Computer Networks 167 (2020) 107033

Contents lists available at ScienceDirect

Computer Networks

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

Full encapsulation or internal buffering in OpenFlow based hardware switches?

Deepak Singh[a,*], Bryan Ng[a,1], 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

Article history:
Received 1 July 2019
Revised 26 September 2019
Accepted 26 November 2019
Available online 26 November 2019

ABSTRACT

Software-Defined Networking (SDN) enables programmability in the network through a software-dependent control function. OpenFlow is a de-facto protocol for communication between an SDN switch and the controller. OpenFlow specifications generally allow two methods for packet encapsulation of data packets at the switch that require decisions from the controller, (a) full encapsulation, and (b) internal buffering. However, full encapsulation of data packets has been the default choice for packet processing, and internal buffering has not been explored much, especially for hardware switching. In this paper, we model and analyse the effect of internal buffering on the performance of an OpenFlow hardware switch. We compare queueing models for the switch with full encapsulation and internal buffering for hardware switching. The results show that internal buffering significantly reduces the average packet transfer delay by 85% and packet loss probability by 60%. The results further provide guidelines to network operators that internal buffering for hardware switching is useful for controllers with lower processing rates, especially in delay-sensitive applications running over SDN. For loss-sensitive applications running over SDN, the full encapsulation method is more robust handling flows with a high table miss probability.

1. Introduction

Software-Defined Networking (SDN) provides flexible packet processing to higher layer applications via protocols such as OpenFlow [1] and ForCES [2]. OpenFlow is the most widely used protocol to define communication between the controller and switch in SDN paradigm [3]. In SDN, the control function resides in a logically centralised controller that has network-wide visibility, while the packet forwarding is executed by the switch [3]. This “separation of concerns” between the control function and packet forwarding allows automation and programmability in the network through software.

The network automation and programmability via SDN has driven network engineers and researchers to extend SDN into wireless networks such as Industrial Internet of Things (IIoT) [4], fifth-generation (5G) wireless networks [5], and Software defined Wireless Access Networks (SDWAN) [6]. These wireless networks are relatively less stable than their wired equivalents, operate less efficiently with increasing network size (e.g. due to the contention in CSMA/CA type wireless access) and may sustain significant packet bursts over the switch-controller path leading to packet loss and poor throughput [7,8]. Fortunately, OpenFlow switches have provisions to address the problem of contention and memory issues at the low-end switch via internal buffering [9].

Internal buffering in communication networks is used as temporary buffering of packets within the switch [10]. An OpenFlow switch with internal buffering temporarily buffers data packets within the switch and send asynchronous messages designated as “Packet-In” to the controller. A Packet-In message consists of a fraction of a data packet (around 20%) encapsulated with the OpenFlow header and buffer identifier associated with a data packet [11–13].

OpenFlow switches forward packets based on the entries of flow tables (an entry can be thought of as a row with multiple columns). A flow table is made up of flow table entries (FTEs) which generally consist of match fields with associated actions (the columns in a row). If there is no matching FTE for incoming packets of a flow at the switch, the first packet of that flow is passed to the SDN controller for decisioning. For the rest of this paper, we assume that OpenFlow is the protocol for communication between

* 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).
[1] Principal corresponding author.

https://doi.org/10.1016/j.comnet.2019.107033
1389-1286/© 2019 Elsevier B.V. All rights reserved.

the controller and switch in SDN [14] and we only consider hardware switches (see [15,16] for hardware vs. software switches).

OpenFlow specifications provide two packet encapsulation methods. These packet encapsulation methods are (a) internal buffering of data packets, and (b) full encapsulation of data packets. Internal buffering of data packets allows only part of data packets to be encapsulated within Packet-In messages. However, this requires additional memory at the OpenFlow switch to temporarily buffer data packets that can result in resource contention in the controller. Similarly, full encapsulation (FE) of data packets allows an OpenFlow switch (typically lower end switches) with no support for internal buffering to encapsulate an entire data packet with a Packet-In message. Therefore, the Packet-in message in the internal buffering method is smaller in size compared to the full encapsulation method. Clearly, these packet encapsulation methods have benefits and trade-offs which will be identified and analysed in this paper.

The packet encapsulation methods for the OpenFlow switch are modeled and compared analytically using queueing theory in this paper. Queueing models help switch designers and network analysers to predict performance measures before actual deployment, making it cost-effective and faster for network planning. The primary objective of this paper is to identify benefits and tradeoffs between full encapsulation of data packets and internal buffering of data packets at the OpenFlow switch.

The rest of paper is organised into six sections. In Section 2, we discuss the existing related works that use an OpenFlow based packet encapsulation methods. The hardware switching of an OpenFlow based switch is discussed in the Section 3 using the generic model. An OpenFlow based packet encapsulation methods are described and modelled using queueing theory in Section 4. The analytical and simulation results with performance analysis are discussed in Section 5. The values for traffic and switch parameters used in Section 5 are justified in Section 6 through sensitivity analyses. Finally, this paper is concluded with the summary of results in Section 7.

2. Related work

Buffering has been extensively used in the traditional switches such as ATM and Banyan switches to avoid delay and loss of packets under heavy traffic conditions [10]. Under heavy traffic conditions, the logically centralised controller in SDN can be a bottleneck with increasing flow requests for decisions. Therefore, internal buffering in SDN switches reduce the communication workload on the OpenFlow channel between the controller and switch as studied in [11].

The work in [11] focuses on the buffer management scheme and channel utilisation using their software and hardware prototypes. These prototypes are suggestive improvement toward an existing OpenFlow buffer management scheme but have paved the path for research in internal buffering in SDN.

There is little research exploring benefits and drawbacks of internal buffering in SDN. This may be due to the current lack of support for internal buffering in commodity switches [13]. The analytical modelling approach in this paper is a cheaper and faster approach to predict the performance of SDN without actually procuring equipment and setting up a test-bed.

From a performance analysis perspective of SDN via analytical modelling, queueing theory has been seen a resurgence for modelling SDN since the publication of the M/M/1 model for a single controller–switch network in [17]. Queueing models predict network performance and provide insights to network designers and analysers for their SDN deployments [16]. Queueing theory has been used in [17–21] to model an OpenFlow switch. These are few selected models that have significant contribution to modelling and performance analysis of an OpenFlow switch in SDN. However, these models have assumed the full encapsulation of data packets requiring the decisioning from the SDN controller, primarily because it is the default option for packet processing in the OpenFlow specifications [12].

The work presented in [12] is the first to model an OpenFlow switch with internal buffering. In this work, the full encapsulation of data packets is compared with the internal buffering. The results from this work showed that the packet encapsulation with internal buffering significantly reduces: (i) the average delay of packets traversing the network and (ii) packet loss probability compared to the full encapsulation method, but these benefits were obtained at the cost of requiring additional memory to support internal buffering. The work in [12] considers a software switch and the model cannot account for: (i) the line speed processing of a hardware switch and (ii) potential interactions between the high speed switch processor and the other processors in the network.

The work in [16] is among the first to investigate the effect of hardware switching on the OpenFlow switch. In this work, a software-based OpenFlow switch is compared with a hardware-based OpenFlow switch with the full encapsulation of data packets. Their results showed that a hardware switching with the full encapsulation method significantly reduces the overall delay and packet loss probability in the SDN.

The work in [13] compares a software-based OpenFlow switch with a hardware-based OpenFlow switch with the internal buffering of data packets. The results show that a hardware-based OpenFlow switch provides better scalability with increasing number of hosts per switch than a software-based OpenFlow switch.

However, the comparative analysis between the internal buffering and full encapsulation methods in hardware-based OpenFlow switches has not been studied. The comparative analysis in this paper helps network engineers to understand the hardware switching effects of these methods in an OpenFlow switch. In this paper, the full encapsulation of data packets is compared with internal buffering of data packets for a hardware switch in an OpenFlow switch.

3. Generic model for an OpenFlow based hardware switch

The generic model for a hardware-based SDN switch equipped with the specialised hardware and CPU (central processing unit) is shown in Fig. 1. The hardware switch maintains flow tables in both TCAM (ternary content-addressable memory) and SDRAM (synchronous dynamic random access memory) which are synchronised through a middleware layer on the switch. This synchronisation ensures consistent forwarding behaviour and avoids duplicate flow table entries (FTEs) [22,23].

Fig. 1 shows four phases that a hardware-based SDN switch must capture. The four phases are:

- Phase (1): The flow arrives at the specialised hardware in the switch.
- Phase (2): The first packet of a flow is matched against FTEs stored in TCAM. If there is no matching FTE for a packet in TCAM, a packet is forwarded to the CPU to match against FTEs stored in SDRAM, otherwise serviced to the destination.
- Phase (3): A packet without matching FTE is processed by the CPU using packet encapsulation methods and then forwarded to the controller.
- Phase (4): The controller feedback with flow updates in both SDRAM and TCAM. After the flow updates, the packet is serviced to the destination by the CPU.

Based on these four important phases, queueing models for packet encapsulation methods in a hardware-based SDN switch are developed in the following section. The queueing models are

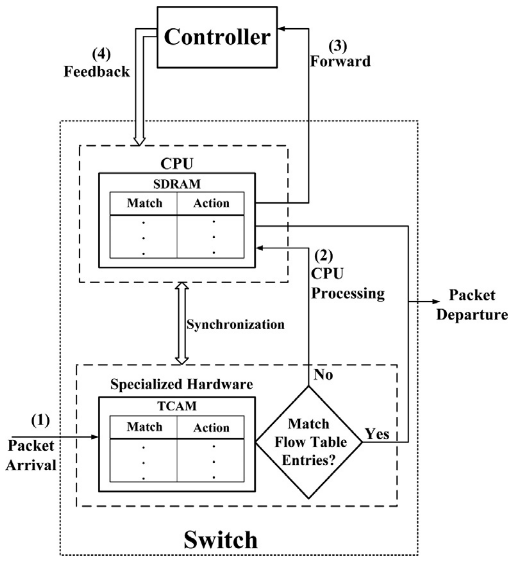


Fig. 1. Generic model for a hardware-based SDN switch with the specialised hardware and the CPU [13,16].

further described with details of network occurrences and performance metrics such as delay and loss probability.

4. Queueing models for packet encapsulation methods

For brevity, queueing models for hardware switching with the packet encapsulation method that require an internal buffering of data packets is named as "Internal Buffering" and full encapsulation of data packets is named as "Full Encapsulation". Hardware switching involves the CPU and specialised hardware (namely, ternary content-addressable memory or TCAM) [24].

It is assumed that the CPU uses priority queues with a finite capacity with GI/M/1/K distribution and the specialised hardware is represented as M/M/1/K distribution [16,20,25]. The priority queues provide isolation between the packets arriving from the controller and packet to be processed by the CPU when there is no matching FTE in the specialised hardware.

The priority queues were first introduced in [20,25] to distinguish packets processed by the CPU into two classes, Class HP and Class CP. The Class HP represents the low priority class of the CPU for the external data packet that has no matching entry in the switch's specialised hardware. The data packet is sent from the specialised hardware to the Class HP with probability β. The Class CP represents the high priority class of the CPU to store packets feedback from the controller. It is assumed that the CPU synchronises the table information (in the CPU) with specialised hardware with negligible delay. Both Class CP and Class HP queues share service rate μ_{sp} while the specialised hardware queue is serviced at the rate μ_{sh} and $\mu_{sh} \gg \mu_{sp}$.

Similarly, the controller is represented as an M/M/1 queue with service rate μ_c. For the remaining parameters, λ_1 is the external arrival rate at the TCAM and T represents the throughput of the queue. It is further assumed that the queue buffer capacities of the Class CP, Class HP, specialised hardware, and internal buffer are K_1, K_2, K_3, and K_4 respectively.

For comparative analysis between the full encapsulation method and internal buffering, the primary performance metrics are the average delay and packet loss probability. These performance measures have a huge impact on the overall user experience. The average delay determines the speed of the network while the packet loss probability is a negative indicator that causes a breakup in the network's communication.

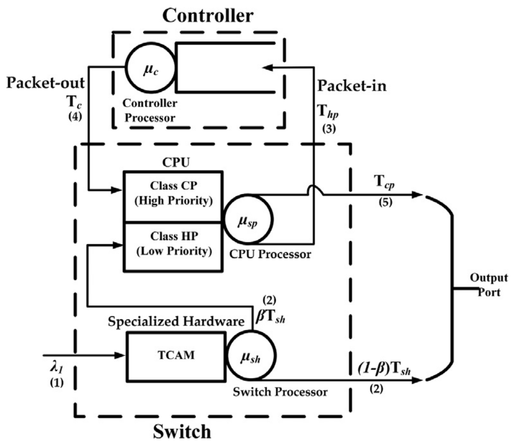


Fig. 2. SDN hardware switching using *full encapsulation* method.

4.1. Full encapsulation

The packet processing for the full encapsulation queueing model can be explained in five steps as shown in Fig. 2 [16]: (1) external data packets arrive to the specialised hardware of the switch, (2) data packets are forwarded to the Class HP of the CPU if the specialised hardware in the switch has no matching FTE **or** the packet leaves the switch (i.e. is forwarded to the destination via an output port), (3) data packets are encapsulated with Packet-In control message and forwarded to the controller, (4) controller feeds the forwarding information back to switch via a "packet-out" message to the CPU (queue up in Class CP), and (5) finally the CPU processes control packets in the Class CP, updates and synchronises flow tables with the specialised hardware, and forwards data packets to the destination through an output port.

The full encapsulation model is modelled as a continuous time Markov process with four state variables, $\{(n_c(t), n_{cp}(t), n_{hp}(t), n_{sh}(t)), t \geq 0\}$. Each of the four state variables symbolised by $n_c(t)$, $n_{cp}(t)$, $n_{hp}(t)$, and $n_{sh}(t)$ tracks the packet count in the controller, Class CP, Class HP and specialised hardware. Moreover, each state variable is a positive integer bounded by ∞, K_1, K_2, and K_3 respectively (see definition in Section 4). Thus, the Markov process for the full encapsulation model at time t be defined as

$$\{n_c(t), n_{cp}(t), n_{hp}(t), n_{sh}(t)\} = \{w, x, y, z\}. \tag{1}$$

The state transitions of Markov process for the full encapsulation model are shown in Table 1. The state transitions are used to compute the stationary distribution probability (π) and is a well documented matrix-analytic approach [26–28] for the problem we are solving (thus not described further in this paper). Using π, the performance metrics like throughput, average delay and packet loss probability can be computed for the full encapsulation method.

Average number of data packets for full encapsulation model (denoted by L_{FE}) is the average number of data packets in the CPU and specialised hardware of the switch for full encapsulation model. In the full encapsulation model, data packets travel through the specialised hardware (i.e TCAM), the CPU (i.e the Class HP and Class CP), and the controller. Therefore, L_{FE} is given as

$$L_{FE} = \sum_{w=0}^{\infty} \sum_{x=0}^{K_1} \sum_{y=0}^{K_2} \sum_{z=0}^{K_3} (w + x + y + z)\pi_{w,x,y,z}. \tag{2}$$

Average data packet transfer delay for full encapsulation model (denoted by t_{FE}) is the mean sojourn time of a data packet in an SDN network with a single controller and hardware switch with the full encapsulation packet processing method. Applying Little's formula

Table 1
Mapping network occurrences to Markov process transitions for Full Encapsulation.

Network occurrence	From	To	Rate
An external data packet enters switch TCAM.	(w, x, y, z)	$(w, x, y, z+1)$	λ_1
Switch processor services one packet from TCAM.	(w, x, y, z)	$(w, x, y, z-1)$	$\mu_{sh}(1-\beta)$
One packet forwarded to CPU (Class HP) from TCAM.	(w, x, y, z)	$(w, x, y+1, z-1)$	$\mu_{sh}\beta$
Packet-In message is sent to controller.	$(w, 0, y, z)$	$(w+1, 0, y-1, z)$	μ_{sp}
One packet serviced by Controller forwarded to CPU (Class CP).	(w, x, y, z)	$(w-1, x+1, y, z)$	μ_c
A single packet serviced by switch CPU.	(w, x, y, z)	$(w, x-1, y, z)$	μ_{sp}

on Eq. (2), the average time a packet spends in the system is given as

$$t_{FE} = \frac{L_{FE}}{T_{FE}}, \tag{3}$$

where T_{FE} is the total throughput of the switch using the full encapsulation method and is given as

$$T_{FE} = T_{cp} + (1-\beta)T_{sh}. \tag{4}$$

Packet loss probability (PL_{FE}) is the packet loss probability due to blocking of packets at the Class CP, Class HP and specialised hardware queue for the full encapsulation model. The packet loss probability in the full encapsulation model is expressed as

$$PL_{FE} = 1 - T_{FE}/\lambda_1. \tag{5}$$

4.2. Internal buffering

The packet processing for the internal buffering model is same as the full encapsulation model with an additional internal buffer as shown in Fig. 3. If there is no matching FTE for a flow at the TCAM, data packets are temporarily buffered in an internal memory and a portion of the data packet is encapsulated with a Packet-In message which is then forwarded to the controller. Similarly, when the controller feedbacks a Packet-Out message with control packet, CPU instantaneously processes the control packet in Class CP, updates and synchronises the flow table with the TCAM, extracts temporarily buffered data packet from the internal buffer and forwards it to the destination through an output port.

The internal buffering in an OpenFlow switch 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 description and range of these five state variables are shown in Table 2.

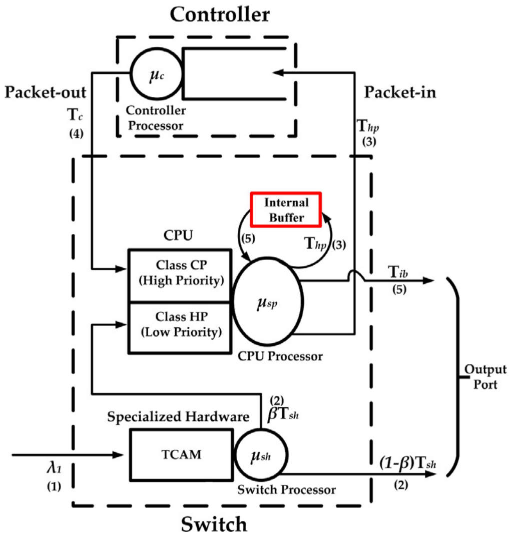


Fig. 3. SDN hardware switching using *internal buffering* method.

Table 2
State variables for Internal Buffering.

State	Description	Range
$n_b(t)$	Number of packets in the internal buffer	$[0, K_4]$
$n_c(t)$	Number of packets in the controller	$[0, \infty]$
$n_{cs}(t)$	Number of packets in the Class CP	$[0, K_1]$
$n_{es}(t)$	Number of packets in the Class HP	$[0, K_2]$
$n_{sh}(t)$	Number of packets in the specialised hardware	$[0, K_3]$

Let the Markov process for the internal buffering model at time t be defined as

$$\{n_b(t), n_c(t), n_{cs}(t), n_{es}(t), n_{sh}(t)\} = \{v, w, x, y, z\}. \tag{6}$$

The number of packets in the controller and Class CP are dependent on the number of temporarily buffered packets in the internal buffer hence $x = (v - w)$ and the state variables $\{v,\ w,\ x,\ y,\ z\}$ are positive integers bounded by $\{K_4,\ \infty,\ K_1,\ K_2,\ K_3\}$ respectively.

The mapping of network occurrences to transitions for the internal buffering model are shown in Table 3 and these help us compute performance metrics for the internal buffering model through stationary distribution probability π.

Throughput of Internal Buffer (denoted by T_{ib}) is the sum of probabilities that the internal buffer at the CPU has at least one data packet to forward with service rate of μ_{sp} for the internal buffering model. The throughputs of the internal buffer (T_{ib}) and Class CP (T_{cp}) for the internal buffering model are the same as the processing of the packet in Class CP by the CPU is instantly followed by the extraction of the packet from the internal buffer [12]. This is also reflected in the last row of Table 3 where the processing of the Class CP packet and the extraction of the internally buffered packet by the CPU occurs at the same time. Therefore, T_{ib} is given as

$$T_{ib} = T_{cp} = \mu_{sp} \sum_{v=1}^{K_4} \sum_{w=0}^{v-1} \sum_{y=0}^{K_2} \sum_{z=0}^{K_3} \pi_{v,w,x,y,z}. \tag{7}$$

Average number of data packets for the internal buffering model (denoted by L_{IB}) is the average number of data packets in the CPU and specialised hardware of the switch for internal buffering model. In the internal buffering model, data packets travel only through the specialised hardware (i.e TCAM) and the CPU (i.e the Class HP and the internal buffer). Therefore, L_{IB} is given as

$$L_{IB} = \sum_{v=0}^{K_4} \sum_{w=0}^{v} \sum_{y=0}^{K_2} \sum_{z=0}^{K_3} (v + y + z)\pi_{v,w,x,y,z}. \tag{8}$$

Average data packet transfer delay for the internal buffering model (denoted by t_{IB}) is the mean sojourn time of a data packet in an OpenFlow-based SDN with a single controller and hardware switch with the support for an internal buffer. The average packet transfer delay of a packet in the internal buffering model is obtained by applying Little's theorem to Eq. (8) which is expressed as

$$t_{IB} = L_{IB}/T_{IB}, \tag{9}$$

Table 3
Mapping network occurrences to Markov process transitions for Internal Buffering.

Network occurrence	From	To	Rate
An external data packet enters switch TCAM.	(v, w, x, y, z)	$(v, w, x, y, z+1)$	λ_1
Switch processor services one packet from TCAM.	(v, w, x, y, z)	$(v, w, x, y, z-1)$	$\mu_{sh}(1-\beta)$
One packet forwarded to CPU (Class HP) from TCAM.	(v, w, x, y, z)	$(v, w, x, y+1, z-1)$	$\mu_{sh}\beta$
One packet forwarded to the internal buffer from Class HP.	$(v, w, 0, y, z)$	$(v+1, w, 0, y-1, z)$	μ_{sp}
Packet-In message is sent to controller.	$(v, w, 0, y, z)$	$(v, w+1, 0, y-1, z)$	μ_{sp}
One packet forwarded to switch (Class CP) from Controller.	$(v > 0, w, x, y, z)$	$(v > 0, w-1, x+1, y, z)$	μ_c
One packet leaves the system from internal buffer triggered by packet-out in Class CP.	$(v > 0, w, x, y, z)$	$(v-1, w, x-1, y, z)$	μ_{sp}

where T_{IB} is the throughput of the switch using the internal buffering method, expressed as

$$T_{IB} = T_{ib} + (1-\beta)T_{sh}. \tag{10}$$

Packet loss probability (denoted by PL_{IB}) is the packet loss probability due to blocking of packets at the Class CP, Class HP, internal buffer and specialised hardware queue of the switch. The packet loss probability for the internal buffering model (PL_{IB}) is given as

$$PL_{IB} = 1 - T_{IB}/\lambda_1. \tag{11}$$

4.3. Buffer dimensioning

Buffer dimensioning ensures packet losses due to queueing are below losses that occur in the outgoing links. The losses in the outgoing links are given by the Bit Error Rate (BER). The BER can be approximately expressed as a function of the Packet Error Ratio (PER) with the relation: $PER = 1-(1-BER)^N$ where N is the number of bits in the packet. Following the example of [29,30], we conduct a very simple approximation of the minimum buffer size to ensure no queueing losses by assuming all queues are M/M/1 and each queue operates independent of all other queues.

Let K_{FE} denote the minimum buffer capacity to ensure no queueing loses for the full encapsulation model. The minimum buffer capacity for K_{FE} is simply the sum of the minimum buffer capacities K_1, K_2 and K_3:

$$\begin{aligned} K_{FE} &= K_1 + K_2 + K_3, \\ &= \frac{\log PER}{\log \rho_{cp}} + \frac{\log PER}{\log \rho_{hp}} + \frac{\log PER}{\log \rho_{sh}}, \end{aligned} \tag{12}$$

where ρ_{cp}, ρ_{hp}, and ρ_{sh} are the server utilisation at the Class CP, Class HP, and specialised hardware, respectively, which are defined as

$$\rho_{cp} = \frac{\beta\lambda_1}{\mu_{sp}}, \qquad \rho_{hp} = \frac{\beta\lambda_1}{\mu_{sp}}, \qquad \rho_{sh} = \frac{\lambda_1}{\mu_{sh}}.$$

The minimum buffer capacity for the internal buffer is denoted as K_4 which is calculated as

$$K_4 = \left\lceil \frac{\log PER}{\log \rho_{ib}} \right\rceil, \tag{13}$$

where ρ_{ib} is the server utilisation at the internal buffer of the CPU which is defined as

$$\rho_{ib} = \frac{\beta\lambda_1}{\mu_{sp}}.$$

Hence the minimum buffer capacity for the internal buffer model (say K_{IB}) is $K_4 + K_{FE}$.

5. Results

The parameters used for analysis and simulation are shown in Table 4. The table miss probability β has a geometrical relationship with a flow size [20] and is varied from 0.1 to 1, the switch processor or CPU service rate (μ_{sp}) is assumed to be 1000 packets/sec,

Table 4
Parameter used for both Full encapsulation and Internal buffering models.

Parameter	Value
Table miss probability, β	0.1 ~ 1
CPU service rate, μ_{sp} (packets/sec)	1000
TCAM service rate, μ_{sh} (packets/sec)	$1000 \times \mu_{sp}$
Controller to CPU service rate Ratio (μ_c/μ_{sp}), m_r	0.1 ~ 2
External data packet arrival rate, λ_1 (packets/sec)	120, 240, 480
Bit Error Rate, BER	10^{-12}
MTU TCP packet size (byte)	1500

the controller to switch processing ratio (m_r) is varied from 0.1 to 2, and the specialised hardware service rate (μ_{sh} – in packets/sec) is assumed to be 1000 times faster than the CPU. The external arrival rate of data packets (λ_1) to the switch from each host is fixed at 120, 240 and 480 packets/sec to investigate the effect of increasing traffic volumes. These values of β, μ_{sp}, m_r, μ_{sh}, and λ_1 are arbitrary values used for sensitivity analysis to identify the trends.

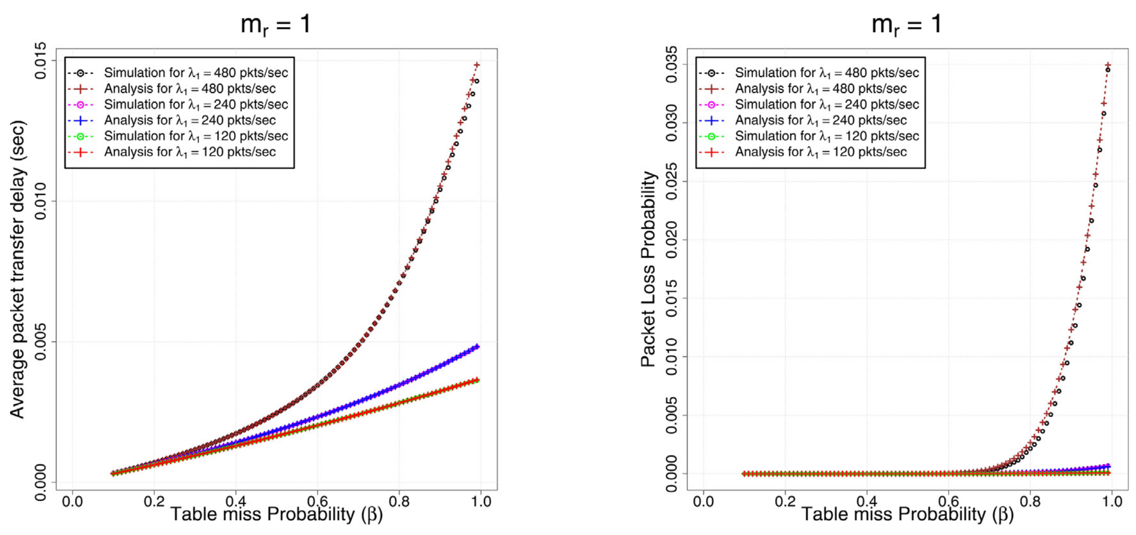
The value of the number of bits (i.e. N) for the MTU TCP packet size of 1500 bytes is 12,000 bits. Similarly, the switch and controller are assumed to be deployed over an Ethernet network for which the BER is 10^{-12} [31] and the value of PER is calculated as 1.2×10^{-8} as discussed in Section 4.3.

In the following subsections, queue capacities of the Class HP and specialised hardware queue is assumed to be half of their minimum queue capacities determined from buffer dimensioning, using Eqs. (12) and (13), i.e. $\lceil \frac{K_2}{2} \rceil$ and $\lceil \frac{K_3}{2} \rceil$ respectively. This is done to take into consideration the packet loss in the switch. Similarly, buffer dimensioning is done for the Class CP (K_1) and internal buffer (K_4) to ensure no loss of control packets.

Based on our assumptions, the value of K_3 is calculated as 2 for λ_1 equal to 120, 240, and 480 packets/sec. Similarly, the values of K_1, K_2, and K_4 are same and calculated as 5 for λ_1 equal to 120 packets/sec, 7 for λ_1 equal to 240 packets/sec, and 13 for λ_1 equal to 480 packets/sec.

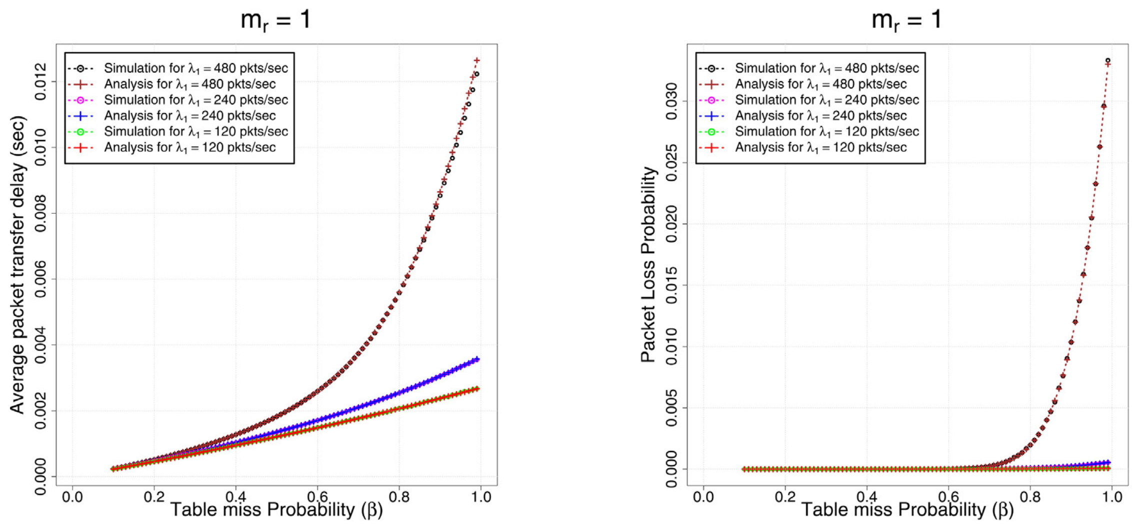
The discrete-event simulation in this manuscript is based on Monte Carlo simulation where Monte Carlo simulators use random number generators to simulate the system. The events for the Monte Carlo simulation are related to the transition rates of queueing models which cause models to change their states. Both the queueing model and discrete-event simulation use the same assumptions. The simulations are conducted with 100 random seeds and the results are reported with 95% confidence intervals (CI).

To compute the steady-state probabilities using the matrix-analytic method, a MATLAB toolbox [32] with standard library functions to solve QBD Markov chains has been used. As discussed earlier, the full encapsulation is represented as a level-independent QBD and the internal buffering as a level-dependent QBD. Therefore, in this paper, Logarithmic Reduction algorithm [33] has been used for the full encapsulation to compute a rate matrix due to a faster quadratic convergence rate. Similarly, the Matrix Continued Fraction (MCF) algorithm [34] has been used for the internal buffering which is faster and efficient to handle iteration while computing rate matrices for each level. The generation of block



(a) (b)

Fig. 4. Validation of Full encapsulation model by comparing: (a) Average packet transfer delay, and (b) Packet loss probability.



(a) (b)

Fig. 5. Validation of Internal buffering model in terms of (a) Average packet transfer delay, and (b) Packet loss probability.

matrices which are elements of rate matrices for the full encapsulation and internal buffering are briefly discussed in [16] and [13], respectively.

5.1. Validation: full encapsulation and internal buffering

The validation of analytical results for the full encapsulation model and the internal buffering model is done by comparing it with discrete event simulation results. Figs. 4 and 5 show the validation results for the full encapsulation and internal buffering models respectively for increasing β with $m_r = 1$. The error percentage between analysis and simulation predictions for both average packet transfer delay and packet loss probability is between 0.6%-2.8% as shown in Figs. 4 and 5. The error for the internal buffering model is slightly higher than the full encapsulation model by 1%. This range of error is acceptable for the internal buffering model due to level dependency of packets in the controller and Class CP on the packet temporarily buffered at the internal buffer [35].

5.2. Comparing computation time

In this subsection, the computation time between analysis and discrete-event simulation for the full encapsulation and internal buffering is compared. The computation time for λ_1 equal to 120, 240, and 480 packets/sec is computed. This comparison helps to observe the computation overload between analysis and simulation for varying external arrival rate in a hardware switch. The values of the queue buffer capacities for λ_1 equal to 120, 240, and 480 packets/sec have been set in the Section 5 along with other parameters.

Fig. 6 shows the average computation time in milli second to calculate the average packet transfer delay and packet loss probability for varying λ_1. Fig. 6(a) shows the computation overload for the full encapsulation and Fig. 6(b) shows the computation overload for the internal buffering. From Fig. 6(a) and (b), it can be observed that the computation overload for analysis drastically increases for 480 packets/sec compared to discrete-event simulation.

This is because the queue buffer capacities for λ_1 equal to 480 packets/sec is high compared to 120 and 240 packets/sec. Due to

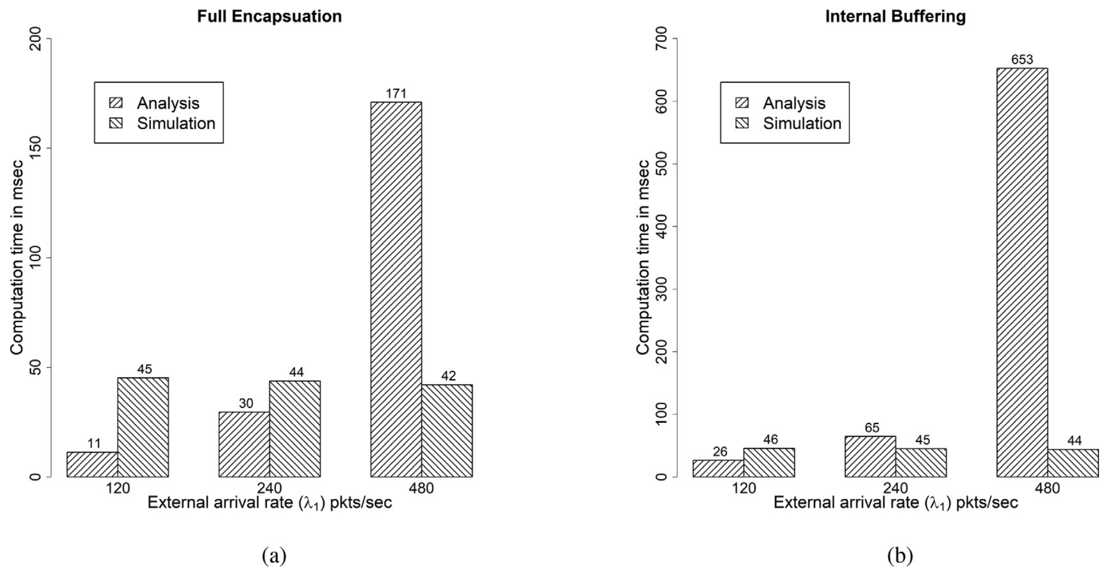


Fig. 6. Computation time between analysis and simulation for : (a) Full encapsulation and (b) Internal buffering.

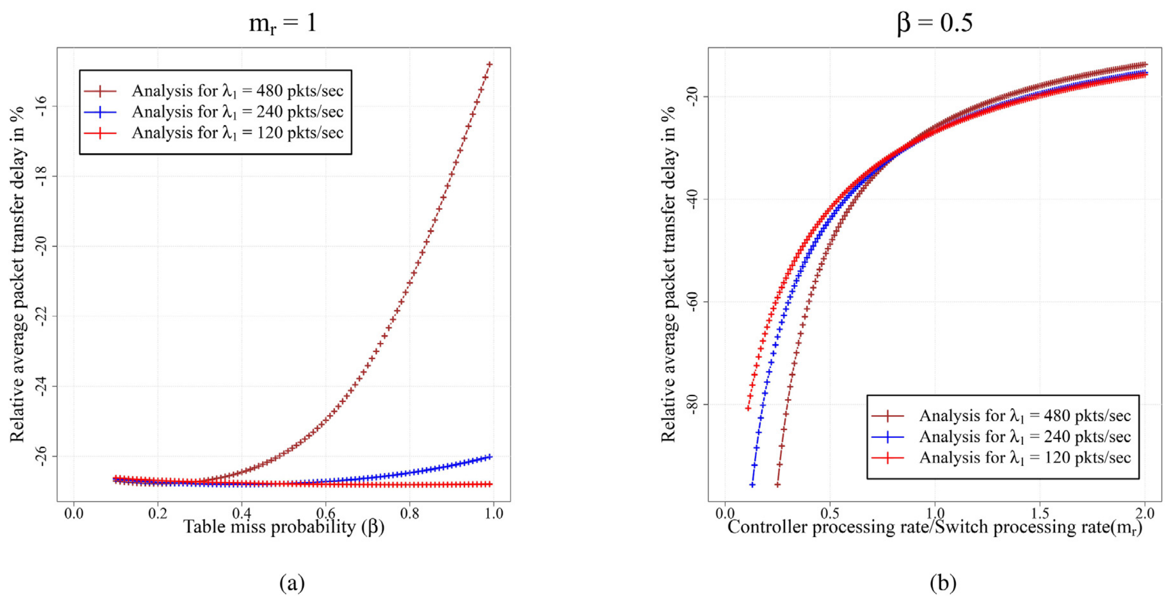


Fig. 7. Relative average packet transfer delay between the full encapsulation model and internal buffering model i.e. ϵ_d for (a) increasing β and $m_r = 1$; and (b) increasing m_r and $\beta = 0.5$.

a high buffer capacity, the matrix size increases signigicanlty causing significant increase in the computation time using the matrix-analytic method. The computation overload for the internal buffering increases drastically because of a higher multi-dimensional state space than the full encapsulation.

5.3. Comparing average delay

In this subsection, the average packet transfer delay between the full encapsulation (denoted by t_{FE} as in Eq. (3)) and the internal buffering (denoted by t_{IB} as in Eq. (9)) is compared. This comparison helps to investigate the effect of an internal buffer in a hardware switch with reference to the average packet transfer delay.

The relative average packet transfer delay (denoted by ϵ_d) between the full encapsulation and internal buffering (both with finite capacity) is calculated as

$$\epsilon_d = \frac{(t_{IB} - t_{FE})}{t_{FE}} \times 100\%.$$

A positive value of ϵ_d means that the full encapsulation model has a lower average delay for a packet to travel through the network compared to the internal buffering model.

Fig. 7 shows the relative average packet transfer delay between the full encapsulation and internal buffering models in percentile. In Fig. 7(a), where m_r is constant at 1, it can be observed that the internal buffering method achieves up to 28% reduction in the average packet delay compared to the full encapsulation. With increasing β, the number of data packets sent to the controller is increased. These data packets requiring decisioning from the controller are processed by the CPU which has a slower processor than the specialised hardware. With internal buffering, the CPU encapsulates a small part of a data packet (mostly packet header information) in the Packet-In message which is processed faster by the

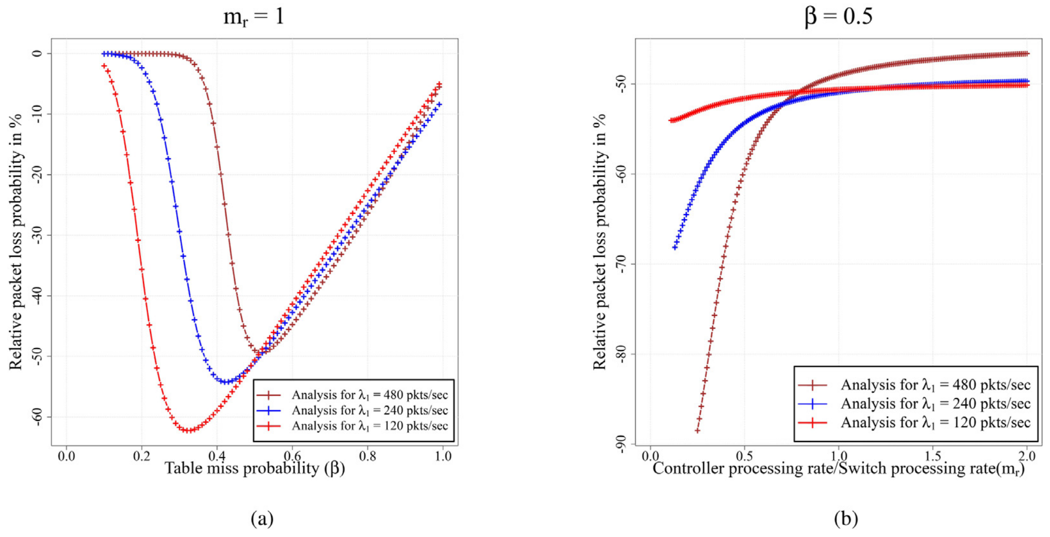


Fig. 8. Relative average packet loss probability between the full encapsulation model and internal buffering model i.e. ϵ_t for (a) increasing β and $m_r = 1$; and (b) increasing m_r and $\beta = 0.5$.

controller than Packet-In messages encapsulated with a full data packet. The internal buffering significantly reduces the overall delay for increasing traffic forwarded by the specialised hardware as seen in Fig. 7(a).

With the increasing m_r, as shown in Fig. 7(b), the controller processing capacity increases. However, the performance of controller degrades with increasing flow update requests from the switch. The real benefit of an internal buffer at the OpenFlow switch can be observed for a slower controller, represented by lower m_r value, as seen in Fig. 7(b) where the average packet transfer delay is reduced up to 85%. As seen in Fig. 7(b), the relative average packet transfer delay decreases from 85% to 20% with increasing processing capacity of the controller. With the internal buffering of packets awaiting decision from the controller, the control traffic size can be reduced allowing the controller to process packets faster.

This reduction in average packet transfer delay with lower β and m_r shows the benefit of internal buffering over the full packet encapsulation method for hardware switching in an OpenFlow switch.

5.4. Comparing packet loss probabilities

In this subsection, the average packet loss probability between the full encapsulation (PL_{FE} in Eq. (5)) and the internal buffering (PL_{IB} in Eq. (11)) is compared. This comparison highlights the effect of internal buffer in a hardware switch with reference to the packet loss probability.

The relative packet loss probability (denoted by ϵ_l) between the full encapsulation and internal buffering (both with finite capacity) is calculated as

$$\epsilon_l = \frac{(PL_{IB} - PL_{FE})}{PL_{FE}} \times 100\%.$$

A positive value of ϵ_l means the full encapsulation model has a lower packet loss probability compared to the internal buffering model.

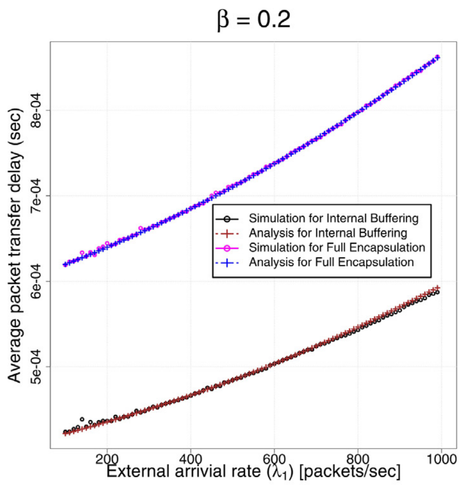
Fig. 8 shows the relative packet loss probability between the full encapsulation and internal buffering models in percentile. Fig. 8(a) and (b) show the relative packet loss probability for increasing β and m_r, respectively. In Fig. 8(a), we show that internal buffering exhibits up to 60% drop in the packet loss probability for lower packet arrival rates, viz. $\lambda_1 = 120$ or 240 pkts/sec. At higher packet arrival rate, $\lambda_1 = 480$ pkts/sec, the 60% drop in packet loss probability at lower β reduces quickly to 6% as β increases. This is due to the finite buffer capacity for the internal buffer. The finite internal buffer can temporarily store limited number of data packets. With a lower value of β, the number of data packets to be stored in the internal buffer is lower resulting in a lower packet loss probability. Whereas, with a higher value of β, the number of data packets to be stored in the internal buffer is higher resulting in a higher packet loss probability. The higher packet loss probability in the internal buffer reduces the benefit of the internal buffering model over the full encapsulation model as seen in Fig. 8(a). Therefore, the full encapsulation method is more appropriate to handle incoming flows that have a higher table miss probability.

Similarly, in Fig. 8(b), we show that the internal buffering model exhibits up to 89% reduction in the packet loss probability for lower m_r and up to 50% reduction for higher m_r. It is observed that higher value of λ_1 (i.e 480 pkts/sec) significantly reduces packet loss probability for a lower value of m_r (i.e $m_r < 1$), whereas the lower value of λ_1 (i.e 120 or 240 pkts/sec) shows better packet loss probability for a higher value of m_r (i.e $m_r > 1$). This is because lower m_r means the controller is slower than the CPU and higher m_r means the controller is faster than the CPU. The slower controller processes Packet-In messages slower and consequently increases the blocking of data packets in the CPU. With the internal buffer, the blocking of data packets in the CPU is significantly reduced. On the other hand, the faster controller processes the Packet-In messages faster and reduces the blocking of data packets in the CPU. This reduces the benefit of the internal buffer especially for a higher value of λ_1.

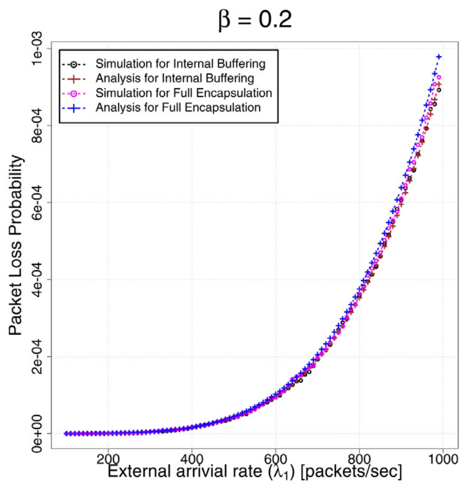
This shows the benefit of internal buffering over full encapsulation method for the OpenFlow-based hardware switching, that significantly reduces the packet loss probability.

6. Traffic and switch parameters

In this section, we perform sensitivity analysis to justify our selection of values for λ_1 (that are 120, 240, 480 pkts/sec) in Section 5. We also justify the selection of μ_{sh} as 1000 times of μ_{sp} by varying m_s in the full encapsulation and internal buffering models. Finally, we use realistic values of parameters for Zodiac switch based on [20] to compare the full encapsulation and internal buffering models.



(a)



(b)

Fig. 9. Effect of increasing λ_1 on the full encapsulation method and internal buffering model for (a) average packet transfer delay (b) and packet loss probability.

For this section, the value of β is fixed to 0.2 which is typical β value for a flow in an OpenFlow-based SDN switch [20]. This value of β is also close to the realistic β values shown in Table 5.

6.1. Sensitivity analysis by varying λ_1

For this analysis, the value of λ_1 is varied from 10 to 999 packets/sec. The maximum value of λ_1 is set as 999 packets/sec which is less than μ_{sp} of 1000 packet/sec to maintain a stationary distribution in the CPU.

Fig. 9 shows the effect of varying λ_1 between the full encapsulation and internal buffering models. Fig. 9(a) and (b) show the average packet transfer delay and packet loss probability, respectively, for increasing λ_1. From Fig. 9(a), the average packet transfer delay for both the full encapsulation and internal buffering models increases with increasing λ_1. The difference between these models looks uniform as seen in Fig. 7(a) for β of 0.2.

Similarly, from Fig. 9(b), the average packet loss probability for both the full encapsulation and internal buffering models are identical and increases with increasing λ_1. The differences between these models are almost negligible as seen in Fig. 8(a) for β of 0.2.

Therefore, the selection of λ_1 as {120, 240, 480} packets/sec or any other positive values less than μ_{sp} for given β would not affect the relative performance of the full encapsulation and internal buffering models.

6.2. Sensitivity analysis by varying μ_{sh}

Next, we study the effects of μ_{sh} through m_s which is varied from 10 to 1000 (i.e. $\mu_{sh} = m_s \times \mu_{sp}$). The value of λ_1 is fixed to 480 packets/sec.

Fig. 10 shows the effect of varying m_s between the full encapsulation and internal buffering models. Fig. 10(a) and (b) show the average packet transfer delay and packet loss probability, respectively, for increasing m_s. From Fig. 10, the average packet transfer delay and packet loss probability for both models remain steady for $m_s > 100$.

From Fig. 10(a), the average packet transfer delay for the full encapsulation model remains uniformly higher than the internal buffering model with increasing m_s. This is because higher value of μ_{sh} reduces the average packet transfer delay uniformly in the specialised hardware for both models.

Similarly, from Fig. 10(b), the average packet loss probability for the full encapsulation model and internal buffering model remains identical. This is because loss incurred by the specialised hardware is negligible and is almost zero with increasing m_s.

Therefore, the selection of m_s as 1000 or any other values greater than 100 for given β and λ_1 would not affect the relative performance of the full encapsulation and internal buffering models.

Table 5
Measured parameters for hardware switches.

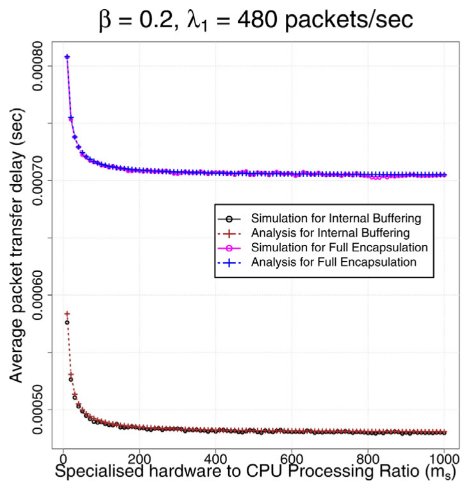
Parameter	Zodiac	P3295
CPU service rate, μ_{sp} (packets/ms)	1.536	132,000
Controller service rate, μ_c (packets/ms)	38568	
Arrival rate, λ_1 (packets/ms)	{1.8, 12.6, 33.2, 74.6}	
Average flow size, $1/\beta$ (packets)	{6.6, 15.3, 9.3, 9.8}	

6.3. Using realistic values for parameters

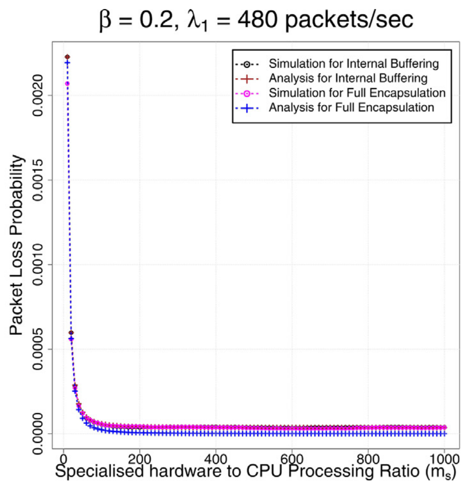
Lastly, we validate the models using realistic values of parameters for the switches and controllers which is shown in Table 5. The parameters are measured in a network using a controller implemented on a Dell OptiPlex 9010 running Ubuntu 16.04 LTS (64-bit), using Intel i7-3770 CPU (3.40 GHz $\times$ 8) with 2 $\times$ 4 GB DDR3 RAM (1600 MHz each). The Zodiac switch is a low end switch by Northbound Networks while the P3295 switch is a medium range switch from Pica8. We use the Ryu controller running OpenFlow v1.3 to manage the hardware switches.

The remaining parameters other than those shown in Table 5 are same as in Table 4. Figs. 11 and 12 show bar charts for the average packet transfer delay and packet loss probability using the fixed values of {λ_1, $1/\beta$}. Fig. 11 shows the average packet transfer delay for Zodiac and P3295 switches, while Fig. 12 shows the packet loss probability for Zodiac switch only. This is because the packet loss probability for P3295 switch is negligible and almost zero for all fixed values of {λ_1, $1/\beta$}. The CPU service rate for P3295 switch is approximately 86,000 times faster than Zodiac switch resulting in negligible packet losses in P3295 switch.

From Fig. 11(a) and (b), we observe that the internal buffering model achieves lower average packet transfer delay than the full encapsulation model as previously observed in Figs. 7(a), 9(a) and 10(a). Similarly, from Fig. 12, the full encapsulation and inter-

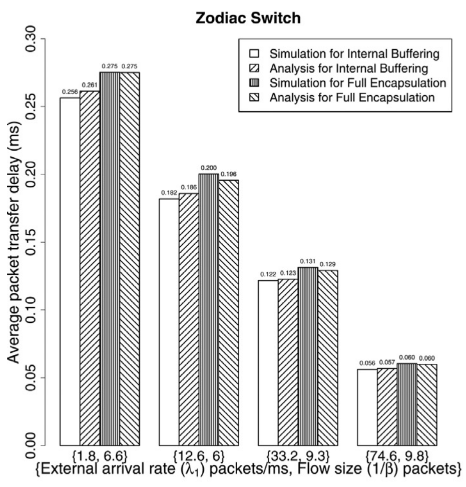


(a)

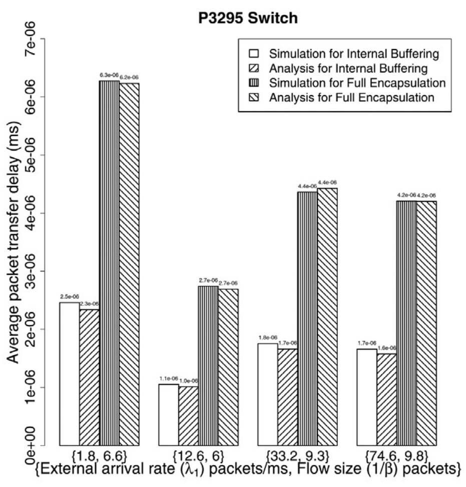


(b)

Fig. 10. Effect of increasing μ_{sh} via increasing m_s on the full encapsulation method and internal buffering model for (a) average packet transfer delay (b) and packet loss probability.



(a) Zodiac



(b) P3295

Fig. 11. Average packet transfer delay for fixed {λ_1, μ_{sh}} on the full encapsulation model and internal buffering model using real switch parameters.

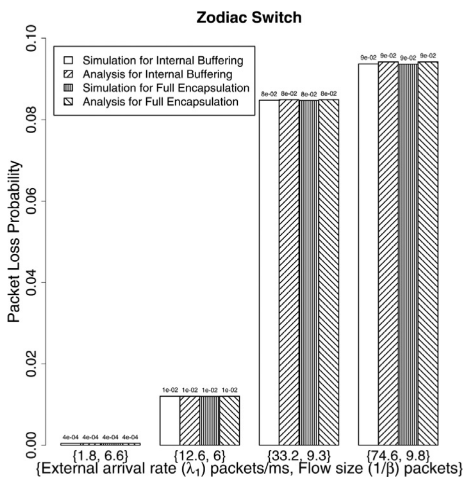


Fig. 12. Packet loss probability for fixed {λ_1, μ_{sh}} on the full encapsulation model and internal buffering model using Zodiac switch's parameters. The packet loss probability for the P3295 switch is close to zero and therefore not shown.

nal buffering models both have similar packet loss probability for lower β as previously observed in Figs. 8(a), 9(b) and 10(b).

Therefore, it shows that the models proposed in this paper can be used to predict an SDN performance with real parameters. The analysis in this subsection also validates our analysis in Section 5.

7. Conclusion

In this paper, the two packet encapsulation methods based on the OpenFlow specification for hardware switching were compared using queueing models, viz., the full encapsulation model and the internal buffering model, respectively.

From comparisons, the following conclusions were made:

- Internal buffering significantly reduces the average packet transfer delay (almost 85%) for a slower controller and (almost 20%) for a faster controller than the full packet encapsulation method.
- Internal buffering significantly reduces the packet loss probability (almost 60%) for a lower table miss probability and (almost 6%) for a higher table miss probability than the full encapsulation method.

From above-mentioned conclusions, the following guidelines can be provided to network operators for optimum performance:

- For a delay-sensitive SDN, network operators can choose the internal buffering method over the full encapsulation method when a slower controller is used.
- For a loss-sensitive SDN, the full encapsulation method is preferred over the internal buffering method for OpenFlow switches. This choice becomes more evidence with a higher table miss probability.

Finally, from sensitivity analyses by varying λ_1 and μ_{sh} in the full encapsulation and internal buffering models, following conclusions were made to observe similar trends in the performance analysis irrespective of values used for parameters:

- The value of an external arrival rate at the switch TCAM can be of any positive values but less than the CPU processor for fixed table miss probability.
- The value of the switch processor should be at least a hundred times of the CPU processor for fixed table miss probability and external arrival rate.

This work can be extended by considering the partial encapsulation of data packets. The partial encapsulation of data packets is the combination of the full encapsulation and internal buffering. In the partial encapsulation of data packets, OpenFlow switches with limited memory use the full encapsulation method when they run out of memory for the internal buffering. Also, in this work, the internal buffer is used to temporarily buffer data packets that are to be forwarded to the controller. However, future works can consider the use of internal buffering to buffer other data packets as well to avoid packet losses at the switch with limited buffer capacities.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The work of Deepak Singh was supported by a Victoria Doctoral Scholarship. Bryan Ng and Winston K.G. Seah were supported in part by Victoria University's Huawei NZ Research Programme, Software-Defined Green Internet of Things project #E2881.

Supplementary material

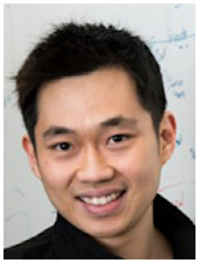
Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2019.107033.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[2] L. Yang, R. Dantu, T. Anderson, R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, 2004, Internet Engineering Task Force, RFC 3746. http://www.tools.ietf.org/html/rfc3746.
[3] P. Goransson, C. Black, Software Defined Networks: a Comprehensive Approach, Elsevier, 2014.
[4] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, A.V. Vasilakos, Software-defined industrial internet of things in the context of industry 4.0, IEEE Sens. J. 16 (20) (2016) 7373–7380.
[5] R. Trivisonno, R. Guerzoni, I. Vaishnavi, D. Soldani, SDN-Based 5G mobile networks: architecture, functions, procedures and backward compatibility, Trans. Emerg. Telecommun.Technologies 26 (1) (2015) 82–92.
[6] 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, in: ACM SIGCOMM Computer Communication Review, volume 43, ACM, 2013, pp. 3–14.
[7] Y. Gwon, H.T. Kung, Inferring origin flow patterns in wi-fi with deep learning, in: ICAC, 2014, pp. 73–83.
[8] B.P.S. Sahoo, C.-C. Chou, C.-W. Weng, H.-Y. Wei, Enabling millimeter-wave 5G networks for massive IoT applications, (2018) arXiv:1808.04457.
[9] ONF, OpenFlow Switch Specification, Technical Report, Open Networking Foundation, 2013.
[10] H.H. Kurmann, H.M. Kurmann, On the Emulation of Impairments in ATM-networks, vdf Hochschulverlag AG, 1997.
[11] J. Mao, B. Han, Z. Sun, X. Lu, Z. Zhang, Efficient mismatched packet buffer management with packet order-preserving for openflow networks, Comput. Netw. 110 (2016) 91–103.
[12] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, W.K.G. Seah, Modelling Switches with Internal Buffering in Software-Defined Networks, in: Proceedings of the 27th International Conference on Computer Communication and Networks (ICCCN), 2018. Hangzhou, China.
[13] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, W.K.G. Seah, Analytical modelling of software and hardware switches with internal buffer in software-Defined networks, J. Netw. Comput. Applica. 126 (2019) 22–37.
[14] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: a comprehensive survey, Proceed. IEEE 103 (1) (2015) 14–76.
[15] B. Davie, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Gude, A. Padmanabhan, T. Petty, K. Duda, A. Chanda, A database approach to SDN control plane design, SIGCOMM Comput. Commun. Rev. 47 (1) (2017) 15–26.
[16] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, W.K.G. Seah, Modelling software-defined networking: software and hardware switches, J. Netw. Comput. Applica. 122 (2018) 24–36.
[17] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, P. Tran-Gia, Modeling and performance evaluation of an OpenFlow architecture, in: Proceedings of the 23rd International Teletraffic Congress (ITC), San Francisco, CA, USA, 6–9 November, 2011, pp. 1–7.
[18] K. Mahmood, A. Chilwan, x amp, a sterb, O. #x00F, M. Jarschel, Modelling of openflow-based software-defined networks: the multiple node case, IET Netw. 4 (5) (2015) 278–284.
[19] W. Miao, G. Min, Y. Wu, H. Wang, J. Hu, Performance modelling and analysis of software-Defined networking under bursty multimedia traffic, ACM Trans. Multim. Comput. Commun. Applica. (TOMM) 12 (5s) (2016) 77.
[20] Y. Goto, H. Masuyama, B. Ng, W.K.G. Seah, Y. Takahashi, 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), 2016. London, UK.
[21] A. Fahmin, Y.-C. Lai, M.S. Hossain, Y.-D. Lin, Performance modeling and comparison of NFV integrated with SDN: under or aside? J. Netw. Comput. Applica. 113 (2018) 119–129.
[22] M. Kuźniar, P. Perešíni, D. Kostić, What you need to know about SDN flow tables, in: Passive and Active Measurement, Springer, 2015, pp. 347–359.
[23] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, G. Xie, 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, 2013, pp. 85–90.
[24] D.B. Rawat, S.R. Reddy, Software defined networking architecture, security and energy efficiency: a survey, IEEE Commun. Surv. Tutor. 19 (1) (2017) 325–346, doi:10.1109/COMST.2016.2618874.
[25] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, W.K.G. Seah, Modelling Software-Defined Networking: Switch Design with Finite Buffer and Priority Queueing, in: Proceedings of the IEEE 42nd Conference on Local Computer Networks (LCN), 2017. Singapore.
[26] I. Adan, J. Resing, Queueing Theory, Eindhoven University of Technology, Eindhoven, 2002 http://www.home.ewi.utwente.nl/~scheinhardtwrw/queueingdictaat.pdf.
[27] G. Bolch, S. Greiner, H. de Meer, K.S. Trivedi, Queueing Networks and Markov Chains, 2nd, John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.
[28] G. Latouche, V. Ramaswami, Introduction to Matrix Analytic Methods in Stochastic Modeling, 5, Siam, 1999.
[29] R.J. Simcoe, T.-B. Pei, Perspectives on ATM switch architecture and the influence of traffic pattern assumptions on switch design, SIGCOMM Comput. Commun. Rev. 25 (2) (1995) 93–105.
[30] S.C. Liew, Performance of various input-buffered and output-buffered ATM switch design principles under bursty traffic: simulation study, IEEE Trans. Commun. 42 (234) (1994) 1371–1379, doi:10.1109/TCOMM.1994.580245.
[31] ISO/IEC/IEEE International standard for ethernet, ISO/IEC/IEEE 8802-3:2014(E) (2014) 1–3754, doi:10.1109/IEEESTD.2014.6781545.
[32] D.A. Bini, B. Meini, S. Steffe, B. Van Houdt, Structured markov chains solver: software tools, in: Proceeding from the 2006 workshop on Tools for solving structured Markov chains, ACM, 2006, p. 14.
[33] G. Latouche, V. Ramaswami, A logarithmic reduction algorithm for quasi-birth-death processes, J. Appl. Probab. 30 (3) (1993) 650–674.
[34] T. Hanschke, A matrix continued fraction algorithm for the multiserver repeated order queue, Math. Comput. Model. 30 (3–4) (1999) 159–170.
[35] T. Dayar, W. Sandmann, D. Spieler, V. Wolf, Infinite level-dependent QBD pro-

cesses and matrix-analytic solutions for stochastic chemical kinetics, Adva. Appl. Probab. 43 (4) (2011) 1005–1026.

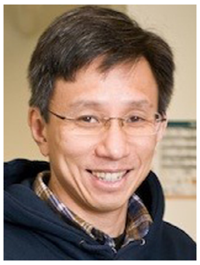
Deepak Kumar Singh received the received the Dr.Eng. degree from Victoria University of Wellington, New Zealand, in 2019. He is currently working as a Research Assistant in Victoria University of Wellington, New Zealand. His research focuses on modelling of Software-Defined Network, data modelling and recommendation system.



Bryan Ng completed his Ph.D. (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 2007008, 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.