symmetry

MDPI

Article

A Rapid Deployment Mechanism of Forwarding Rules for Reactive Mode SDN Networks

Ming-Tsung Kao [1], Shang-Juh Kao [1], Hsueh-Wen Tseng [1] and Fu-Min Chang [2,*]

[1] Department of Computer Science and Engineering, National Chung-Hsing University, 145, Xingda Rd., South District, Taichung 402, Taiwan; kevinkao@cs.nchu.edu.tw (M.-T.K.); sjkao@cs.nchu.edu.tw (S.-J.K.); hwtseng@nchu.edu.tw (H.-W.T.)

[2] Department of Finance, Chaoyang University of Technology, 168, Jifeng E. Rd., Wufeng District, Taichung 41349, Taiwan

* Correspondence: fmchang@cyut.edu.tw

Abstract: In reactive mode software-defined networking (SDN) networks, a new initiated flow requires back-and-forth communications between the controller and the switches along the forwarding route. As SDN is getting popularly accepted, many studies have reported on how to reduce the amount of communication traffic and to release the controller's loading. Several techniques have been proposed, such as proactive and active mode integration, MPLS adoption, and various forwarding rule installation techniques. In this paper, we adopt the idea of the tunnel penetration technique, called the tunnel boring machine in SDN or SDN-TBM, to effectively cut down the traffic between switches and the controller as well as to speed up packet delivery. Using the TBM mechanism, the communication symmetry between the controller and the switches on the path is broken and transformed into asymmetry. Only the first and last switches of each application flow need to make forwarding queries to the controller, and all intermediate switches simply forward packets consisting of the forwarding information needed to determine the next-hop switch. An M/M/1 queueing model is developed to verify the feasibility and efficiency of the proposal. Under the simulation of more than a million flows with 3–8 intermediate switches, the packet sojourn time using SDN-TBM mechanism is less than that of adopting the conventional SDN and JumpFlow model. Additionally, by adopting SDN-TBM, both the number of packet-in and packet-out packets and the controller's loading are significantly reduced.

Keywords: software-defined networking; Openflow; reactive mode; forwarding rules

Citation: Kao, M.-T.; Kao, S.-J.; Tseng, H.-W.; Chang, F.-M. A Rapid Deployment Mechanism of Forwarding Rules for Reactive Mode SDN Networks. *Symmetry* **2022**, *14*, 1026. https://doi.org/10.3390/sym14051026

Academic Editors: Teen-Hang Meen, Charles Tijus, Chun-Yen Chang and Po-Lei Lee

Received: 9 April 2022
Accepted: 15 May 2022
Published: 17 May 2022

1. Introduction

Software-defined networking (SDN) [1,2] is a centralized, revolutionized technology for alternative network management control, which separates network operations into the control plane and the data plane [3,4]. The controller controls the packet forwarding route through the control plane using OpenFlow protocol [5,6] and packets are transmitted among switches through the data plane. Relying on the information of network usages collected from all switches, the controller is able to efficiently provide a forwarding decision for each flow. In addition to define the mechanism for forwarding rules composition, the OpenFlow protocol also provides the way to deal with the incoming packets over the switch, such as to modify the destination IP address, to change the source MAC address, or to be associated with a VLAN ID. Multiple actions, called an action set, can also be defined to control SDN network operations.

As in SDN networks, the switch always refers the flow entries to forward the flow packet; when and how to manage the flow table is critical in forwarding efficiency. In the flow table, each flow entry defines a forwarding rule, which specifies the switch of the next hop and actions of how to deal with the packet. The approach of deploying flow entries can be either a proactive mode or reactive mode [7]. In the proactive mode, the

controller determines the forwarding path before the arrival of a packet by writing the forwarding rules into the flow tables in advance. When a packet arrives, the switch simply follows the forwarding rule to transmit the packet. The forwarding policy of the proactive mode is similar to traditional IP network routing, MPLS [8], or segment routing [9]. The proactive mode operations are suitable for a static network environment where routing paths are not frequently adjusted. In reactive mode SDN, upon the arrival of a packet, the flow table at the switch is inspected, and if there is an entry associated with the packet's destination, the packet is forwarded accordingly. Otherwise, the switch sends a query request, called a packet-in message, to the controller for forwarding determination. Upon receiving a packet-in packet, the controller returns a packet-out message to the switch, which installs a flow entry to the flow table at the switch. Consequently, the pending packet can be successfully transmitted following the forwarding rule. The forwarding route management at the controller enhances the capability of balancing the network resource usage. Opposed to the proactive mode, the operations of the reactive mode are more suitable for the dynamic network environment.

Most current data centers involve dynamic virtual machines connection to provide network services; the reactive mode SDN is well suited for this frequently changed network environment. However, with the reactive mode operations, the controller requires dialogs for all switches on the forwarding path. The number of dialogs is symmetric to the number of switches. Therefore, issuing forwarding rules for each unfamiliar packet, the controller can easily become a bottleneck. If a packet passes through N switches to reach its destination, there are 2N communications between the controller and switches, which may increase the controller's loading as N becomes larger. Even though all subsequent packets in a flow may reference an existing flow rule for forwarding without the controller's interventions, most current network applications, such as social networks IoT [10], UAV [11], NIDS [12], etc., are prone to generate a large volume of unfamiliar flows. The occurrences of unfamiliar flows will certainly increase the traffic load over the controller and hence cause the processing delay. Therefore, for the reactive mode SDN applications to be practical, novel approaches of effectively reducing the packet sojourn time and minimizing the number of packet-in messages deserve further study and development.

Under reactive mode SDN networks, in order to effectively reduce the number of packet-in messages, we borrowed the idea from the tunnel boring machine (TBM) [13]. A TBM, also known as 'Mole', is a machine used to excavate a tunnel with circular cross sections in various solid and rock formations. We adopt the concept of tunnel excavation to establish forwarding paths. As the TBM is placed at the entrance of a tunnel, the machine excavates a hole following the scheduled trail until it reaches the exit. When a switch first connects to the controller asking for an unfamiliar packet transmission, which is similar to the initial setup for TBM at the entrance, the controller generates forwarding rules for the switches along the path. Meanwhile, the packet is temporarily buffered at the controller until the tunnel is passed through. In other words, we emulate the tunnel penetration technique to establish a forwarding path and propose a mechanism called the SDN tunnel boring machine (SDN-TBM). The idea is to leave the first packet of a flow at the controller until the forwarding path is established, while only the boring packet (BP), which contains forwarding rules and actions, is transmitted over the switches along the path. Consequently, all intermediate switches can be waived from the sending packet-in messages to the controller.

The rest of the paper is organized as follows. Related work is given in Section 2. The architecture and operations of SDN-TBM are presented in Section 3. In Section 4, an M/M/1 queuing model is developed for the system performance. The evaluation results are then reported, and conclusions are given.

2. Related Work

In the study of reactive mode SDN networks, several efforts have been presented to reduce the controller's load, such as multiple controllers, flow entries merge, and MPLS

label switching. Ma et al. [14] proposed a meta controller (MC), which is based on building a manager for several controllers to control SDN operations in a decentralized manner. The key idea of multiple controllers is to redistribute switch–controller association to offload the overloaded controllers. Meta controllers may reduce the load over the control plane but could also increase the overall bandwidth usage. Neghabi et al. [15] surveyed several multi-controller proposals. Most of the proposed approaches focus on controllers' load balancing. Generally, the multi-controller approach may distribute the load, but it is costly and may suffer from low scalability and low availability, as well as high complexity. Moreover, the multi-controller approach may not be able to reduce packet-in messages.

Flow entries can also be logically merged depending on the tags, as reported by MacDavid et al. [16]. They proposed PathSets, which uses a compression encoding scheme to insert a compact tag into packet header. By the increased probability of flow matching, the merging approach can be effective to reduce the number of packet-in messages. However, the forwarding route may lack flexibility due to the invariant bundle of flow rules. This approach is suitable for data centers with stable network topology but is not for topology with frequent changes.

One other approach to reduce the number of packet-in messages is to adopt the concept of label switching as in MPLS [17–20]. Under the MPLS-based approach, routing information is encoded by the 12-bits VLAN identifier (VID). When a packet enters its ingress switch, it is encapsulated with multiple port numbers, which map to the forwarding switches on the route. Intermediate switches will use the VID to forward the packet to the next hop and delete the used header until the packet is delivered to its final destination. The VID field can record 3–5 hops. The scheme cannot need the controller service when receiving an unfamiliar packet, except for ingress switch. Therefore, it effectively reduces the number of packet-in messages. However, the switches may only forward packets (or flows) to designated ports but be unable to perform more actions for dealing with the packets. Consequently, the ability of forwarding the packet is restricted, such as changing the source or the destination MAC or IP addresses. Hence, even though the MPLS-based approach can solve the controller's overloading problem, SDN features may not be fully implemented.

Once the first packet-in message of a flow has been received by the controller and a forwarding path has been determined, the controller may send packet-out messages to all intermediate switches. It seems that this approach may be effective in reducing the number of packet-in messages. However, the arrival of the packet-out message may be later than the arrival of a subsequent packet over the switch, which may cause the switch to send packet-in messages. Hu et al. proposed a scheme called Softring [21] to allow the packet to stay in a circular route at the switch to avoid unnecessary packet-in requests. However, this approach requires extra processing time and extra space to deal with the packet at the switch, which causes an overhead to the switch. We briefly explain the pros and cons of the SDN-TBM mechanism compared to other solutions in Table 1.

Table 1. Comparison of different methods.

Method	Pros	Cons
Multiple Controllers	Distribute the requirements of the switch and reduce the overload of the controller.	Management of synchronous flow rules and fault tolerance mechanisms are not easy to implement.
PathSets	Effective to reduce the number of packet-in messages.	Forwarding route may lack flexibility due to an invariant bundle of flow rules.
MPLS-based	A simple implementation is possible.	Only forward packets to the designated port but is unable to perform more packet processing operations.
Reacitve and proactive mix mode	Effective in reducing the number of packet-in messages.	May cause packet-in messages to be sent repeatedly.
SDN-TBM	Effectively addresses the high load of reactive mode on the controller.	Need to add switch functionality

3. Design of SDN-TBM

As in reactive mode SDN networks, when an unfamiliar packet arrives at a switch, the switch issues a packet-in message to the controller, waits for a forwarding instruction from the controller, and resumes its packet forwarding process. All intermediate switches along the route to the destination follow the similar interactions with the controller, which results in a severe delay and a large number of communications involved. In our proposed approach, only the first and the final switches along the forwarding route are required to connect to the controller, while a light packet, called a boring packet (BP), including only the requisite information for packet forwarding is transmitted between switches. The use of boring packets is different to the circular route design in Softring, which allows all intermediate switches along the path to be free in communication with the controller for transmitting a service flow.

3.1. Overview of SDN-TBM Operations

Other than conventional SDN functions, the SDN-TBM system adds three functional modules to enable the boring packet processing: boring packet processor (BPP), TBM processor (TBMP), and packet holding store (PHS), as shown in Figure 1 of the SDN-TBM architecture. BPP is included in each OpenFlow switch and plays the role of both packet-in message transmitter and boring packet resolver, while TBMP and PHS are extra modules added to the controller. TBMP is the core logic of SDN-TBM, which stores the to-be-delivered packets in PHS and generates boring packets. A to-be-delivered packet is the first packet in a flow, which is temporarily kept in the controller and is delivered when the tunnel has been completely set up. PHS is a database which stores the to-be-delivered packets and the relevant information, such as timestamp, action set, etc.

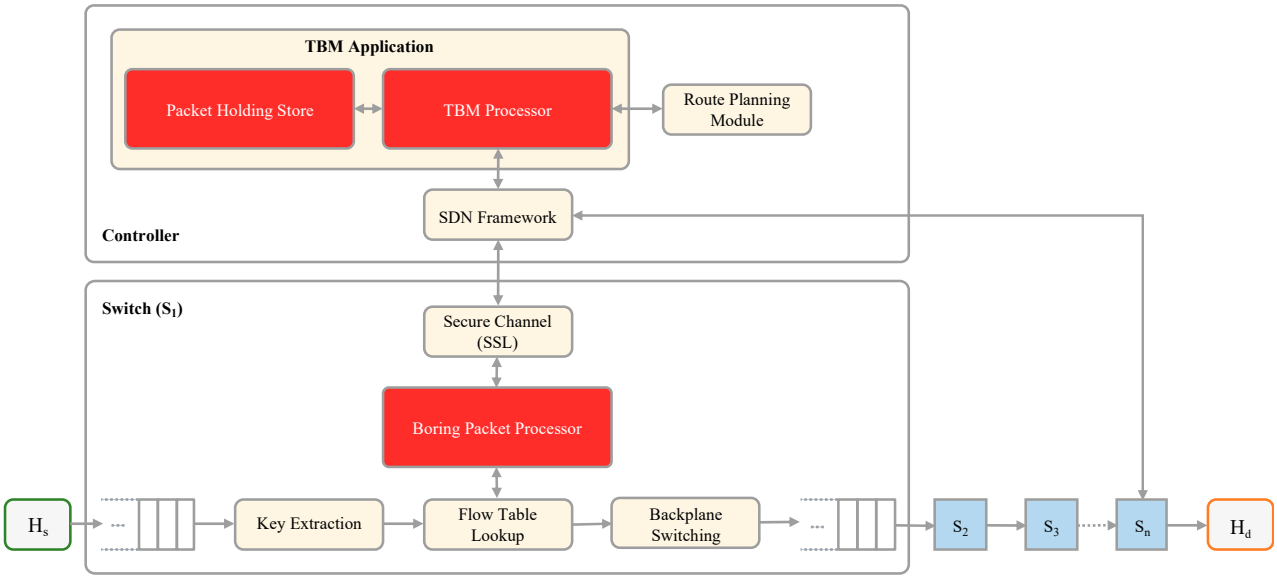


Figure 1. The system architecture of SDN-TBM.

When an unfamiliar flow arrives at the switch, BPP forwards a packet-in message to the controller for the first packet. The packet is passed through BPP to the controller. TBMP on the controller receives the packet-in message and stores it to PHS. The header of the packet is sent to the route planning module for forwarding the path calculation. Once the forwarding rules are determined, the information is provided to TBMP to construct a boring packet (BP). BP, as carried by a packet-out message, is then sent back to the initiated switch. BP consists of the forwarding rules as in action sets for all intermediate switches. Once receiving the BP, the switch updates the flow table and forwards the boring packet to the next switch hop. The next hop switch treats the BP as an unfamiliar packet and issues a packet-in message. However, BPP in the intermediate switch deals with the BP

without sending to the controller, instead, it creates a packet-out boring packet. The flow entry carried by this packet-out BP are then written to the flow table. Meanwhile, the BP is further forwarded to the next switch hop. This procedure continues repeatedly until BP reaches the final switch which the destination host is associated.

When the BP arrives at the last switch, which is the egress switch on the forwarding path, a packet-in boring packet is sent to the controller through the boring packet processor. TBMP in the controller extracts the matched to-be-delivered packet in PHS and constructs a complete packet-out packet, which is the original packet, and returns to the egress switch. Once the first packet of a flow has been successfully delivered, the forwarding tables of intermediate switches along the forwarding path are already updated. All subsequent data packets of the same flow are forwarded accordingly without interventions with the controller.

3.2. Operations at the Controller

Other than built-in SDN framework modules, such as the route planning module, SDN-TBM includes two more modules, packet holding store (PHS) and TBM processor (TBMP), as shown in Figure 1. TBMP communicates with built-in modules and stores the packet into PHS. TBMP is also responsible for generating a packet-out message, either containing a boring packet (BP) or a regular message. PHS stores to-be-delivered packets, including the information of the packet id, timestamp, action_set, and payload for each holding packet. As shown in Figure 2, PID denotes the packet identifier of the to-be-delivered packet, timestamp records the creation time, and action_set records the sequence of actions to be applied to the switch on the forwarding path. For each associated switch, there is an entry including a tag of boring packet, the switch identifier, dbid (or switch id), length of action-set, and action-set. The data structure of the boring packet contains the header portions, which are inherited from the original packet, and payload portions, which consist of tag, dbid, length, and actions, as can be shown in Figure 3.

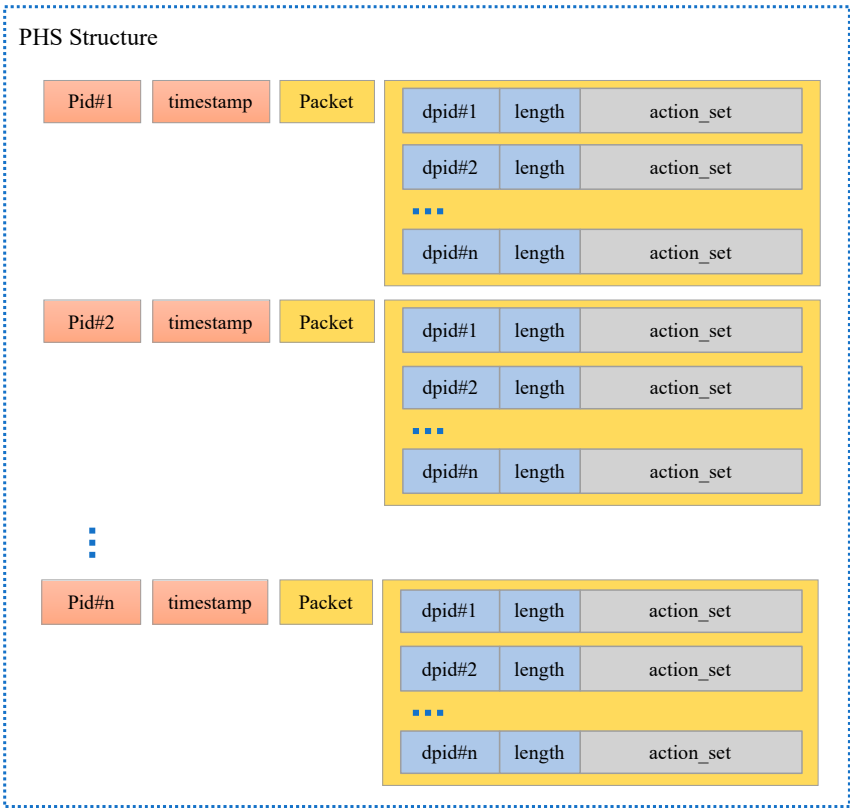


Figure 2. Data structure of PHS.

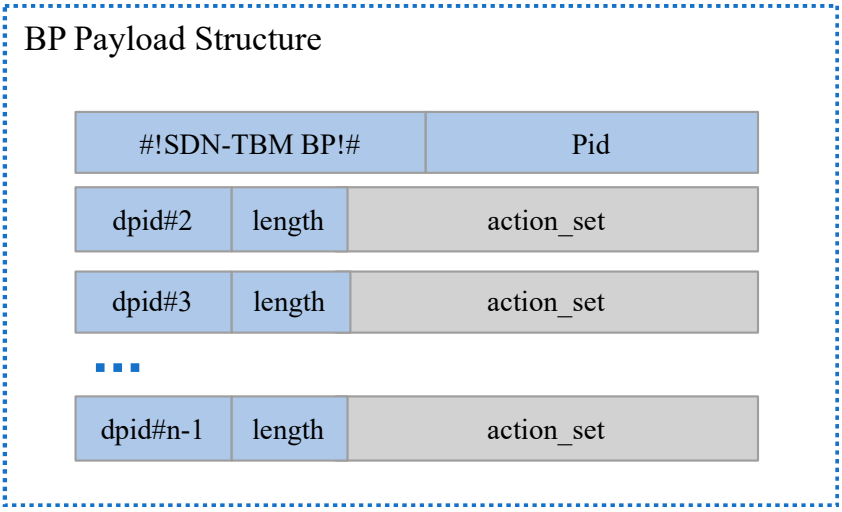


Figure 3. Data structure of BP payload.

The TBMP module plays the central role of SDN-TBM, which performs the core logic for dealing with the packet management at the controller. When a packet is received by the controller, the step-by-step operations of TBMP are described in the following, as supplemented in Figure 4.

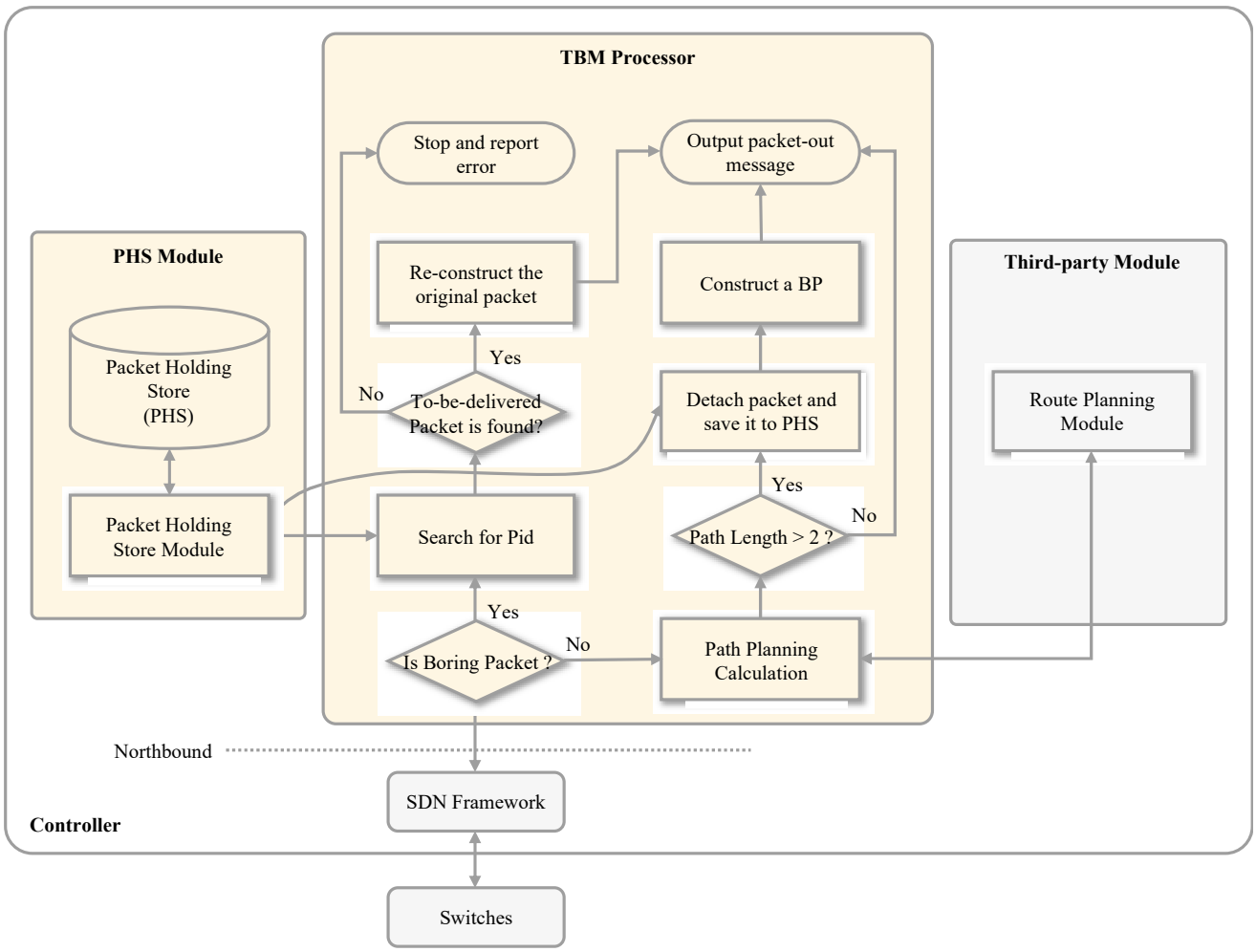


Figure 4. Flowchart of the TBMP.

Step 1: Check whether the incoming packet is a BP or not. If it is a BP, then jump to step 6, otherwise, proceed to step 2.
Step 2: Calculate the forwarding path. The calculation of forwarding path requires interactions with the route planning module in the third-party modules. Forwarding rules for the switches along the path are also determined.
Step 3: Check the number of hops. If the number of hops is 2, a packet-out packet is constructed and is returned back to the switch as regular SDN workflow. Otherwise, prepare to build a boring packet.
Step 4: Store the packet into PHS. If there are more than 2 switch hops in the forwarding path, the packet is temporarily saved in the packet holding store together with relevant forwarding rules. The packet header is extracted for the purpose of generating a boring packet.
Step 5: Generate a packet-out boring packet. Following the saving of the packet, a boring packet is constructed by including the header and all action-sets, as determined in step 2. This packet includes the boring packet identification as well as the identification of to-be-delivered packet, which is previous stored in PHS.
Step 6: Search for stored packet in PHS. It assumes that the incoming packet is a boring packet. This is the case that the last switch in the path sends the packet-in message and asks for the original complete packet. Pid is used as the key to search for the to-be-delivered packet.
Step 7: Is the to-be-delivered packet found? If the target Pid is not matched, the searching process fails, and an error is reported. Otherwise, the original packet is ready to be re-constructed.
Step 8: Replace the BP with original packet to form the packet-out packet. A packet-out message is generated by replacing the BP with the original packet and is returned to the last switch as listed in the boring packet.

The packet holding store (PHS) interacts with the tunnel boring machine processor (TBMP) in two occasions: store the packet before generating a boring packet and retrieve the to-be-delivered packet to construct a packet-out message for the final switch. A packet identifier is created when the packet is stored in PHS, and this identifier is used as an index to search for to-be-delivered packets in the database.

Through the operations at the controller, SDN-TBM differentiates between a regular packet-in packet and a boring packet and generates a packet-out boring packet. Under SDN-TBM, only the switch receiving the first packet from a new flow and the last switch of the forwarding path may send a packet-in packet to the controller; all intermediate switches along the route will not be in touch with the controller.

3.3. Operations at the Switches

An SDN-TBM-enabled system requires the boring packet processor (BPP) to be included in each participating switch. BBP is a functional module that deals with packet-in and packet-out messages, particularly the boring packets. As shown in Figure 5, once a packet arrives at a switch, there is normal packet forwarding if it is matched packet. Otherwise, it fails in searching for a match in flow table, a packet-in message is generated and it is passed to BPP for boring packet inspection. If the arriving packet is not a boring packet, then it is simply forwarded to the controller and waits for a packet-out message sent from the controller. For the purpose of consistent processing, a packet-in message is always passed to the boring packet processor (BPP). When a packet-out boring packet is received by the switch, the switch retrieves the flow entry and updates the flow table. Accordingly, the boring packet is then forwarded to the next hop.

When the BP is received at the next hop switch, the receiving switch routinely looks for a match in the flow table. An unsuccessful match results in generating a packet-in message, and it is sent to a boring packet processor (BPP). After being sure that the packet is a BP, BPP extracts the flow entry for this switch and generates a packet-out message which

contains the boring packet. Subsequently, the flow table is updated, and the boring packet is forwarding accordingly. This procedure continues until the BP reaches the last switch.

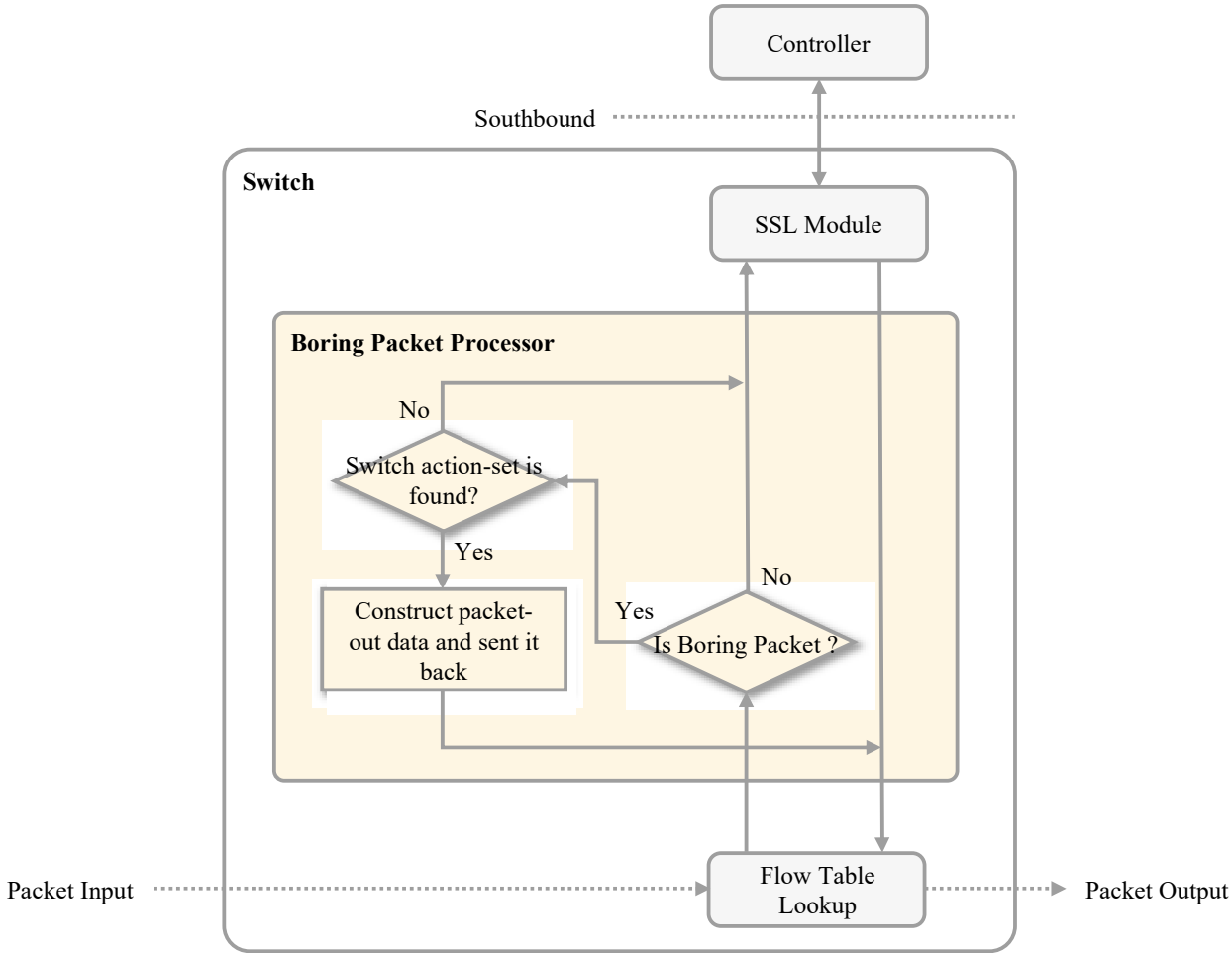


Figure 5. Flowchart of the BP processor.

The last switch requires a special processing for receiving the boring packet. When the BPP at the last switch discovers that it is the final hop in the forwarding path, it transforms the packet-in boring packet to the controller without flow entry update being executed. The controller receiving the packet-in boring packet will reconstruct the original packet and send it as a packet-out packet to the switch. Hence, the last switch is able to directly send the packet to the associated destination host.

3.4. Security Discussion

The most worrisome aspect of the SDN-TBM mechanism is the modification of forwarding rules in BP packets. A hacker who can add, modify, or remove traffic rules from a switch by modifying BP packets can lead to incorrect forwarding behavior on the data plane. Therefore, a hacker may modify BP packets. First, we assume that the controller is secure and that the communication between the switch and the controller is secure. In this way, a hacker can only intercept BP packets on the network link of the data plane or at the switch. In a high control, high security data center, it is difficult to intercept the network link to listen for packets. Moreover, BP packets are only delivered on a certain path, and a hacker cannot listen to all BP packets. If hackers can do so in the data plane, it also means that the IDC is no longer secure. In addition, a hacker can emulate BP packets and inject them into the data platform through the source host. The SDN-TBM mechanism will treat the emulated packets as normal packets and faithfully deliver them to the destination host.

This is because such emulated packets cannot be used by the switch's BP module to change the traffic rules. Therefore, we believe that the SDN-TBM mechanism does not increase the risk of attacks on SDN networks.

4. Analysis Model

In this section, following previous studies [22–29], we develop the corresponding queuing models, shown in Figure 6, to analyze the average sojourn time of an external packet (EP) passing through the conventional SDN and the proposed SDN-TBM networks. For both queuing models, we assume that both switch and controller are Markovian servers, and the service time of the controller incorporates the two-way transmission time between the switch and the controller. The overall traffic process at the switch and the controller follows the Poisson process, and two processes occur in different time slots. In addition, two infinite buffers are deployed in the switch and the controller. Following [22], we model the operation of a switch by using an M/M/1 queue while the operation of a controller is modeled using an M/G/1 queue. Table 2 defines the notation we use in this paper.

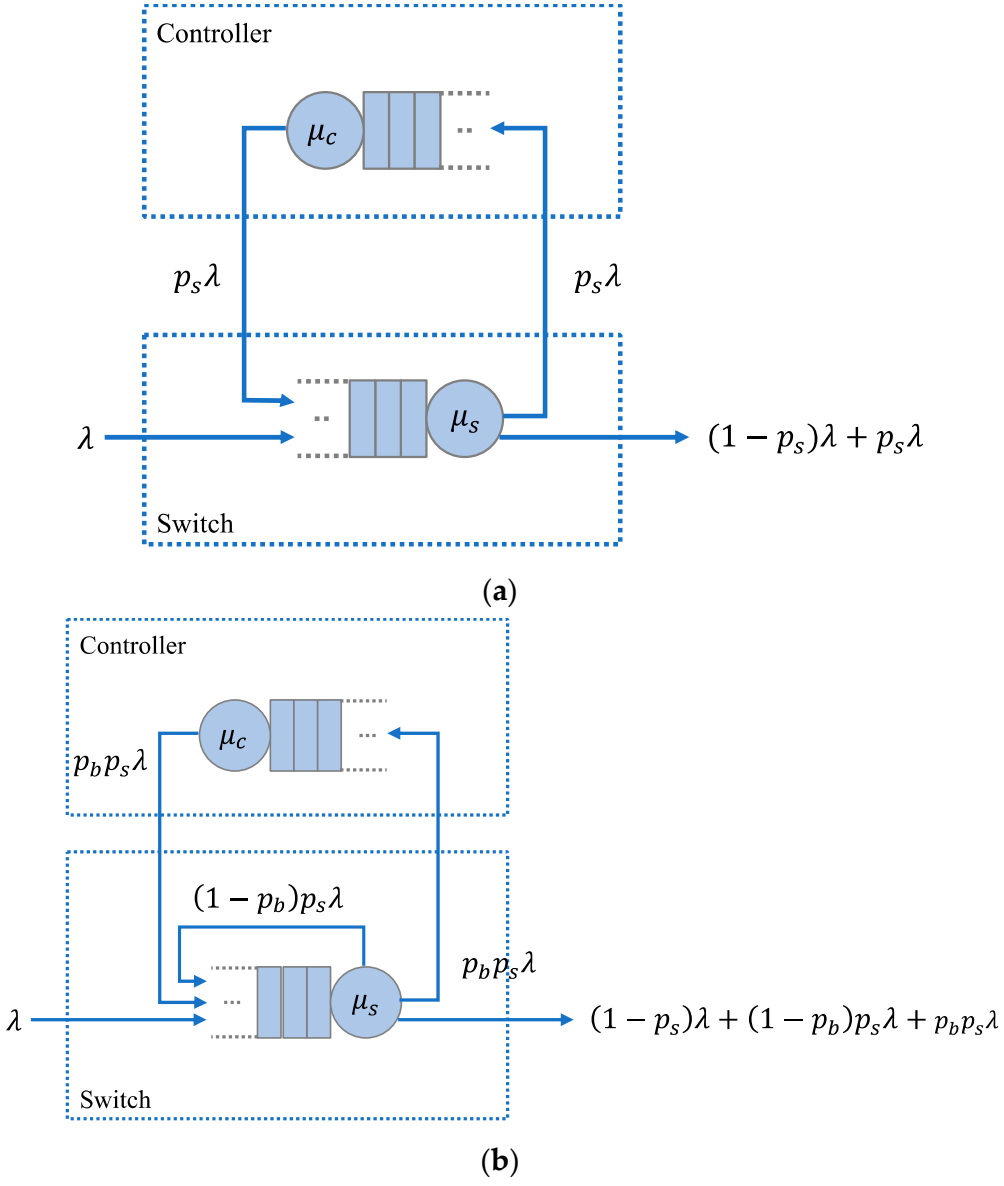


Figure 6. Queuing models of system. (**a**) Conventional SDN model, (**b**) SDN-TBM model.

Table 2. Variable notation.

Notation	Meaning
λ	Arrival rate of EP to the switch according to Poisson distribution
μ_s	Switch service rate
μ_c	Controller service rate
p_s	Probability that a packet is forwarded to the controller
p_b	Probability that a non-BP packets to the controller

We first discuss the scenario of an EP arriving at a switch. For the conventional SDN network, as shown in Figure 6a, we assume that λ denotes the arriving rate of an EP arriving at a switch according to a Poisson distribution, μ_s represents the service rate of the switch, μ_c denotes the service rate of the controller, and p_s represents the probability that a packet is forwarded to the controller. When an EP arrives at a switch and finds no flow entry matched in the flow table, it is forwarded to the controller $p_s\lambda$. Otherwise, the EP is forwarded to the next hop $(1-p_s)\lambda$.

For the SDN-TBM network, as depicted in Figure 6b, we assume that λ denotes the arriving rate of an EP arriving at the switch according to a Poisson distribution, μ_s denotes the service rate of the switch. We also let p_b represent the probability of forwarding the non-boring packets (BPs) to the controller, and μ_c denotes the service rate of the controller. When an EP, which is not a BP, arrives at a switch and finds no flow entry matching in the flow table, it is forwarded to the controller $p_b p_s \lambda$. Otherwise, the packet is processed by the BPP, and then the processed packet is sent directly to the same switch $(1-p_b)p_s\lambda$.

Next, we derive the average sojourn time of an EP for both models. Let T_s and T_b denote the sojourn time of the packet-in message in the switch and the BPP, respectively. T_s and T_b can be expressed as follows

$$T_s = \frac{1}{\mu_s - \lambda}, \tag{1}$$

$$T_b = \frac{1}{\mu_s - p_s\lambda}. \tag{2}$$

Let T_c^{sdn} and T_c^{tbm} denote the sojourn time of a packet in the controller in the SDN and SDN-TBM mechanisms, respectively. T_c^{sdn} and T_c^{tbm} can be expressed as follows:

$$T_c^{sdn} = \frac{1}{\mu_c - p_s\lambda}, \tag{3}$$

$$T_c^{tbm} = \frac{1}{\mu_c - p_b p_s\lambda}. \tag{4}$$

The total sojourn time W^{sdn} of a packet entering the SDN network and the total sojourn time W^{tbm} of a packet entering the SDN-TBM network can be respectively expressed as follows:

$$W^{sdn} = \begin{cases} T_s \quad with\ probability\ 1-p_s \\ T_s + T_c^{sdn} + T_s'\ with\ probability\ p_s \end{cases}, \tag{5}$$

$$W^{tbm} = \begin{cases} T_s \quad\quad with\ probability\ 1-p_s \\ T_s + T_b + T_s'\ with\ probability\ (1-p_b)p_s \\ T_s + T_b + T_c^{tbm} + T_s'\ with\ probability\ p_b p_s. \end{cases}, \tag{6}$$

Note that T_s' represents the sojourn time of a packet in the switch, where the packet is retransmitted to the same switch after processing by BPP. We assume that T_s' is equal to T_s. Therefore, we can calculate the mean of W^{sdn} and W^{tbm} according to the following expressions:

$$\begin{aligned} E\left[W^{sdn}\right] &= (1-p_s)E[T_s] + p_s\left(E[T_s] + E\left[T_c^{sdn}\right] + E[T_s']\right) \\ &= E[T_s] + p_s\left(E\left[T_c^{sdn}\right] + E[T_s']\right) \\ &\approx 2E[T_s] + p_s E\left[T_c^{sdn}\right], \end{aligned} \tag{7}$$

$$\begin{aligned} E\left[W^{tbm}\right] &= (1-p_s)E[T_s] + ((1-p_b)p_s)(E[T_s] + E[T_b] + E[T_s']) \\ &+ p_b p_s\left(E[T_s] + E[T_b] + E\left[T_c^{tbm}\right] + E[T_s']\right) \\ &= E[T_s] + ((1-p_b)p_s)E[T_b] + p_b p_s E\left[T_c^{tbm}\right] + p_s E[T_s'] \\ &\approx (1+p_s)E[T_s] + ((1-p_b)p_s)E[T_b] + p_b p_s E\left[T_c^{tbm}\right]. \end{aligned} \tag{8}$$

By substituting Equations (1)–(4) into Equations (7) and (8), we can obtain $E[W^{sdn}]$ and $E\left[W^{tbm}\right]$ as follows:

$$\left[W^{sdn}\right] = 2\left(\frac{1}{\mu_s - \lambda}\right) + p_s\left(\frac{1}{\mu_c - p_s\lambda}\right), \tag{9}$$

$$E\left[W^{tbm}\right] = (1+p_s)\left(\frac{1}{\mu_s - \lambda}\right) + ((1-p_b)p_s)\left(\frac{1}{\mu_s - p_s\lambda}\right) + p_b p_s\left(\frac{1}{\mu_c - p_b p_s\lambda}\right). \tag{10}$$

Next, we analyze the average sojourn time of a packet passing through multiple switches for both networks. For convenience, we do not consider the transmission time of the link because the transmission time of the link is extremely short. Only the processing time of the switch and the controller is considered. Figure 7 shows the operation flow of a packet arriving at conventional SDN and SDN-TBM networks. We assume that the number of switches on the transmission path of the packet from the source to the destination is k. For the conventional SDN network illustrated in Figure 7a, each switch operates independently, and the controller communicates with each switch individually when an unfamiliar packet is encountered. A switch communicates with the controller when it receives an unfamiliar packet. On the other hand, for the SDN-TBM model illustrated in Figure 7b, only the first and last switches on the transmission path will directly communicate with the controller as an unfamiliar packet is encountered. Therefore, the value of p_b is equal to $\frac{2}{k}$. The total average sojourn time of a packet passing through multiple switches for both networks, $E\left[W_f^{sdn}\right]$ *and* $E\left[W_f^{tbm}\right]$, is calculated as follows:

$$E\left[W_f^{sdn}\right] = \sum_{i=1}^{k}\left(2\left(\frac{1}{\mu_s^i - \lambda_f}\right) + p_s^i\left(\frac{1}{\mu_c - p_s^i\lambda_f}\right)\right), \tag{11}$$

$$E\left[W_f^{tbm}\right] = \sum_{i=1}^{k}\left(\left(1+p_s^i\right)\left(\frac{1}{\mu_s^i - \lambda_f}\right) + \left(\left(1-p_s^i\right)p_s^i\right)\left(\frac{1}{\mu_s^i - p_s^i\lambda_f}\right) + p_b^i p_s^i\left(\frac{1}{\mu_c - p_b^i p_s^i\lambda_f}\right)\right) \tag{12}$$

where k represents the total number of switches in the flow path, and λ_f represents one flow packet.

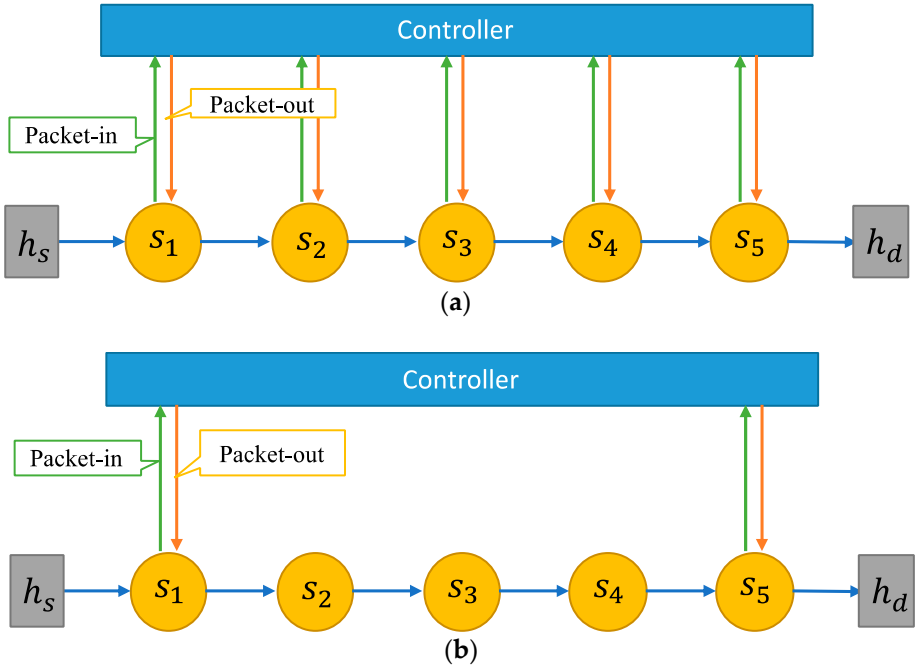


Figure 7. Procession flow of the first packet. (**a**) Conventional SDN model, (**b**) SDN-TBM model.

5. Numerical Analysis and Simulation

Based on the queueing models developed in the last section, we conduct two numerical experiments to investigate the effects of system parameters on the average packet sojourn time. We also conduct a simulation to verify the feasibility of the proposed SDN-TBM approach and compare it to the conventional SDN approach and JumpFlow approach in terms of the packet sojourn time and the number of packet-in messages. To validate the performance of the proposed SDN-TBM approach, we developed a simulator to support three modes, i.e., standard SDN, JumpFlow, and SDN-TBM. This simulator supports flow table comparison and timeout mechanisms for flow entries, as well as network environments with multiple switches. This simulator runs in Table 3 of the device specifications.

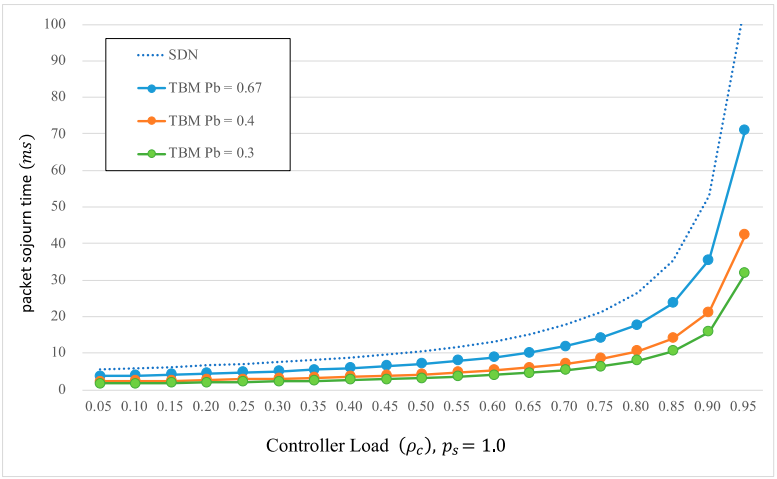
Table 3. The system environment.

The Development Environment of System	
OS	Ubuntu 18.04.3 LTS
CPU	Intel Xeon 2.3 GHz
RAM	16 GB
OpenvSwitch	v2.13, C++
Ryu Controller	Python
Simulator	PHP 7

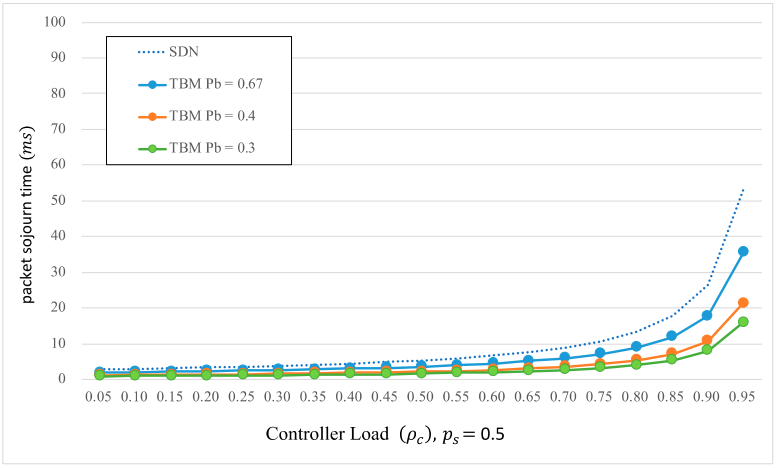
5.1. The Effect of p_b and p_s on the Sojourn Time in a Single Switch

The first numerical experiment investigates the effects of p_b and p_s on the packet sojourn time in a single switch. Noted that p_s represents the probability that a packet is forwarded to the controller, and p_b represents the probability of forwarding the non-boring packets (BPs) to the controller. The work of [2,14] indicated that the distribution of the number of switches on a path in a data center is {3, 5, 7}. Therefore, we vary the values of p_b as {0.67, 0.4, 0.3}. We also vary the values of p_s as {1.0, 0.5, 0.2}. Because the load ratio of the controller will affect the packet sojourn time, we vary the controller load ratio from 0.05 to 0.95 by an incremental 0.05. We compare the proposed SDN-TBM approach to the conventional SDN approach proposed by Pranata et al. [15]. The numerical results are shown in Figure 8a–c. From Figure 8, one can see that for a fixed value of p_s, the average

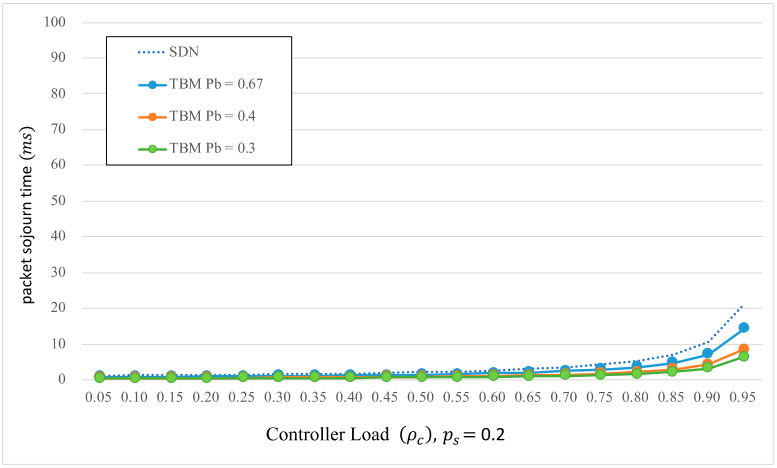
packet sojourn time increases as the controller load ratio increases. For a fixed p_s and a fixed controller load ratio, the average packet sojourn time in a switch decreases as the value of p_b decreases. One also can find that, from Figure 8, our approach outperforms the SDN approach. For example, in Figure 8a, when the controller load ratio approaches 95%, the average packet sojourn time drops significantly by approximately 70%, 60%, and 33%, and the average packet sojourn time is approximately 54%.



(**a**)



(**b**)



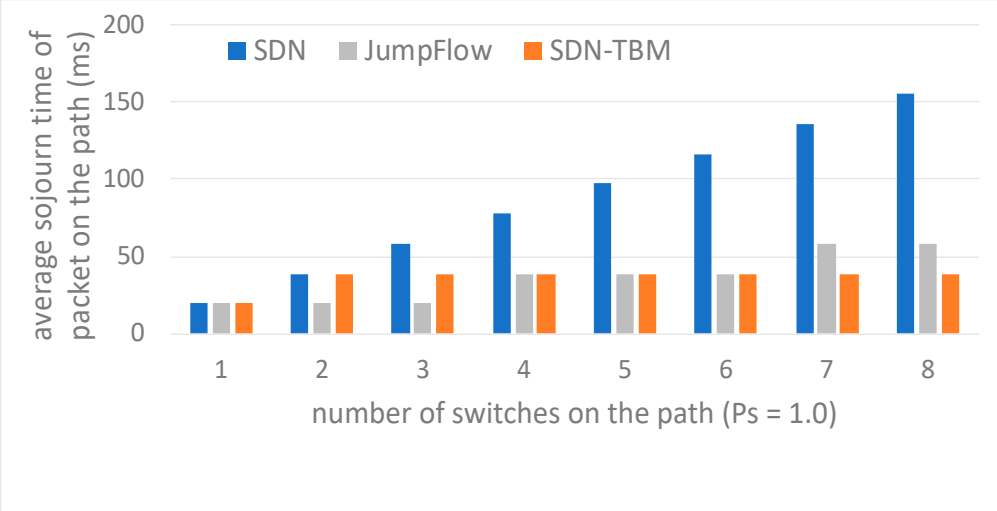
(**c**)

Figure 8. Effect of controller load on packet sojourn time. (**a**) $p_s = 1.0$, (**b**) $p_s = 0.5$, (**c**) $p_s = 0.2$.

5.2. The Effect of p_s on the Average Packet Sojourn time on a Path

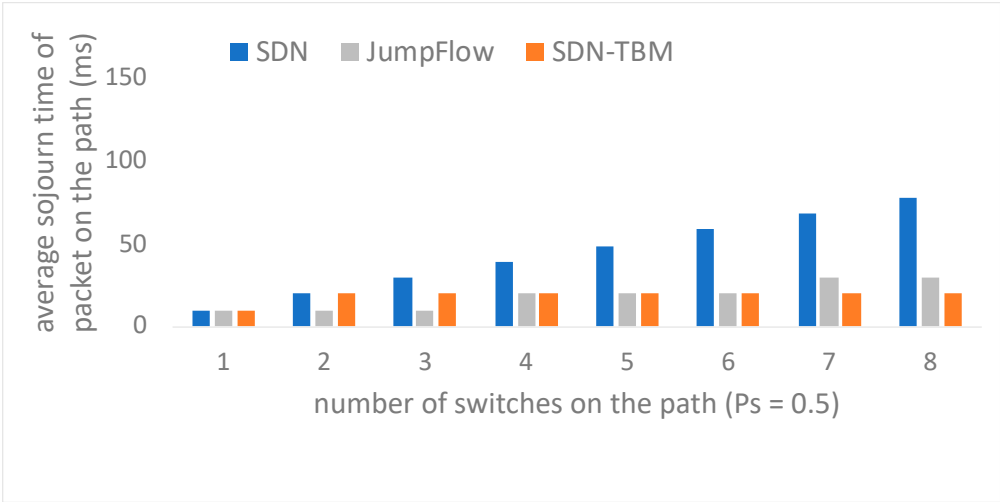
The second numerical experiment studies the effect of p_s on the average packet sojourn time on a path. In this scenario, we assume that there are several transmission paths with different lengths, and a large number of flows are transmitted on the same path. For those flows on the same path, we vary the values of p_s as 1, 0.5, and 0.2. We also vary the number of switches on a path from 1 to 8 by an incremental 1. We compare the SDN-TBM approach to conventional SDN approach and JumpFlow approach [18]. For SDN and SDN-TBM approaches, the switches on the flow path operate independently, and the effect of packet-in messages on the controller is treated independently as well. In JumpFlow approach, by referring [30], we assume that each switch has 24 ports. In addition, because the size of the VLAN Identifier (VID) of a packet is 12 bits, only the next two hops can be recorded in VID.

The numerical results are shown in Figure 9. The figure shows that both SDN-TBM and JumpFlow approaches can effectively reduce the sojourn time of packet on a path. The JumpFlow approach performs better than the SDN-TBM approach when the number of switches on a path is less than 3. When the number of switches on a path is in the range from 4 to 6, the average sojourn time of packet in our approach is close to the JumpFlow approach. When the number of switches on a path exceeds 6, our approach is superior to the JumpFlow approach. Although the JumpFlow approach performs better than or nearly as well as our approach when the number of switches on a path is less than or equal to 6, but the JumpFlow approach has many limitations in the SDN environment. First, with JumpFlow approach, the SDN network needs to support the VLAN protocol. Second, the JumpFlow approach can only forward packets to the designated port and cannot forward packets to different routes based on the packet information. In addition, with JumpFlow approach, the number of ports in the switch affects the largest number of hops. Recording fewer hops in the VID will increase the number of requests to the controller, which increases the packet sojourn time. In addition, the JumpFlow approach cannot solve the challenges in the reactive mode of the SDN network, which are mentioned in [21]. Unlike the JumpFlow approach, our approach can fully support the features of the standard SDN and does not need to install other specific communication protocols.

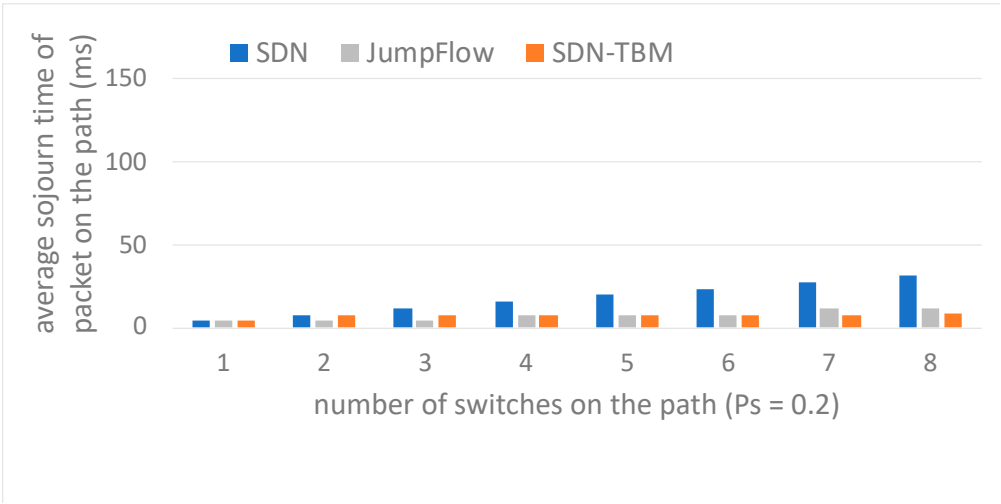


(**a**)

Figure 9. *Cont.*



(**b**)



(**c**)

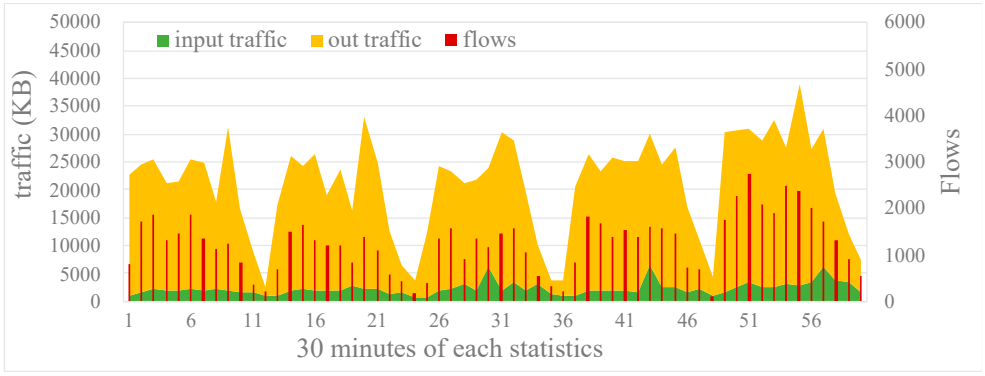
Figure 9. Comparison of sojourn time for flow forwarding on a path. (**a**) $p_s = 1.0$, (**b**) $p_s = 0.5$, (**c**) $p_s = 0.2$.

5.3. Simulation and Results

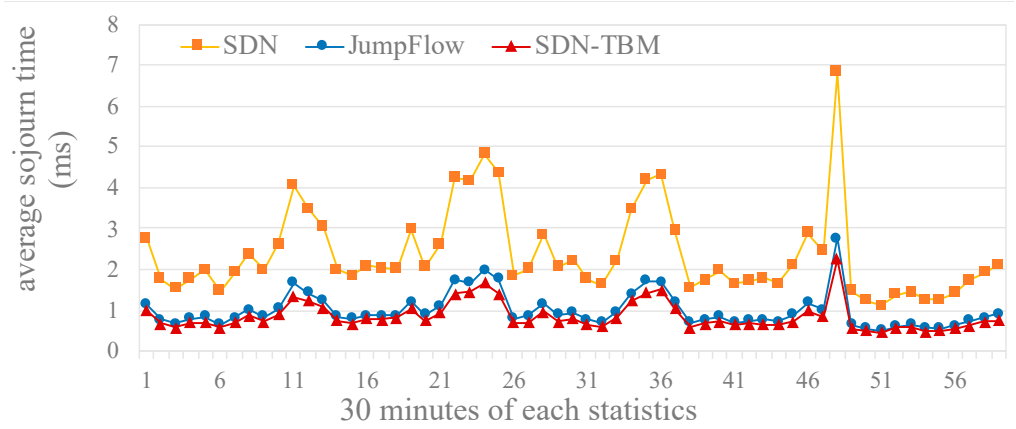
In the new generation of cloud application services, microservices are increasingly used to provide network resources. Most of the RESTful API protocols [31] are used in business applications and most of the MQTT protocols are used in IoT applications. Francesco et al. proposed solutions for the problems related to social internet of things (SIoT) and multi-internet of things (MIoT) [10]. The data characteristics for such a large number of small data streams require dynamic microservice migration and scaling. Network management in data centers can be done using the SDN-TBM mechanism to quickly adjust network paths and reduce the service latency time.

We collected real data for simulation in a set of formal commercial information systems. This dataset uses the RESTful API protocol and is characterized by a large number of flows and consists of fewer packets, which is a common traffic type for most applications today. Such a characteristic can be precisely controlled by the reactive mode of the SDN. The data period is two months. We record the traffic flows data every 30 min, and the contents include input traffic, output traffic, and flows. For convenience, we only show the 60 pieces of collected traffic flow data in Figure 10a. The properties of collected traffic flow and system parameters are shown in Table 4. Comparisons of the average sojourn time and the average number of packet-in messages among the SDN, JumpFlow, and SDN-TBM

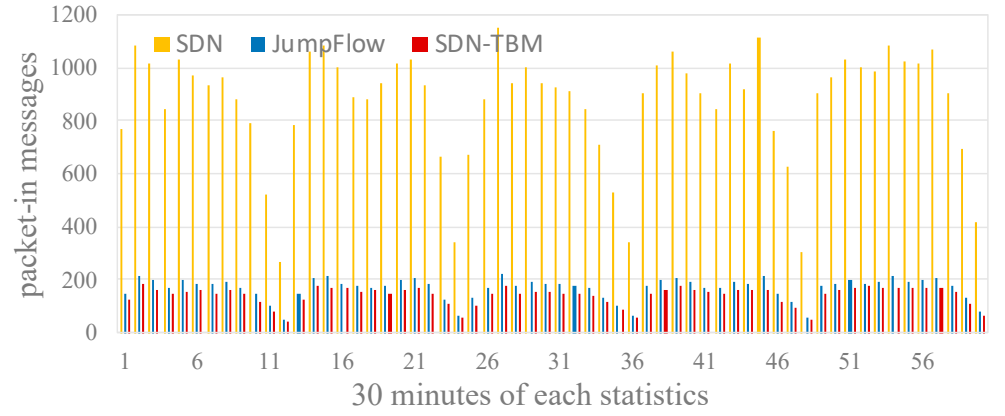
approaches are shown in Figure 10b,c, respectively. In Figure 10b, one can observe that the prominent points appeared when the number of unfamiliar flows was greater than the number of regular flows. From the Figure 10b,c, we can find that the average sojourn time and the average number of packet-in messages in the SDN-TBM approach are lower than the other approaches.



(**a**)



(**b**)



(**c**)

Figure 10. Comparison of RESTful data simulations for SDN, JumpFlow and SDN-TBM models. (**a**) traffic flow data, (**b**) average sojourn time, (**c**) packet-in messages.

Table 4. Simulation properties.

Property	Value
Number of records	35,533,9161
Number of flows	1,330,909
Range of flow size	1–18,000,625 bytes
Average flow size	10,975.37 bytes
Range of packets in a flow	1–231,279 packets
Average packets in a flow	20.97 packets
Length of path	3–8 hops
Time out of flow entry	30 min

We summarize the influence of unfamiliar flows on the sojourn time of flows for the three approaches in Figure 11. The values of the SDN-TBM approach are close because any length of path sends only two packet-in messages to the controller. In the JumpFlow approach, because different path lengths send different packet-in messages to the controller, the sojourn time values are diverse. Figure 12, summarizing the influence of unfamiliar flows on the number of packet-in messages, shows a trend similar to that of the influence of unfamiliar flows on the sojourn time. Based on the simulation results, the average packet sojourn time of the SDN-TBM approach outperforms the JumpFlow approach and the conventional SDN approach as 15.81% and approximately 67.16%, respectively. In addition, the number of packet-in messages in the SDN-TBM approach outperforms the JumpFlow approach and the conventional SDN approach as 15.97% and 83.69%, respectively.

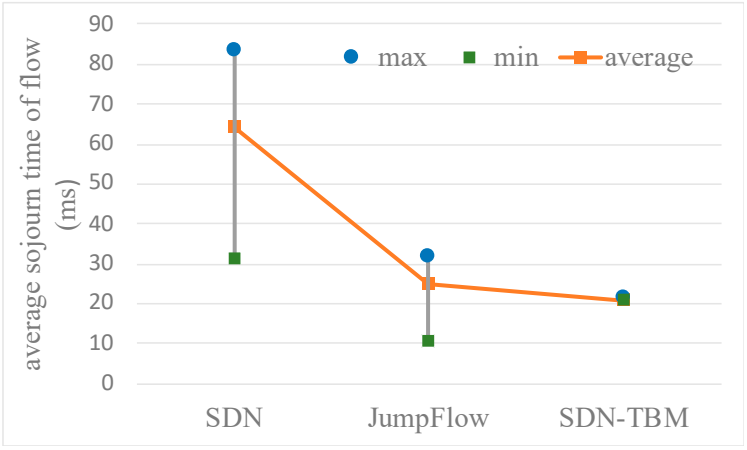


Figure 11. Impact of unfamiliar flows on the average sojourn time.

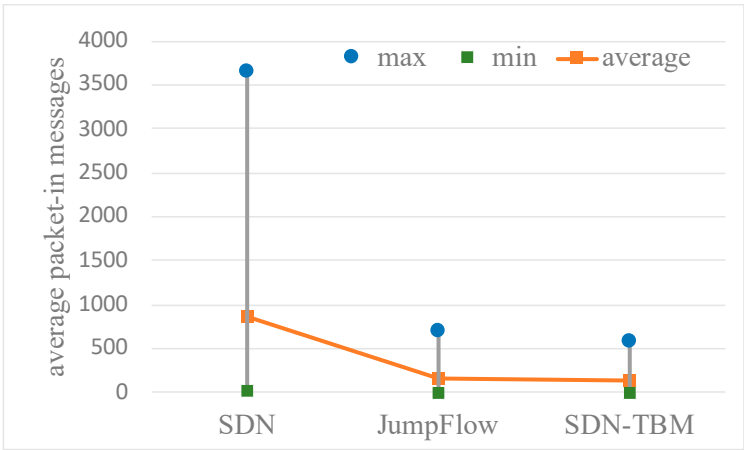


Figure 12. Impact of unfamiliar flows on the average of the packet-in message.

6. Conclusions and Future Work

We proposed an SDN-TBM mechanism to reduce the load on the controller in the reactive mode of SDN. In the SDN-TBM mechanism, only the starting and destination switches for each application flow are connected to the controller; all intermediate switches on the routing path simply carry the information required for forwarding purposes. Consequently, the number of packet-in messages is reduced; this reduction results in reduced load on the controller. We also developed a queuing model to analyze the SDN-TBM process and performed two numerical experiments to study the effects of changes in parameter values on the average sojourn time of a packet and on the average sojourn time of a packet on a path. A simulation was also carried out. The numerical results revealed that the SDN-TBM model can reduce the average packet sojourn time by up to 70% compared to the conventional SDN. Additionally, when the controller load increased to 95%, the average sojourn time was reduced by approximately 54%. Moreover, the SDN-TBM model outperforms the JumpFlow model when the number of switches on the path exceeds 6. The simulation results showed similar trends. However, the JumpFlow model only performs routing functions. The SDN-TBM approach not only performs routing functions but also can deploy the flow entries of the switches on the path.

Author Contributions: M.-T.K., S.-J.K., H.-W.T. and F.-M.C. contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hakiri, A.; Gokhale, A.; Berthou, P.; Schmidt, D.C.; Gayraud, T. Software-Defined Networking: Challenges and research opportunities for Future Internet. *Comput. Netw.* **2014**, *75*, 453–471. [CrossRef]
2. Lara, A.; Kolasani, A.; Ramamurthy, B. Network Innovation using OpenFlow: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 493–512. [CrossRef]
3. Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P. The design and implementation of open vswitch. In Proceedings of the 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), Oakland, CA, USA, 4–6 May 2015; pp. 117–130.
4. Sattar, D.; Matrawy, A. An empirical model of packet processing delay of the Open vSwitch. In Proceedings of the 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, Canada, 10–13 October 2017; pp. 1–6. [CrossRef]
5. Chen, Z.; Wu, Y.; Ge, J.; Yuepeng, E. A New Lookup Model for Multiple Flow Tables of Open Flow with Implementation and Optimization Considerations. In Proceedings of the 2014 IEEE International Conference on Computer and Information Technology (CIT), Xi'an, China, 11–13 September 2014; pp. 528–532. [CrossRef]
6. Hatami, R.; Bahramgiri, H. High-performance architecture for flow-table lookup in SDN on FPGA. *J. Supercomput.* **2019**, *75*, 384–399. [CrossRef]
7. Fernandez, M.P. Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive. In Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013; pp. 1009–1016. [CrossRef]
8. Xiao, X.; Hannan, A.; Bailey, B.; Ni, L.M. Traffic engineering with MPLS in the Internet. *IEEE Netw.* **2000**, *14*, 28–33. [CrossRef]
9. Filsfils, C.; Nainar, N.K.; Pignataro, C.; Cardona, J.C.; Francois, P. The Segment Routing Architecture. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015; pp. 1–6. [CrossRef]
10. Cauteruccio, F.; Cinelli, L.; Fortino, G.; Savaglio, C.; Terracina, G.; Ursino, D.; Virgili, L. An approach to compute the scope of a social object in a Multi-IoT scenario. *Pervasive Mob. Comput.* **2020**, *67*, 101223. [CrossRef]
11. Praveena, V.; Praveena, V.; Ponnusamy, C.; Ihsan, A.; Alroobaea, R.; Yahya, S.; Raza, M.A. Optimal Deep Reinforcement Learning for Intrusion Detection in UAVs. *Comput. Mater. Contin.* **2022**, *70*, 2639–2653. [CrossRef]
12. Sudar, K.M.; Beulah, M.; Deepalakshmi, P.; Nagaraj, P.; Chinnasamy, P. Detection of Distributed Denial of Service Attacks in SDN using Machine learning techniques. In Proceedings of the 2021 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 27–29 January 2021.

13. En.wikipedia.org. Tunnel Boring Machine. 2020. Available online: https://en.wikipedia.org/wiki/Tunnel_boring_machine (accessed on 19 May 2021).
14. Ma, Y.-W.; Chen, J.-L.; Tsai, Y.-H.; Cheng, K.-H.; Hung, W.-C. Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking. *Wirel. Pers. Commun.* **2017**, *94*, 3549–3574. [CrossRef]
15. Neghabi, A.A.; Jafari Navimipour, N.; Hosseinzadeh, M.; Rezaee, A. Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature. *IEEE Access* **2018**, *6*, 14159–14178. [CrossRef]
16. MacDavid, R.; Birkner, R.; Rottenstreich, O.; Gupta, A.; Feamster, N.; Rexford, J. Concise Encoding of Flow Attributes in SDN Switches. In Proceedings of the SOSR '17: Symposium on SDN Research, Santa Clara, CA, USA, 3–4 April 2017; ACM: New York, NY, USA, 2017; pp. 48–60. [CrossRef]
17. Guo, Z.; Xu, Y.; Cello, M.; Zhang, J.; Wang, Z.; Liu, M.; Chao, H.J. JumpFlow: Reducing flow table usage in software-defined networks. *Comput. Netw.* **2015**, *92*, 300–315. [CrossRef]
18. Jia, X.; Li, Q.; Jiang, Y.; Guo, Z.; Sun, J. A low overhead flow-holding algorithm in software-defined networks. *Comput. Netw.* **2017**, *124*, 170–180. [CrossRef]
19. Pranata, A.A.; Jun, T.S.; Kim, D.S. Overhead reduction scheme for SDN-based Data Center Networks. *Comput. Stand. Interfaces* **2019**, *63*, 1–15. [CrossRef]
20. Pranata, A.A.; Lee, J.M.; Kim, D.S. OpenFlow Controller-Switch Communication Overhead Reduction Scheme on Industrial Data Center Networks. In Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 4–7 September 2018; pp. 243–250. [CrossRef]
21. Hu, C.; Hou, K.; Li, H.; Wang, R.; Zheng, P.; Zhang, P.; Wang, H. SoftRing: Taming the reactive model for software defined networks. In Proceedings of the 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, Canada, 10–13 October 2017; pp. 1–10. [CrossRef]
22. Chilwan, A.; Mahmood, K.N.; Østerb, O.; Jarschel, M. On Modeling Controller-Switch Interaction in Openflow Based SDNS. *IJCNC* **2014**, *6*, 137–150. [CrossRef]
23. Goto, Y.; Ng, B.; Seah, W.K.G.; Takahashi, Y. Queueing analysis of software defined network with realistic OpenFlow–based switch model. *Comput. Netw.* **2019**, *164*, 106892. [CrossRef]
24. Jarschel, M.; Oechsner, S.; Schlosser, D.; Pries, R.; Goll, S.; Tran-Gia, P. Modeling and performance evaluation of an OpenFlow architecture. In Proceedings of the 2011 23rd International Teletraffic Congress (ITC), Cracow, Poland, 4–7 September 2012; pp. 1–7.
25. Mahmood, K.; Chilwan, A.; Østerb, O.N.; Jarschel, M. On the Modeling of Open Flowbased SDNS: The Single Node Case. In Proceedings of the Third International Conference on Advanced Information Technologies & Applications, Zurich, Switzerland, 2–4 January 2014; Academy & Industry Research Collaboration Center (AIRCC): New Delhi, India, 2014; pp. 207–217. [CrossRef]
26. Miao, W.; Min, G.; Wu, Y.; Wang, H. Performance Modelling of Preemption-Based Packet Scheduling for Data Plane in Software Defined Networks. In Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), Chengdu, China, 19–21 December 2015; pp. 60–65. [CrossRef]
27. Xiong, B.; Peng, X.; Zhao, J. A Concise Queuing Model for Controller Performance in Software-Defined Networks. *JCP* **2016**, *11*, 232–237. [CrossRef]
28. Singh, D.; Ng, B.; Lai, Y.-C.; Lin, Y.-D.; Seah, W.K.G. Modelling Software-Defined Networking: Software and hardware switches. *J. Netw. Comput. Appl.* **2018**, *122*, 24–36. [CrossRef]
29. Xiong, B.; Yang, K.; Zhao, J.; Li, W.; Li, K. Performance evaluation of OpenFlow-based software-defined networks based on queueing model. *Comput. Netw.* **2016**, *102*, 172–185. [CrossRef]
30. Farrington, N.; Rubow, E.; Vahdat, A. Data Center Switch Architecture in the Age of Merchant Silicon. In Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects, New York, NY, USA, 25–27 August 2009; pp. 93–102. [CrossRef]
31. Rodriguez, A. Restful web services: The basics. *IBM Dev.* **2008**, *33*, 18.