packets in the controller is equal to that in the internal buffer which is not full i.e. $x = w$ and $w < K_3$,

$$A_{111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_c, & z = 0; \\ -N\lambda - \mu_{sp} - \mu_c, & 0 < z < K_2; \\ -\mu_{sp} - \mu_c, & z = K_2; \\ 0, & \text{otherwise,} \end{cases}$$

(iv) When the number of packets in the controller and the internal buffer are equal to the queue size of the internal buffer i.e. $x = w = K_3$,

$$A_{111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_c, & z = 0; \\ -N\lambda - \mu_{sp}(1-\beta) - \mu_c, & 0 < z < K_2, \\ -\mu_{sp}(1-\beta) - \mu_c, & z = K_2, \\ 0, & \text{otherwise,} \end{cases}$$

3.3. Elements of matrix B_1

The sub-matrix B_1 for SPQ represents the phase distribution of the controller, Class CS, and Class ES when the number of packets in the internal buffer is unchanged and there is no packet in the internal buffer (i.e. $n_b(t)$ or w in Eq. (3) is equal to 0):

$$B_{1(x,x')} = \begin{cases} B_{11}{}^{(x)}, & x' = x = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$B_{11}{}^{(0)}{}_{(y,y')} = \begin{cases} B_{111}{}^{(y)}, & y' = y = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$B_{111}{}^{(0)}{}_{(z,z')} = \begin{cases} N\lambda, & z' = z+1, \\ \mu_{sp}(1-\beta), & z' = z-1, \\ 0, & \text{otherwise.} \end{cases}$$

The diagonal elements of $B_{111}{}^{(0)}{}_{(z,z')}$ where z is equal to z' are expressed as

$$B_{111}{}^{(0)}{}_{(z,z')} = \begin{cases} -N\lambda, & z = 0, \\ -N\lambda - \mu_{sp}, & 0 < z < K_2, \\ -\mu_{sp}, & z = K_2, \\ 0, & \text{otherwise.} \end{cases}$$

3.4. Elements of matrix A_2

The sub-matrix A_2 for SPQ represents the phase distribution of the controller, Class CS and Class ES when the number of packets in the internal buffer (i.e. $n_b(t)$ or w in Eq. (3)) decreases by 1:

$$A_{2(x,x')} = \begin{cases} A_{21}{}^{(x)}, & x' = x, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{21}{}^{(x)}{}_{(y,y')} = \begin{cases} A_{212}{}^{(y)}, & y' = y-1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{212}{}^{(y)}{}_{(z,z')} = \begin{cases} \mu_{sp}, & z' = z, \\ 0, & \text{otherwise.} \end{cases}$$

3.5. Peformance metrics for SPQ

The throughputs of Class CS (T_{cs}) and the internal buffer (T_b) for SPQ are same because we have assumed a data packet in the internal buffer is extracted instantaneously after a control packet in Class CS has been processed. This assumption is reflected in the permissible transitions table for SPQ as shown in Table 3. The throughput of the internal buffer for SPQ is given by the sum of probabilities that the internal buffer has at least one data packet to forward (service rate of μ_{sp}) and this is given by:

$$T_b = T_{cs} = \mu_{sp} \sum_{w=1}^{K_3} \sum_{x=0}^{w-1} \sum_{z=0}^{K_2} \pi_{w,x,y,z} \quad . \tag{4}$$

overall Similarly, the throughput of the controller (T_c) for SPQ is given by the sum of probabilities that the controller has at least one control packet to forward (service rate of μ_c) with the condition that there is at least one data packet temporarily buffered in the internal buffer, and this is given by:

$$T_c = \mu_c \sum_{w=1}^{K_3} \sum_{x=1}^{w} \sum_{z=0}^{K_2} \pi_{w,x,y,z} \quad . \tag{5}$$

Also, the throughput of Class ES (T_{es}) for SPQ is given by the sum of probabilities that the Class ES has at least one data packet to forward (service rate of μ_{sp}) and there is no packet in Class CS in the stationary state, and this is given by:

$$T_{es} = \mu_{sp} \sum_{w=0}^{K_3} \sum_{x=0}^{w} \sum_{z=1}^{K_2} \pi_{w,x,0,z} \quad . \tag{6}$$

The average number of data packets in SPQ is $E[L]_{SPQ}$ where data packets travel only through the switch (the Class ES and the internal buffer). Therefore, $E[L]_{SPQ}$ is expressed as:

$$E[L]_{SPQ} = \sum_{w=0}^{K_3} \sum_{x=0}^{w} \sum_{z=0}^{K_2} (w+z)\pi_{w,x,y,z}. \tag{7}$$

Again, applying Little's theorem to Eq. (7) yields the average packet transfer delay in SPQ (commonly denoted by the mean sojourn time of the packet) at the switch (denoted by t_{SPQ}) which is expressed as:

$$t_{SPQ} = E[L]_{SPQ}/T_{SPQ}, \tag{8}$$

where T_{SPQ} is the throughput of SPQ expressed as:

$$T_{SPQ} = T_b + (1-\beta)T_{es}. \tag{9}$$

Similarly, the average packet loss rate of the Class CS (PL_{cs}), the Class ES (PL_{es}), and the internal buffer (PL_{ib}) represents the average number of packets being blocked or dropped by the Class CS, the Class ES, and the CPU's internal buffer out of total incoming packets. The loss rates PL_{cs}, PL_{es}, and (PL_{ib}) for Model SPQ are expressed as:

$$\begin{aligned} PL_{cs} = PL_{ib} &= 1 - T_{cs}/T_c, \\ PL_{es} &= 1 - T_{es}/N\lambda. \end{aligned} \tag{10}$$

Assuming independence between the arrival at the Class CS, the Class ES and the internal buffer, the total packet loss rate for SPQ (PL_{SPQ}) is the sum of packet loss rate in the Class CS, the Class ES and the internal buffer which is given as,

$$PL_{SPQ} = PL_{cs} + PL_{es} + PL_{ib}. \tag{11}$$

4. Hardware switch with internal buffer: HPQ

Similar to “SPQ” for a software switch with the internal buffer, we have named queueing model for a hardware switch with the internal buffer as Model HPQ, where “H” refers to a hardware data plane. HPQ is an extension of SPQ, with one additional server and a queue for the specialised hardware with M/M/1/K distribution.

As shown in Fig. 4, the switch has two servers, one for the specialised hardware (referred as hardware processor and denoted by μ_{sh}) and other one for the CPU (referred as CPU processor and denoted by μ_{sp}). Similar to SPQ, CPU is modelled as a finite capacity with non-preemptive two-priority class queues; Class HP (similar to Class ES for SPQ) as a low priority, Class CP (similar to Class CS for SPQ) as a high priority.

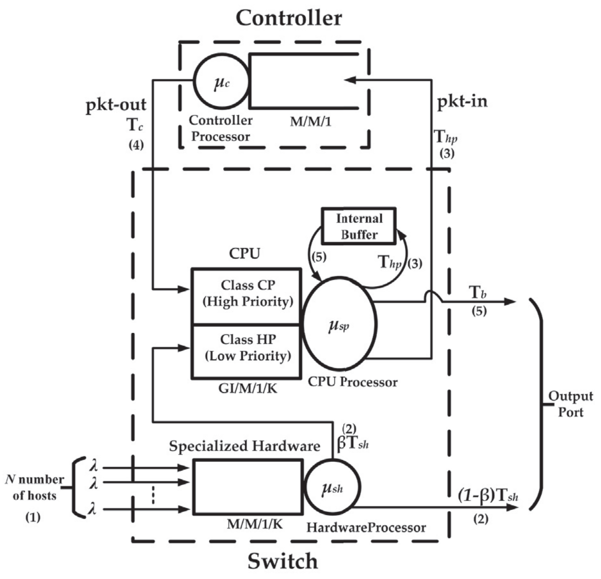


Fig. 4. HPQ–An SDN hardware switch with internal buffer.

The packet processing in HPQ can be explained in five steps as shown in Fig. 4: (1) external data packets arrive at the specialised hardware of the switch from N number of hosts connected to the switch, (2) data packets are forwarded to Class HP of the CPU if specialised hardware in the switch does not have matching FTE **or** forwarded to destination through an output port, (3) data packets are temporarily buffered in the internal memory and a fraction of data packets are forwarded to the controller encapsulated with packet-in messages, (4) the controller feedback the forwarding information with packet-out messages to Class CP of the CPU, (5) finally the CPU processes control packets in Class CP, updates and synchronises the software flow table with the flow table in the specialised hardware, extracts temporarily buffered data packets from the internal buffer and forwards them to the destination through an output port.

HPQ is modelled as a continuous time Markov process with five state variables, $\{(n_b(t), n_c(t), n_{cs}(t), n_{es}(t), n_{sh}(t)), t \geq 0\}$. The state variables denoted by $n_b(t)$, $n_c(t)$, $n_{cs}(t)$, $n_{es}(t)$, and $n_{sh}(t)$ represent the number of packets in the internal buffer, controller, Class CP, Class HP, and the specialised hardware respectively.

Similar to SPQ, queue capacities of the internal buffer, Class CP and Class HP are K_3, K_1, and K_2 respectively; and the controller is assumed to have infinite capacity. The queue capacity of the specialised hardware is K_4. Let the Markov process at time t be defined as:

$$\{n_b(t), n_c(t), n_{cs}(t), n_{es}(t), n_{sh}(t)\} = \{v, w, x, y, z\} \qquad (12)$$

where $v \in \mathbb{Z}_+^{\leq K_3}$, $w \in \mathbb{Z}_+$, $x \in \mathbb{Z}_+^{\leq K_1}$, $y \in \mathbb{Z}_+^{\leq K_2}$, and $z \in \mathbb{Z}_+^{\leq K_4}$. The number of packets in the controller and Class CP is dependent on the number of temporarily buffered packets in the internal buffer, therefore state space of the internal buffer, controller and Class CP can be rewritten as, $v \in \mathbb{Z}_+^{\leq K_3}$, $w \in \mathbb{Z}_+^{\leq w}$, and $x = (v - w)$ subject to $(v - w) \leq K_1$.

Due to the dependency of $n_c(t)$ and $n_{cs}(t)$ on the internal buffer, the process governing the number of packets in HPQ is also non-homogenous QBD process with the internal buffer as a level variable; the controller, Class CP, Class HP, and the specialised hardware as phase variables. The permissible transitions for the Markov chain $\{(n_b(t), n_c(t), n_{cs}(t), n_{es}(t), n_{sh}(t))\}$ are shown in Table 4. These transitions help us derive sub-matrices (A_0, A_1, B_1 and A_2) of the generator matrix (G) for HPQ. These sub-matrices are input to matrix geometric solution to compute the stationary distribution probability (π) which is used to determine performance metrics for HPQ.

4.1. *Elements of matrix A_0*

The sub-matrix A_0 for HPQ represents the phase distribution of the controller, Class CP, Class HP, and the specialised hardware when the number of packets in the internal buffer (i.e. $n_b(t)$ or v in Eq. (12)) increases by 1:

$$A_{0(w,w')} = \begin{cases} {A_{00}}^{(w)}, & w' = w + 1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$${A_{00}}^{(w)}_{(x,x')} = \begin{cases} {A_{001}}^{(x)}, & x' = x = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$${A_{001}}^{(0)}_{(y,y')} = \begin{cases} {A_{0012}}^{(y)}, & y' = y - 1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$${A_{0012}}^{(y)}_{(z,z')} = \begin{cases} \mu_{sp}, & z' = z, \\ 0, & \text{otherwise.} \end{cases}$$

4.2. *Elements of matrix A_1*

The sub-matrix A_1 for HPQ represents the phase distribution of the controller, Class CP, Class HP, and the specialised hardware when the number of packets in the internal buffer remain unchanged and there are some packets in the internal buffer (i.e. $n_b(t)$ or v in Eq. (12) is a positive integer that remain unchanged):

$$A_{1(w,w')} = \begin{cases} {A_{11}}^{(w)}, & w' = w, \\ {A_{12}}^{(w)}, & w' = w - 1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$${A_{11}}^{(w)}_{(x,x')} = \begin{cases} {A_{111}}^{(x)}, & x' = x, \\ 0, & \text{otherwise,} \end{cases}$$

Table 4
Permissible transitions for Model HPQ.

Event	From	To	Rate
One packet arrives to switch hardware.	$(n_b, n_c, n_{cp}, n_{hp}, n_{sh})$	$(n_b, n_c, n_{cp}, n_{hp}, n_{sh} + 1)$	$N\lambda$
One packet departs from hardware to out of the system (HPQ).	$(n_b, n_c, n_{cp}, n_{hp}, n_{sh})$	$(n_b, n_c, n_{cp}, n_{hp}, n_{sh} - 1)$	$\mu_{sh}(1 - \beta)$
One packet arrives at Class HP for CPU processing.	$(n_b, n_c, n_{cp}, n_{es}, n_{sh})$	$(n_b, n_c, n_{cp}, n_{hp} + 1, n_{sh} - 1)$	$\mu_{sh}\beta$
One packet departs from Class HP to the internal buffer and subsequently one packet-in message is sent to controller.	$(n_b, n_c, 0, n_{hp}, n_{sh})$	$(n_b + 1, n_c + 1, 0, n_{hp} - 1, n_{sh})$	μ_{sp}
One packet serviced by Controller to Class CP.	$(n_b > 0, n_c, n_{cp}, n_{hp}, n_{sh})$	$(n_b > 0, n_c - 1, n_{cp} + 1, n_{hp}, n_{sh})$	μ_c
One packet_out in Class CP is processed and subsequently one packet departs from the internal buffer to out of the system (HPQ).	$(n_b > 0, n_c, n_{cp}, n_{hp}, n_{sh})$	$(n_b - 1, n_c, n_{cp} - 1, n_{hp}, n_{sh})$	μ_{sp}

and

$$A_{12}{}^{(w)}_{(x,x')} = \begin{cases} A_{120}{}^{(x)}, & x' = x+1, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{111}{}^{(x)}{}_{(y,y')} = \begin{cases} A_{1111}{}^{(y)}, & y' = y, \\ A_{1110}{}^{(y)}, & y' = y+1, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$A_{120}{}^{(x)}{}_{(y,y')} = \begin{cases} A_{1201}{}^{(y)}, & y' = y, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$A_{1111}{}^{(y)}{}_{(z,z')} = \begin{cases} N\lambda, & z' = z+1, \\ \mu_{sh}(1-\beta), & z' = z-1, \\ 0, & \text{otherwise,} \end{cases}$$

$$A_{1110}{}^{(y)}{}_{(z,z')} = \begin{cases} \mu_{sh}\beta, & z' = z-1, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$A_{1201}{}^{(y)}{}_{(z,z')} = \begin{cases} \mu_c, & z' = z, \\ 0, & \text{otherwise.} \end{cases}$$

The diagonal elements of $A_{1111}{}^{(y)}{}_{(z,z')}$ where z is equal to z' has four distinct cases:

(i) When there is no packet in the controller (i.e. $n_c(t)$ or w in Eq. (12) is equal to 0),

$$A_{1111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_{sp}, & 0 \le y \le K_2, \\ & z = 0; \\ -N\lambda - \mu_{sh} - \mu_{sp}, & 0 \le y < K_2, \\ & 0 < z < K_4; \\ -N\lambda - \mu_{sh}(1-\beta) - \mu_{sp}, & y = K_2, \\ & 0 < z < K_4; \\ -\mu_{sh} - \mu_{sp}, & 0 \le y < K_2, \\ & z = K_4; \\ -\mu_{sh}(1-\beta) - \mu_{sp}, & y = K_2, \\ & z = K_4; \\ 0, & \text{otherwise,} \end{cases}$$

(ii) When the number of packets in the controller is less than that the number of packets in the internal buffer i.e. $0 < w < v$ and $v < K_3$,

$$A_{1111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_{sp} - \mu_c, & 0 \le y \le K_2, \\ & z = 0; \\ -N\lambda - \mu_{sp} - \mu_{sh} - \mu_c, & 0 \le y < K_2, \\ & 0 < z < K_4; \\ -\mu_{sp} - \mu_{sh} - \mu_c, & 0 \le y < K_2, \\ & z = K_4; \\ -N\lambda - \mu_{sp} & y = K_2, \\ -\mu_{sh}(1-\beta) - \mu_c, & 0 < z < K_4; \\ -\mu_{sp} - \mu_{sh}(1-\beta) & y = K_2, \\ -\mu_c, & z = K_4; \\ 0, & \text{otherwise,} \end{cases}$$

(iii) When the number of packets in the controller is equal to that in the internal buffer which is not full i.e. $w = v$ and $v < K_3$,

$$A_{1111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_c, & y = z = 0; \\ -N\lambda - \mu_{sh} - \mu_c, & y = 0, \\ & 0 < z < K_4; \\ -\mu_{sh} - \mu_c, & y = 0, \\ & z = K_4; \\ -N\lambda - \mu_{sp} - \mu_c, & 0 < y \le K_2, \\ & z = 0; \\ -N\lambda - \mu_{sp} - \mu_{sh} - \mu_c, & 0 < y \le K_2, \\ & 0 < z < K_4; \\ -\mu_{sp} - \mu_{sh} - \mu_c, & 0 < y \le K_2, \\ & z = K_4; \\ -N\lambda - \mu_{sp} - \mu_c & y = K_2, \\ -\mu_{sh}(1-\beta), & 0 < z < K_4; \\ -\mu_{sp} - \mu_{sh}(1-\beta) & y = K_2, \\ -\mu_c, & z = K_4; \\ 0, & \text{otherwise,} \end{cases}$$

(iv) When the number of packets in the controller and the internal buffer are equal to the queue size of the internal buffer i.e. $w = v = K_3$,

$$A_{1111}{}^{(y)}{}_{(z,z')} = \begin{cases} -N\lambda - \mu_c, & 0 \le y \le K_2, \\ & z = 0; \\ -N\lambda - \mu_{sh} - \mu_c, & 0 \le y < K_2, \\ & 0 < z < K_4; \\ -\mu_{sh} - \mu_c, & 0 \le y < K_2, \\ & z = K_4; \\ -N\lambda - \mu_{sh}(1-\beta) & y = K_2, \\ -\mu_c, & 0 < z < K_4; \\ -\mu_{sh}(1-\beta) - \mu_c, & y = K_2, \\ & z = K_4; \\ 0, & \text{otherwise,} \end{cases}$$

4.3. Elements of matrix B_1

The sub-matrix B_1 for HPQ represents the phase distribution of the controller, Class CP, Class HP, and the specialised hardware when the number of packets in the internal buffer remain unchanged and there is no packet in the internal buffer (i.e. $n_b(t)$ or v in Eq. (12) is equal to 0)

$$B_{1(w,w')} = \begin{cases} B_{11}{}^{(w)}, & w' = w = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$B_{11}{}^{(0)}{}_{(x,x')} = \begin{cases} B_{111}{}^{(x)}, & x' = x = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where,

$$B_{111}{}^{(0)}{}_{(y,y')} = \begin{cases} B_{1111}{}^{(x)}, & y' = y, \\ 0, & \text{otherwise.} \end{cases}$$

where,

$$B_{1111}{}^{(y)}{}_{(z,z')} = \begin{cases} N\lambda, & z' = z+1, \\ \mu_{sh}(1-\beta), & z' = z-1, \\ 0, & \text{otherwise.} \end{cases}$$

The diagonal elements of ${B_{1111}}^{(y)}{}_{(z,z')}$ where z is equal to z' is expressed as

$$
{B_{1111}}^{(y)}{}_{(z,z')} = \begin{cases}
-N\lambda, & y=0, z=0;\\
-N\lambda-\mu_{sh}, & y=0,\\
 & 0<z<K_4;\\
-\mu_{sh}, & y=0, z=K_4,\\
-N\lambda-\mu_{sp}, & 0<y\le K_2,\\
 & z=0;\\
-N\lambda-\mu_{sh}-\mu_{sp}, & 0<y<K_2,\\
 & 0<z<K_4;\\
-\mu_{sh}-\mu_{sp}, & 0<y<K_2,\\
 & z=K_4;\\
-N\lambda-\mu_{sh}(1-\beta)-\mu_{sp}, & y=K_2,\\
 & 0<z<K_4;\\
-\mu_{sh}(1-\beta)-\mu_{sp}, & y=K_2, z=K_4;\\
0, & \text{otherwise.}
\end{cases}
$$

4.4. Elements of matrix A_2

The sub-matrix A_2 for HPQ represents the phase distribution of the controller, Class CP, Class HP, and the specialised hardware when the number of packets in the internal buffer (i.e. $n_b(t)$ or v in Eq. (12)) decreases by 1:

$$
A_{2(w,w')} = \begin{cases} {A_{21}}^{(w)}, & w'=w,\\ 0, & \text{otherwise,}\end{cases}
$$

where,

$$
{A_{21}}^{(w)}{}_{(x,x')} = \begin{cases} {A_{212}}^{(x)}, & x'=x-1,\\ 0, & \text{otherwise,}\end{cases}
$$

where,

$$
{A_{212}}^{(x)}{}_{(y,y')} = \begin{cases} {A_{2121}}^{(y)}, & y'=y,\\ 0, & \text{otherwise.}\end{cases}
$$

where,

$$
{A_{2121}}^{(y)}{}_{(z,z')} = \begin{cases} \mu_{sp}, & z'=z,\\ 0, & \text{otherwise.}\end{cases}
$$

4.5. Peformance metrics for HPQ

Like SPQ, the throughputs of the Class CP (T_{cp}) and the internal buffer (T_b) for HPQ are same. The throughput of the internal buffer for HPQ is given by the sum of probabilities that the internal buffer has at least one data packet to forward with service rate of μ_{sp}, and this is given by:

$$
T_b = T_{cp} = \mu_{sp}\sum_{v=1}^{K_3}\sum_{w=0}^{v-1}\sum_{x}\sum_{y=0}^{K_2}\sum_{z=0}^{K_4}\pi_{v,w,x,y,z} \quad . \tag{13}
$$

Similarly, the throughput of the controller (T_c) for HPQ is given by the sum of probabilities that the controller has at least one control packet to forward with service rate of μ_c, and there is at least one data packet temporarily buffered in the internal buffer. This is given by:

$$
T_c = \mu_c\sum_{v=1}^{K_3}\sum_{w=1}^{v}\sum_{y=0}^{K_2}\sum_{z=0}^{K_4}\pi_{v,w,x,y,z} \quad . \tag{14}
$$

Also, the throughput of Class HP (T_{hp}) for HPQ is given by the sum of probabilities that the Class HP has at least one data packet to forward with service rate of μ_{sp} and there is no packet in Class CP in the stationary state, and this is given by:

$$
T_{hp} = \mu_{sp}\sum_{v=0}^{K_3}\sum_{w=0}^{v}\sum_{y=1}^{K_2}\sum_{z=0}^{K_4}\pi_{v,w,x,0,z} \quad . \tag{15}
$$

Finally, the throughput of the specialised hardware (T_{sh}) for HPQ is given by the sum of probabilities that the specialised hardware has at least one data packet to forward with service rate of μ_{sh} and this is given by:

$$
T_{sh} = \mu_{sh}\sum_{v=0}^{K_3}\sum_{w=0}^{v}\sum_{y=0}^{K_2}\sum_{z=1}^{K_4}\pi_{v,w,x,y,z} \quad . \tag{16}
$$

The average number of data packets in HPQ is $E[L]_{HPQ}$ where data packets travel only through the specialised hardware (i.e. TCAM) and the CPU (i.e the Class HP and the internal buffer). Therefore, $E[L]_{HPQ}$ is expressed as:

$$
E[L]_{HPQ} = \sum_{v=0}^{K_3}\sum_{w=0}^{v}\sum_{y=0}^{K_2}\sum_{z=0}^{K_4}(v+y+z)\pi_{v,w,x,y,z}. \tag{17}
$$

Again, applying Little's theorem to Eq. (17) yields the average packet transfer delay in HPQ (commonly denoted by the mean sojourn time of the packet) at the switch (denoted by t_{HPQ}) which is expressed as:

$$
t_{HPQ} = E[L]_{HPQ}/T_{HPQ}, \tag{18}
$$

where T_{HPQ} is the throughput of HPQ expressed as:

$$
T_{HPQ} = T_b + (1-\beta)T_{sh}. \tag{19}
$$

Similarly, assuming independence of packet arrivals between the Class CP, Class HP, internal buffer and the specialised hardware queue, the average packet loss rate of the Class CP (PL_{cp}), Class HP (PL_{hp}), internal buffer (PL_{ib}) and the specialised hardware queue (PL_{sh}) represents the average number of packets being blocked or dropped by the Class CS, Class ES, internal buffer and the specialised hardware queue out of total incoming packets in respective queue. The packet loss rates PL_{cp}, PL_{hp}, PL_{ib} and PL_{sh} for HPQ are expressed as,

$$
\begin{aligned}
PL_{cp} &= PL_{ib} = 1 - T_{cp}/T_c,\\
PL_{hp} &= 1 - T_{hp}/T_{sh}, \qquad (20)\\
PL_{sh} &= 1 - T_{sh}/N\lambda.
\end{aligned}
$$

Therefore, the total packet loss rate for HPQ (PL_{HPQ}) is the sum of packet loss rate in the Class CP, Class HP, internal buffer and the specialised hardware queue of the switch which is given as,

$$
PL_{HPQ} = PL_{cp} + PL_{hp} + P_{ib} + PL_{sh}. \tag{21}
$$

In the following section, we will discuss buffer dimensioning for SPQ and HPQ.

5. Buffer dimensioning for SPQ and HPQ

In this section, to perform buffer dimensioning for SPQ and HPQ, we assume that the switch queues are M/M/1 (see Section 2.2) as opposed to GI/M/1/K (used for the CPU in both SPQ and HPQ) and M/M/1/K (used for the specialised hardware in HPQ).

The minimum capacity for the switch in SPQ is denoted by $(K_{\min})_{SPQ}$ which is the sum of K_1 (i.e. minimum queue capacity required for the Class CS), K_2 (i.e. minimum queue capacity required for the Class ES), and K_3 (i.e.minimum queue capacity required for the internal buffer) which are calculated using Eq. (2) as:

$$
K_1 \ge \frac{\log[PER]}{\log[\rho_{cs}]}, K_2 \ge \frac{\log[PER]}{\log[\rho_{es}]}, K_3 \ge \frac{\log[PER]}{\log[\rho_{ib}]}, \tag{22}
$$

Table 5
Parameter used for Numerical Simulation for both SPQ and HPQ.

Parameter	Value
Table miss probability, β	0.1 ~ 1
CPU processing rate, μ_{sp} (packets/sec)	1000
Controller to CPU Processing Ratio (μ_c/μ_{sp}), m_r	0.1 ~ 2
Specialised hardware to CPU Processing Ratio (μ_{sh}/μ_{sp}), m_s	1 ~ 1000
Arrival rate, λ (packets/sec)	12, 24
Bit Error Rate, *BER*	10^{-12}
MTU TCP packet size (byte)	1500
Number of hosts per switch, *N*	1 ~ 80

where ρ_{cs}, ρ_{es}, and ρ_{ib} are the server utilization at the Class CS, Class ES, and the internal buffer, respectively, which are defined as:

$$\rho_{cs} = \frac{\beta N\lambda}{\mu_{sp}}, \qquad \rho_{es} = \frac{N\lambda}{\mu_{sp}}, \qquad \rho_{ib} = \frac{\beta N\lambda}{\mu_{sp}}.$$

Therefore, $(K_{\min})_{SPQ}$ can be expressed as

$$(K_{\min})_{SPQ} = K_1 + K_2 + K_3. \tag{23}$$

Likewise, for HPQ, the minimum queue capacities for the Class CP, Class HP, internal buffer, and the specialised hardware are denoted as K_1, K_2, K_3, and K_4, respectively, and can be calculated using Eq. (2) as:

$$K_1 \geq \frac{\log[PER]}{\log[\rho_{cp}]}, K_2 \geq \frac{\log[PER]}{\log[\rho_{hp}]}, K_3 \geq \frac{\log[PER]}{\log[\rho_{ib}]}, \quad K_4 \geq \frac{\log[PER]}{\log[\rho_{sh}]}, \tag{24}$$

where ρ_{cp}, ρ_{hp}, ρ_{ib}, and ρ_{sh} are the server utilization at the Class CP, Class HP, internal buffer of the CPU, and the specialised hardware, respectively, which are defined as:

$$\rho_{cp} = \frac{N\beta\lambda}{\mu_{sp}}, \rho_{hp} = \frac{N\beta\lambda}{\mu_{sp}}, \rho_{ib} = \frac{N\beta\lambda}{\mu_{sp}}, \rho_{sh} = \frac{N\lambda}{\mu_{sh}}.$$

Therefore, the minimum queue capacity for the switch in HPQ is the sum of minimum queue capacity for the Class CP, Class HP, internal buffer, and the specialised hardware:

$$(K_{\min})_{HPQ} = K_1 + K_2 + K_3 + K_4. \tag{25}$$

In this paper, the minimum queue capacity of the switch for SPQ and HPQ are $(K_{\min})_{SPQ}$ and $(K_{\min})_{HPQ}$, respectively.

6. Results

This section presents the analytical and discrete event simulation results of the unified queueing model for SDN software and hardware switches with the internal buffer (i.e. SPQ and HPQ respectively). This section is divided into the following subsections:

- *Validation*:

– *Validation of analytical models* where analytical results are compared with discrete event simulation results.

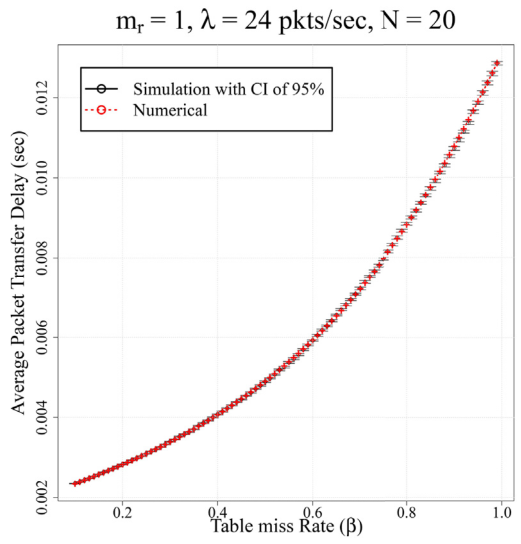
- *Performance Characterisation*:

– *Relative minimum capacity* where the total minimum queue capacity for SPQ and HPQ is compared.
– *Relative average delay* where average packet transfer delay of SPQ and HPQ is compared.
– *Relative packet loss rate* where packet loss rate of SPQ and HPQ is compared.
– *Effect of varying number of hosts connected to the switch* is investigated and compared between SPQ and HPQ.
– *Effect of varying μ_{sh} in a hardware switch* where the effect of varying hardware processing capacity (i.e. μ_{sh}) in HPQ is investigated.

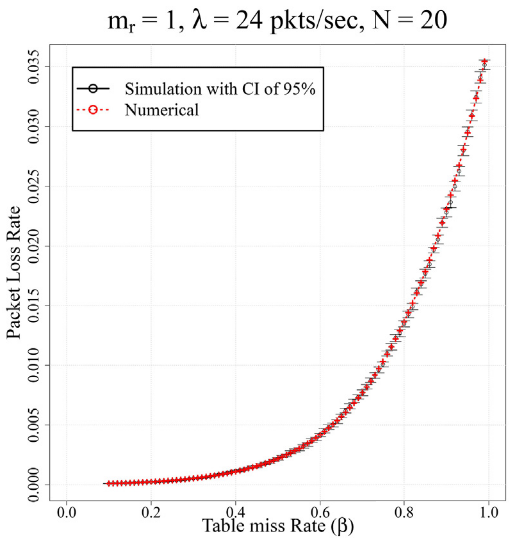
The parameters used for analysis and simulation is shown in Table 5. From Table 5, the table miss probability β varies from 0.1 to 1, the switch processor or CPU processing rate (μ_{sp}) is assumed to be 1000 packets/sec, the controller to switch processing ratio (m_r) varies from 0.1 to 2, and the specialised hardware to CPU processing ratio (m_s) varies from 1 to 1000. The external arrival rate (λ) at the switch from each host is assumed to be 24 or 48 packets/sec and we assume an Ethernet network for which the *BER* is assumed to be 10^{-12}. We use TCP as the transport protocol with maximum transmission unit (MTU) of 1500 bytes. Thus, the PER is 1.2×10^{-8} (using Eq. (1)). The number of hosts per switch (N) is varied from 1 to 80.

The simulations are repeated hundred times and the 95% confidence intervals (CI) are computed on the basis that the errors are normally distributed.

In the following subsections, to take the packet loss rate into consideration, we assume queue capacities of the Class ES (in SPQ), the Class



(a)



(b)

Fig. 5. Validation of SPQ for (a) average packet transfer delay; (b) packet loss rate.

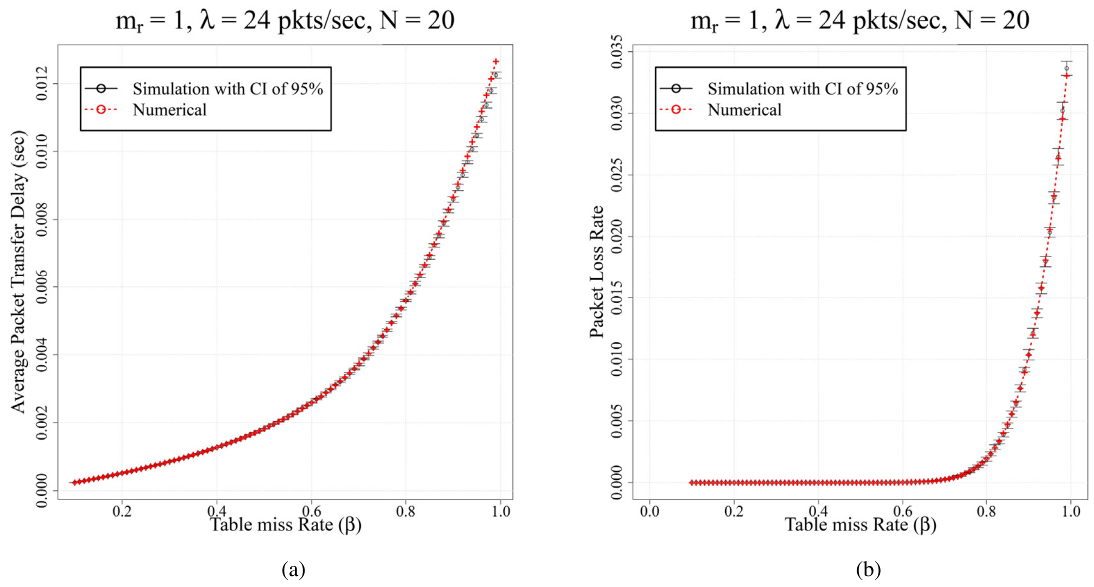


Fig. 6. Validation of HPQ for (a) average packet transfer delay; (b) packet loss rate.

HP (in HPQ) and the specialised hardware queue (in HPQ) to be half of their minimum queue capacities determined from buffer dimensioning (using Eq. (23) and Eq. (25)). The queue capacities of the Class CS (in SPQ), the Class CP (in HPQ) and the internal buffer (in both SPQ and HPQ) are minimum queue capacities determined from buffer dimensioning where there is no packet loss. This buffer sizing ensures no loss of control packets.

6.1. Validation of analytical models

The validation of analytical results for SPQ and HPQ is done by comparing them with discrete event simulation results. Figs. 5 and 6 show the validation results for SPQ and HPQ respectively for increasing β with $m_r = 1$ and $m_s = 1000$. The error percentage between analysis and simulation predictions for both average packet transfer delay and packet loss rate is between 0.6% and 2.8% as shown in Figs. 5 and 6. This range of error is acceptable for analysis as computation of π distributions for nonhomogenous QBD process is prone to inaccuracy due to the possibility of singular matrix becoming nonsingular in machine precision (Dayar et al., 2011).

6.2. Relative minimum capacity

In this subsection, we compute the relative minimum queue capacity between SPQ and HPQ denoted as ϵ_K which is defined as,

$$\epsilon_K = \frac{(K_{\min})_{\text{SPQ}} - (K_{\min})_{\text{HPQ}}}{(K_{\min})_{\text{SPQ}}} \times 100\%.$$

A positive value of ϵ_K means HPQ requires less capacity than SPQ, while a negative value implies HPQ requiring more capacity than SPQ.

Fig. 7 shows the ϵ_K curve for increasing β. From Fig. 7, we can observe that HPQ requires upto 50% more buffer capacity than SPQ. This is because the switch in HPQ requires queue capacities for the CPU, the specialised hardware, and the internal buffer. While, the switch in SPQ requires queue capacities for the CPU and its internal buffer only.

6.3. Relative average delay

We compare the average packet transfer delay between SPQ (denoted by t_{SPQ} as in Eq. (8)) and HPQ (denoted by t_{HPQ} as in Eq. (18)). This comparison helps to investigate the effect of the internal buffer in a software and hardware switch with reference to average packet transfer delay.

The relative average packet transfer delay (denoted by ϵ_d) between SPQ and HPQ (both with finite capacity) is calculated as:

$$\epsilon_d = \frac{(t_{SPQ} - t_{HPQ})}{t_{SPQ}} \times 100\%.$$

A positive value of ϵ_d means HPQ has lower average delay for packet to travel in the network compared to SPQ.

Fig. 8 shows the relative average packet transfer delay between SPQ and HPQ in percentile. Fig. 8(a) and (b) show the relative average delay for increasing β and m_r, respectively with $m_s = 1000$. From Fig. 8(a), we can observe that HPQ exhibits up to 80% reduction in average delay

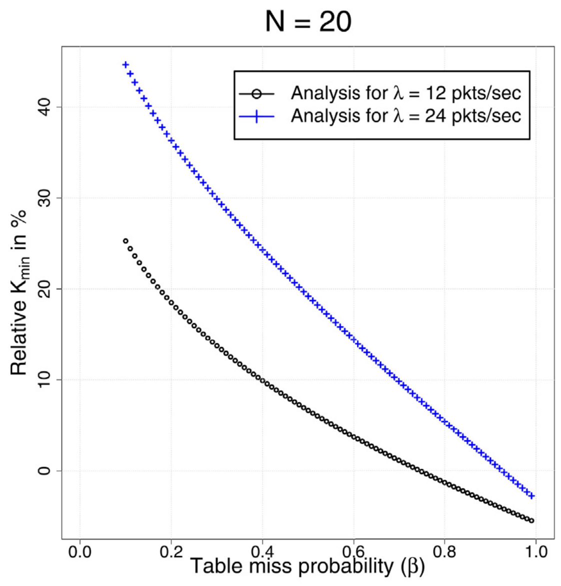
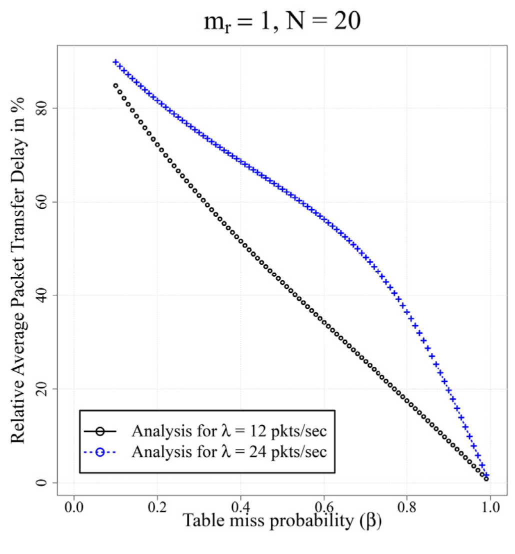
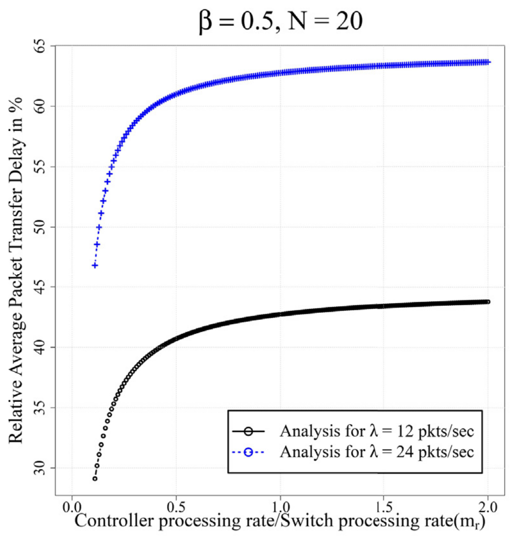


Fig. 7. Relative $K_{\min}$ between SPQ and HPQ in % i.e. ϵ_K for increasing β

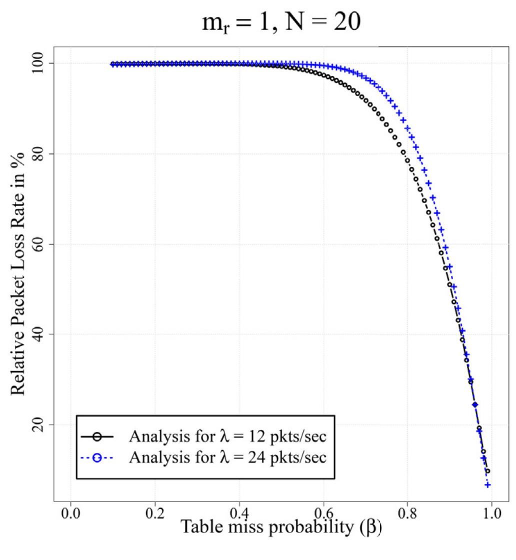


(a)

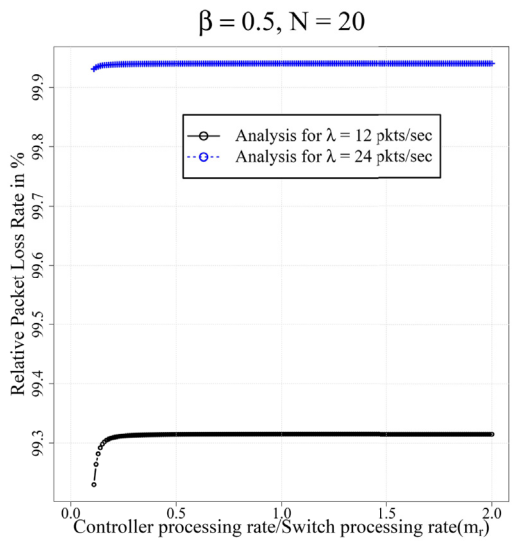


(b)

Fig. 8. Relative average delay between SPQ and HPQ in % i.e. ϵ_d for increasing: (a) $m_r = 1$; (b) $\beta = 0.5$.



(a)



(b)

Fig. 9. Relative average packet loss rate between SPQ and HPQ in % i.e. ϵ_l for increasing: (a) $m_r = 1$; (b) $\beta = 0.5$.

of the packet compared to SPQ for increasing β. Similarly, Fig. 8(b) shows the relative average packet transfer delay between SPQ and HPQ for increasing m_r, where HPQ exhibits up to 60% reduction in average delay of the packet.

This is because the specialised hardware of the switch processes external packets arriving at the switch much faster than the CPU which reduces the overall average delay of the packet. However, this reduction in average delay diminishes with the increasing number of packets being forwarded to the CPU with increasing β as seen in Fig. 8(a). Similarly, with the increasing controller processing capacity, the average delay of packet reduces. The relative reduction in average packet transfer delay reaches saturation when m_r is greater than 1 as seen in Fig. 8(b).

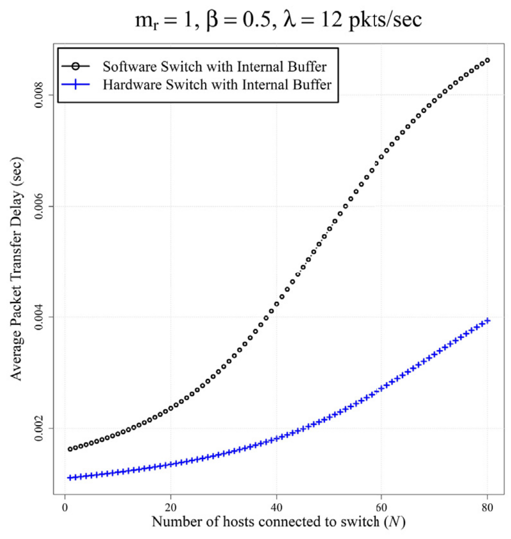
This shows the benefit of a hardware switch with the internal buffer over a software switch with the internal buffer, that significantly reduces the overall average delay of the packet for lower β and higher m_r.

6.4. Relative packet loss rate

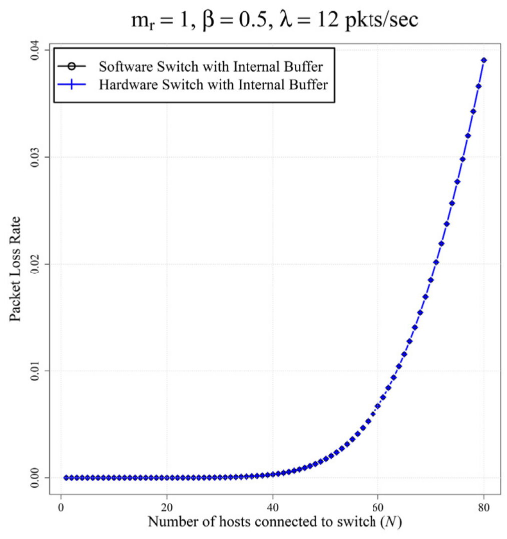
We compared the average packet loss rate between SPQ (denoted by PL_{SPQ} as in Eq. (11)) and HPQ (denoted by PL_{HPQ} as in Eq. (21)). This comparison helped us to investigate the effect of the internal buffer in a software and hardware switch with reference to the average packet loss rate.

The relative average packet loss rate (denoted by ϵ_l) between SPQ and HPQ (both with finite capacity) is calculated as:

$$\epsilon_l = \frac{(PL_{SPQ} - PL_{HPQ})}{PL_{SPQ}} \times 100\%.$$

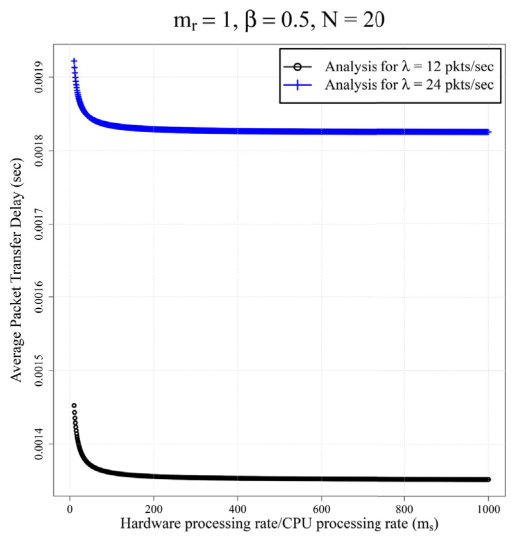


(a)

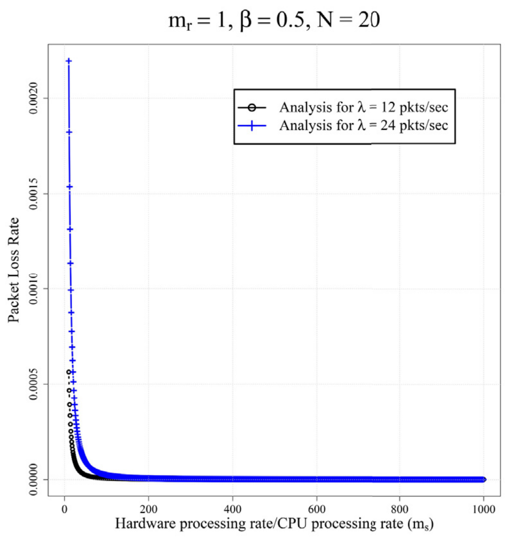


(b)

Fig. 10. Effect of varying number of hosts for $m_r = 1$ and $\beta = 0.5$.



(a)



(b)

Fig. 11. Effect of varying μ_{sh} in hardware switch for $m_r = 1$ and $\beta = 0.5$.

A positive value of ϵ_l means HPQ has lower packet loss rate compared to SPQ.

Fig. 9 shows the relative average packet loss rate between SPQ and HPQ in percentile. Fig. 9(a) and (b) show the relative average packet loss rate for increasing β and m_r, respectively with $m_s = 1000$. From Fig. 9(a) and (b), HPQ exhibits up to 100% reduction in average packet loss rate compared to SPQ increasing β and m_r, respectively.

This reduction in average packet loss rate is because average waiting time of packets in the specialised hardware queue of the switch is less than the CPU. Due to the lower waiting time, the packet loss rate in specialised hardware queue is also lower than the CPU.

This shows the benefit of a hardware switch with the internal buffer over a software switch with the internal buffer, that significantly reduces the packet loss rate.

6.5. Effect of varying number of hosts connected to the switch

In this subsection, the effect of varying number of hosts for both SPQ and HPQ is presented by varying N from 1 to 80. Fig. 10 shows the effect of varying number of hosts for $m_r = 1$ and $\beta = 0.5$. Fig. 10(a) and (b) show the effect of varying number of hosts on average packet transfer delay and packet loss rate respectively. From Fig. 10(a), with the increase in number of hosts, HPQ exhibits much lower average packet transfer delay than SPQ. Similarly, from Fig. 10(b), the packet loss rate for both SPQ and HPQ is identical and increases with the increase in the number of switches.

This increase in the packet loss rate is because with the increase in number of hosts, the net arrival of packets at both SPQ and HPQ increases exponentially. The specialised hardware of HPQ processes these incoming packets at line rate that results into relatively lower average delay than SPQ which has slower processing via the CPU.

6.6. Effect of varying μ_{sh} in a hardware switch

In this subsection, the effect of varying μ_{sh} in a hardware switch with the internal buffer is presented. This is done by varying m_s (i.e. ratio of specialised hardware to CPU processing) from 1 to 1000.

Fig. 11 shows the results for varying μ_{sh} in HPQ with $m_r = 1$ and $\beta = 0.5$. Fig. 11(a) and (b) show the effect of varying μ_{sh} in HPQ for average packet transfer delay and packet loss rate, respectively. From Fig. 11(a) and (b), both average packet transfer delay and packet loss rate becomes steady for m_s greater than 100.

From this investigation, the processing capacity of specialised hardware should be atleast 100 times of the CPU to have optimum reduction in packet transfer delay and almost zero packet loss rate.

7. Conclusion

In this study, we have proposed a unified queueing model for software and hardware switches with the internal buffer. Internal buffering in SDN-based software and hardware switches has not been investigated much, especially from the analytical modelling aspect. Therefore, a unified queueing model is a useful tool for network analysts to get quick insights into SDN-based software and hardware switches with the internal buffer.

The impact of the internal buffer in both software and hardware switches is investigated and the summary of our analysis is as follows:

- A hardware switch significantly reduces the average packet transfer delay (almost by 80%) than a software switch.
- A hardware switch requires additional buffer (almost 50% more) than a software switch, which is the tradeoff for the gains mentioned in the previous point – this insight is not provided by any of the existing models in the literature.
- A hardware switch significantly reduces the packet loss rate (almost by 99%) compared to a software switch.
- For an increasing number of hosts connected to the switch, a hardware switch exhibits significantly lower delay compared to a software switch.

Lastly, the model also suggests that the processing power of the switch and the controller are intrinsically tied. Our results show that no improvements in packet loss occurs after the specialised hardware to CPU processing ratio (m_s) exceeds 0.2.

Acknowledgement

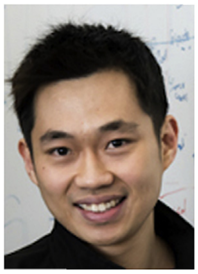
This research is partly supported by Victoria's Huawei NZ Research Programme, Software-Defined Green Internet of Things project (E2881) and a Victoria Doctoral Scholarship.

References

Appelman, M., de Boer, M., 2012. Performance Analysis of OpenFlow Hardware. University of Amsterdam, pp. 2011–2012. Tech. Rep.
Arista EOS OpenFlow, https://www.arista.com/en/um-eos/eos-openflow.
Azodolmolky, S., Nejabati, R., Pazouki, M., Wieder, P., Yahyapour, R., Simeonidou, D., 2013. An analytical model for software defined networking: a network calculus-based approach. In: Proceedings of the IEEE Conference on Global Communications Conference (GLOBECOM). IEEE, pp. 1397–1402.
Cisco Plug-in for OpenFlow Configuration, https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3850/software/release/37e/b-openflow-37e-3850-and-3650/b-openflow-37e-3850_chapter_01.html.
Dayar, T., Sandmann, W., Spieler, D., Wolf, V., 2011. Infinite level-dependent QBD processes and matrix-analytic solutions for stochastic chemical kinetics. Adv. Appl. Probab. 43 (4), 1005–1026.
ExtremeXOS OpenFlow User Guide, https://documentation.extremenetworks.com/openflow/exos_all/openflow/openflow.shtml.
Fahmin, A., Lai, Y.-C., Hossain, M.S., Lin, Y.-D., 2018. Performance modeling and comparison of NFV integrated with SDN: under or aside? J. Netw. Comput. Appl. 113, 119–129.
Goransson, P., Black, C., 2014. Software Defined Networks: A Comprehensive Approach. Elsevier.
Goto, Y., Masuyama, H., Ng, B., Seah, W.K.G., Takahashi, Y., 2016. Queueing analysis of software defined network with realistic openflow–based switch model. In: Proceedings of the IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), London, UK.
HPE Switch Software OpenFlow v1.3 Administrator Guide, http://h22208.www2.hpe.com/eginfolib/networking/docs/switches/K-KA-KB/16-01/5200-0146OFAG/webhelp/content/index.html.
Hu, C., Hou, K., Li, H., Wang, R., Zheng, P., Zhang, P., Wang, H., 2017. SoftRing: taming the reactive model for software defined networks. In: Network Protocols (ICNP), 2017 IEEE 25th International Conference on. IEEE, pp. 1–10.
Huang, J., Xu, L., Duan, Q., Xing, C.-c., Luo, J., Yu, S., 2017. Modeling and performance analysis for multimedia data flows scheduling in software defined networks. J. Netw. Comput. Appl. 83, 89–100.
Hui, J.Y., 2012. Switching and Traffic Theory for Integrated Broadband Networks, vol. 91. Springer Science & Business Media.
Indigo: Open Source Openflow Switches. URL https://floodlight.atlassian.net/wiki/spaces/HOME/overview.
ISO/IEC/IEEE International Standard for Ethernet, 2014. ISO/IEC/IEEE 8802-3:2014(E), pp. 1–3754.
Jarschel, M., Oechsner, S., Schlosser, D., Pries, R., Goll, S., Tran-Gia, P., 2011. Modeling and performance evaluation of an OpenFlow architecture. In: Proceedings of the 23rd International Teletraffic Congress. International Teletraffic Congress, pp. 1–7.
Javed, U., Iqbal, A., Saleh, S., Haider, S.A., Ilyas, M.U., 2017. A stochastic model for transit latency in OpenFlow SDNs. Comput. Network. 113, 218–229.
J. P. Kharoufeh, Level-Dependent Quasi-Birth-And-Death Processes, Wiley Encyclopedia of Operations Research and Management Science.
Koohanestani, A.K., Osgouei, A.G., Saidi, H., Fanian, A., 2017. An analytical model for delay bound of OpenFlow based SDN using network calculus. J. Netw. Comput. Appl. 96, 31–38.
Kuźniar, M., Perešíni, P., Kostić, D., 2015. What you need to know about sdn flow tables. In: International Conference on Passive and Active Network Measurement. Springer, pp. 347–359.
Lai, Y.-C., Ali, A., Hassan, M.M., Hossain, M.S., Lin, Y.-D., 2017. Performance modeling and analysis of TCP connections over software defined networks. In: GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, pp. 1–6.
Latouche, G., Ramaswami, V., 1999. Introduction to Matrix Analytic Methods in Stochastic Modeling, vol. 5. Siam.
Liew, S.C., 1994. Performance of various input-buffered and output-buffered ATM switch design principles under bursty traffic: simulation study. IEEE Trans. Commun. 42 (234), 1371–1379.
Mahmood, K., Chilwan, A., Østerbø, O.N., Jarschel, M., 2014. On the Modeling of OpenFlow-Based SDNs: the Single Node Case, pp. 207–214.
Mahmood, K., Chilwan, A., Østerbø, O.N., Jarschel, M., 2015. Modelling of OpenFlow-based software-defined networks: the multiple node case. Netw., IET 4 (5), 278–284.
Mao, J., Han, B., Sun, Z., Lu, X., Zhang, Z., 2016. Efficient mismatched packet buffer management with packet order-preserving for OpenFlow networks. Comput. Network. 110, 91–103.
McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J., 2008. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38 (2), 69–74.
Miao, W., Min, G., Wu, Y., Wang, H., 2015. Performance modelling of preemption-based packet scheduling for data plane in software defined networks. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). IEEE, pp. 60–65.
Miao, W., Min, G., Wu, Y., Wang, H., Hu, J., 2016. Performance modelling and analysis of software-defined networking under bursty multimedia traffic. ACM Trans. Multimed Comput. Commun. Appl 12 (5s), 77.
Mizuyama, K., Taenaka, Y., Tsukamoto, K., 2017. Estimation based adaptable flow aggregation method for reducing control traffic on software defined wireless networks. In: Proceedings of the IEEE Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE, pp. 363–368.
Motyer, A.J., 2011. Quasi-birth-and-death Processes with an Infinite Phase Space. University of Melbourne, Department of Mathematics and Statistics.
Neuts, M.F., 1994. Matrix-geometric Solutions in Stochastic Models - an Algorithmic Approach. Dover Publications.
Nguyen, K., Ishizu, K., Kojima, F., 2017. An evolvable, scalable, and resilient control channel for software defined wireless access networks. Comput. Electr. Eng. 57, 104–117.
ofsoftswitch13. URL https://github.com/CPqD/ofsoftswitch13.
Ogasawara, S., Takahashi, Y., 2016. Performance analysis of traffic classification in an OpenFlow switch. In: Proceedings of the 2nd IEEE Conference on Cloudification of the Internet of Things (CIoT). IEEE, pp. 1–6.
ONF, October 2013. OpenFlow Switch Specification. Tech. rep.. Open Networking Foundation.
Open vSwitch. URL http://openvswitch.org/.
OpenFlow Support on Juniper Network Devices, https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/junos-sdn-openflow-supported-platforms.html.

Osman, M., Núñez Martínez, J., Mangues-Bafalluy, J., 2017. Hybrid SDN: evaluation of the impact of an unreliable control channel. In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). IEEE, pp. 242–246.
Ost, A., 2013. Performance of Communication Systems: A Model-Based Approach with Matrix-Geometric Methods. Springer Science & Business Media.
Pan, H., Guan, H., Liu, J., Ding, W., Lin, C., Xie, G., 2013. The FlowAdapter: enable flexible multi-table processing on legacy hardware. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. ACM, pp. 85–90.
Pantou: OpenFlow 1.3 for OpenWRT. URL https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-for-OpenWRT.
PicOS Support for OpenFlow 1.3, https://docs.pica8.com/display/PICOS2111cg/PicOS + Support + for + OpenFlow + 1.3.
Rygielski, P., Seliuchenko, M., Kounev, S., Klymash, M., 2016. Performance analysis of SDN switches with hardware and software flow tables. In: Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools 2016).
Serfozo, R., 2009. Basics of Applied Stochastic Processes. Springer Science & Business Media.
Z. Shang, K. Wolter, Delay Evaluation of OpenFlow Network Based on Queueing Model, arXiv preprint arXiv:1608.06491.
Simcoe, R.J., Pei, T.-B., 1995. Perspectives on ATM switch architecture and the influence of traffic pattern assumptions on switch design. SIGCOMM Comput. Commun. Rev. 25 (2), 93–105.
Singh, D., Ng, B., Lai, Y.-C., Lin, Y.-D., Seah, W.K., 2017. Modelling software-defined networking: switch design with finite buffer and priority queueing. In: Proceeding of the 42nd IEEE Conference on Local Computer Networks (LCN). IEEE, pp. 567–570.
Singh, D., Ng, B., Lai, Y.-C., Lin, Y.-D., Seah, W.K., 2018a. Modelling software-defined networking: software and hardware switches. J. Netw. Comput. Appl. 122, 24–36.
Singh, D., Ng, B., Lai, Y.-C., Lin, Y.-D., Seah, W.K., 2018b. Modelling switches with internal buffering in software-defined networks. In: 2018 27th International Conference on Computer Communication and Networks (ICCCN). IEEE, pp. 1–9.
Sood, K., Yu, S., Xiang, Y., 2016. Performance analysis of software-defined network switch using *MGeo*1 model. IEEE Commun. Lett. 20 (12), 2522–2525.
Sun, X.S., Agarwal, A., Ng, T.E., 2015. Controlling race conditions in openflow to accelerate application verification and packet forwarding. IEEE Trans. Netw. Serv. Manag. 12 (2), 263–277.
Takagi, H., 1990. Stochastic Analysis of Computer and Communication Systems. Elsevier Science Inc.
Ye, T.T., Micheli, G.D., Benini, L., 2002. Analysis of power consumption on switch fabrics in network routers. In: Proceedings of the 39th Annual Design Automation Conference. ACM, pp. 524–529.

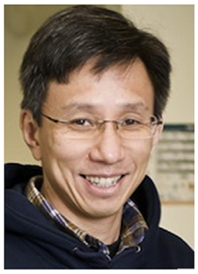
Deepak Kumar Singh received the B.Eng. degree in Electronics and Communication engineering from Kathmandu Engineering College (Affiliated to Tribhuvan University), Kathmandu, Nepal in 2010, the M.Eng. degree in Electronics and Radio engineering from Kyung Hee University, Gyeonggi-do, South Korea, in 2014. He is currently working towards the Ph.D. degree at Victoria University of Wellington, New Zealand. His research focuses on modelling of Software-Defined Network.



Bryan Ng completed his PhD (2010) in the area of communication and networking. He held teaching & research positions in Malaysia and France in addition to attachments to commercial research laboratories Intel, Motorola, Panasonic and Orange Labs. His research interest include performance analysis of communication networks, modelling networking protocols and software defined networking.

Yaun-Cheng Lai received his Ph.D. degree in the Department of Computer and Information Science from National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology in August 2001 and has been a professor since February 2008. His research interests include performance analysis, protocol design, wireless networks, and and network security.

Ying-Dar Lin is a Distinguished Professor of computer science at National Chiao Tung University (NCTU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose, California, during 2007–2008, and the CEO at Telecom Technology Center, Taipei, Taiwan, during 2010–2011. Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic and has been an approved test lab of the Open Networking Foundation (ONF) since July 2014. He also cofounded L7 Networks Inc. in 2002, which was later acquired by D-Link Corp. His research interests include network security, wireless communications, and network cloudification. He is an IEEE Fellow (class of 2013), IEEE Distinguished Lecturer (2014–2017), and ONF Research Associate, and is the Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST). He published a textbook, Computer Networks: An Open Source Approach (McGraw-Hill, 2011).



Winston K.G. Seah received the Dr.Eng. degree from Kyoto University, Kyoto, Japan, in 1997. He is currently Professor of Network Engineering in the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. Prior to this, he has worked for more than 16 years in mission-oriented industrial research, taking ideas from theory to prototypes, most recently, as a Senior Scientist in the Institute for Infocomm Research, Singapore. His latest research interests include Internet of Things, wireless sensor networks powered by ambient energy harvesting, wireless multi-hop networks, software defined networking, and 5G access protocols for machine-type communications