Priority-based Flow Control for Dynamic and Reliable Flow Management in SDN

Bong-Hwan Oh, Serdar Vural, *Member, IEEE,* Ning Wang, *Senior Member, IEEE,* and Rahim Tafazolli, *Senior Member, IEEE,*

***Abstract*—Software-Defined Networking (SDN) is a promising paradigm of computer networks, offering a programmable and centralised network architecture. However, although such a technology supports the ability to dynamically handle network traffic based on real-time and flexible traffic control, SDN-based networks can be vulnerable to dynamic change of flow control rules, which causes transmission disruption and packet loss in SDN hardware switches. This problem can be critical because the interruption and packet loss in SDN switches can bring additional performance degradation for SDN-controlled traffic flows in the data plane. In this paper, we propose a novel robust flow control mechanism referred to as Priority-based Flow Control (PFC) for dynamic but disruption-free flow management when it is necessary to change flow control rules on the fly. PFC minimizes the complexity of flow modification process in SDN switches by temporarily adapting the priority of flow rules in order to substantially reduce the time spent on control-plane processing during run-time. Measurement results show that PFC is able to successfully prevent transmission disruption and packet loss events caused by traffic path changes, thus offering dynamic and lossless traffic control for SDN switches.**

***Index Terms*—Software-Defined Networking, dynamic traffic control, reliability.**

I. Introduction

IMPROVING network utilisation is one of the most challenging issues in network management. The demand to accommodate a diverse range of network services requires significantly over-provisioned network capacity as today's common practice. Such a strategy brings a high deployment cost, even though physical network capacity is still increasing with the advancement of network devices. As a remedy, one way to utilise network resources in a more cost-efficient manner is the use of dynamic traffic management, as available network resources and network conditions are typically highly dynamic.

Software Defined Networking (SDN) [1,2] is a promising network paradigm enabling programmable network platforms, which circumvents the limitation of traditional networks with rigidly coupled control/data planes. The programmability feature of SDN enables network operators to flexibly manage their network resources by decoupling the control plane from the underlying hardware (i.e. network devices such as routers and switches) and by instantly changing data plane traffic forwarding rules. OpenFlow [3] currently is the de facto signaling protocol between the control and data planes which is used to program SDN switches. With OpenFlow, an SDN controller can retrieve complete network view by using its inherent monitoring mechanism. Equipped with the programmable and (logically) centralised control plane offered by SDN, it is easy to monitor network conditions and accordingly change network resource allocation on the fly, which makes SDN one of the key enabling technologies for the new generation of networks.

One of the basic requirements for enabling dynamic reconfiguration of traffic forwarding rules is to ensure disruption-freedom of ongoing traffic flows. Ongoing flows should not suffer from any disruption when a traffic forwarding rule is changed via SDN control. Typical examples for changing flow control rules during run-time can include: online traffic load balancing, detouring from a specific route, and dynamic service chaining into a live network. However, almost all IP networks only support "best effort" services, and it is assumed that network performance in IP networks is normally affected by network congestions and anomalies. Consequently, most transport protocols only consider network congestion as a factor of traffic degradation, when adjusting end-to-end traffic behaviour towards improving flow reliability. Thus, the effect of dynamic traffic control operations should also be taken into account in order to prevent disruptions in transport operations, as well as network performance in general, when changes are made to traffic forwarding rules in real time.

Unfortunately, we discovered that traffic loss can occur in current SDN hardware switches when the forwarding rule being applied to a currently active traffic flow is modified during ongoing traffic transmission. This is attributed to the processing latency, which is the time needed to modify the forwarding rule in a hardware switch. This processing latency can cause transmission disruption, which also leads to packet loss for a transient period of time. This type of traffic loss can bring considerable performance degradation in an SDN network, because the incurred packet loss not only causes the deterioration of the network performance but also triggers the packet loss recovery procedure which in turn adversely impacts the end-to-end transmission performance at the transport layer or the application layer.

In this paper, a priority-based flow control (PFC) mechanism is presented in order to address the traffic disruption problem caused by dynamic changes in flow rules in SDN hardware switches. In a situation where a flow rule needs to be changed

This project is partially funded by the 5G Innovation Centre (5GIC), and EU SoftFIRE project, Grant Number 687860.

The authors are with the 5G Innovation Centre (5GIC), Institute for Communication Systems (ICS), James Clerk Maxwell Building, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom. (e-mail: {b.oh, s.vural, n.wang, r.tafazolli}@surrey.ac.uk).

during on-going traffic transmission, PFC minimises the flow modification complexity by temporarily adapting the priorities of flow rules before executing the actual change, which can substantially reduce both transmission disruption time and traffic loss volume in SDN hardware switches. Furthermore, PFC can be applied without any hardware modification, as it is deployed in SDN controllers as a software solution. According to our SDN testbed experiments, any potential traffic loss due to a run-time change in flow rules can be eliminated by applying PFC. The main contributions of this paper are as follows:

- This work identifies and reports on a new transmission disruption risk in SDN networks. This problem can be caused when a run-time flow rule change is executed by SDN, and can also result in packet loss events during the transition period until the new flow rule is activated. Such run-time changes in flow rules in the middle of on-going traffic sessions may take place for the purpose of load-balancing or planned maintenance where one or multiple active traffic flows may need to have their current rules changed.
- A novel priority-based flow control mechanism is proposed in order to prevent the transmission disruption by minimising the adverse effects of the SDN flow modification process in hardware switches. PFC is a controller-based solution and hence can easily be applied to the currently deployed SDN switches without any modification in SDN switches. Through extensive experiments based on our SDN testbed platform, we show that the proposed PFC scheme is able to completely avoid even transient packet loss and performance deterioration at the transport-layer. By this means, the proposed scheme practically enables seamless run-time change of flow control rules without any impact network performance during run-time.

The rest of the paper is organised as follows. Section II briefly explains traffic flow operations in OpenFlow-based SDN networks, and describes the newly observed transmission disruption problem that is caused by processing delays in hardware SDN switches. This is followed by the description of the SDN-based PFC framework and its operation in Section III. Section IV presents the overall performance evaluation of PFC, according to the identified performance metrics including packet loss statistics, traffic throughput and also the overhead introduced by the PFC scheme. Finally, Section V concludes the paper.

II. Background and Motivation

In an OpenFlow-based SDN architecture, a flow is the basic unit for network traffic which can be controlled by a *flow rule*. A typical flow rule consists of: a set of *flow match fields*, a *flow priority*, and a *flow action set*.

- Flow match fields are used to distinguish different flows, and consist of identifiers for ingress port, metadata, and packet header information (such as source IP, destination IP, source port, destination port, protocol type, etc).
- Flow priority determines the order of execution of the flow rules.

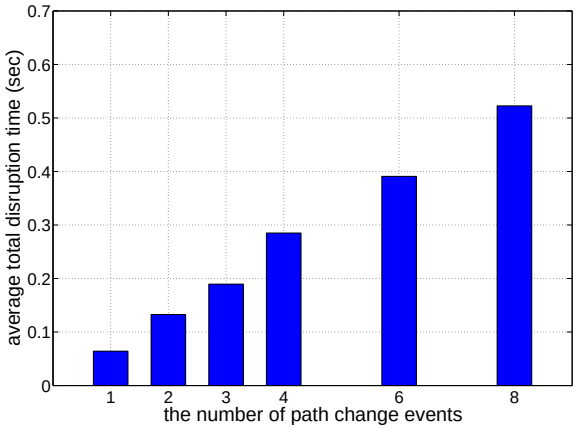

Fig. 1. Transmission disruption problem in SDN.

- Flow action set is a set of actions, where an action is an operation to either forward a flow to a port (or a set of ports) or modify the packets of a flow. The flow action set makes it possible to apply multiple flow action sets to the same traffic flow.

A flow rule is also typically called a *flow entry*, as each flow rule is simply an entry in a flow table of an SDN switch. The overall flow handling process is as follows. When data packets arrive at an SDN switch, a flow-match process is firstly performed at the switch flow table, i.e. the SDN switch searches a set of flow rules that match the incoming packets of this traffic flow. If there are relevant flow rules, the rule with the highest priority is selected, and then the corresponding flow action set is enforced. If there is no matching flow rule, a *table miss operation is performed.*

Besides flow-based traffic handling, one of the core features of SDN is that the flow handling process in SDN switches can be *dynamically* managed by the SDN controller. Flow rules can be easily constructed in the controller, and are dynamically manageable by adding, modifying, or deleting commands through the controller. Consequently, traffic forwarding behaviour can be dynamically changed by the SDN controller depending on network conditions and policies, such as load balancing, fast reroute and service chaining.

Although the OpenFlow SDN architecture supports dynamic traffic management, a stability problem of transient deterioration of traffic performance (e.g. packet loss) can be caused by the dynamic change of flow control rules in SDN. As an example of such a problem, Fig. 1 illustrates the average total disruption time caused by a number of path change events performed by an SDN controller, which replaces the flow rule for an ongoing traffic flow with another flow rule during packet transmission. The measurement environment is presented in Fig. 6, and the details are described in Section IV.A. In the rest of the paper, a flow rule which has been defined and deployed in SDN switches to match a specific ongoing traffic flow is referred to as the ***c**urrent **m**atching flow rule*, denoted by ($flow_{cm}$). In this measurement, the path change events are generated by deleting $flow_{cm}$ during packet forwarding. All the path change events are applied to

the same traffic during packet transmission and the number of path change events varies between 1 and 8. In case of the multiple path change events, the traffic route periodically fluctuates between path 1 and path 2 which are shown in Fig. 6. Performance results show that total traffic disruption time grows with increases in the number of path change events. This is a typical consequence of traffic disruption whenever the current matching flow rule $flow_{cm}$ is modified, and therefore causing packet loss.

The transmission disruption in this example is due to the following reason. When a *delete* command for $flow_{cm}$ arrives at an SDN switch, the SDN switch deletes $flow_{cm}$ and applies the ***n**ext **m**atching flow rule* ($flow_{nm}$) to the current traffic. A new matching flow rule ($flow_{nm}$) is to replace $flow_{cm}$ to serve the current traffic after $flow_{cm}$ is deleted. However, this operation requires some processing at the hardware switch (i.e. TCAM reordering followed by the contention for the limited bus bandwidth [4]) to enforce $flow_{nm}$ to the current flow. This processing operation can also be aggravated by any non-optimal implementation [4,5] or an inflexible switch architecture [13]. When packets of the current traffic arrive at the SDN switch during such processing operations, since the previously matching flow rule ($flow_{cm}$) has already been deactivated, packet loss occurs due to the missing flow rule for that transient time period [5]. This phenomenon can be critical as this is a new type of traffic disruption caused by on-the-fly flow rule change in SDN, which is not currently handled by SDN switches.

Several research studies [4-12] have revealed that some delay is caused by handling control messages or flow rules in SDN hardware switches. The authors of [6] analyzed a flow-setup latency as well as the latency incurred by gathering flow statistics due to the size of the flow table. In [7], control plane performance was examined with the focus on processing delay in SDN switches, which is attributed to flow control operations and gathering flow statistics. Another study [8] focused on the query completion time in SDN and the processing time for the first packet of a new flow rule and a flow modification event. In [9], it was observed that control plane performance can vary and might become unstable due to the control plane processing time, and also the frequency of flow table updates in hardware OpenFlow switches. Although these studies have focused on quantifying the control plane performance in SDN switches, the effect of the changes in packet forwarding rules was not specifically quantified and studied. Although the effect of control plane latency for packet forwarding was studied in [4] and [5], they only focus on performance measurement, without providing a solution to alleviate such an issue.

Several schemes have also been proposed for efficient packet forwarding in SDN [10, 11], and reliability in the SDN control plane [12]. The study in [10] focused on flow table management and [11] proposed switch-based solutions that require hardware modification in SDN switches. In [12], the reliability issues in the SDN control plane were mainly investigated as part of a management framework. Although new SDN switch architectures such as RMT [13] and P4 [14] can address this problem by improving SDN switch performances, they cannot be directly applied to off-the-shelf SDN switches.

The goal of this study is therefore to propose a robust flow control mechanism based on flow priority, in order to ensure dynamic but disruption-free traffic control in SDN. The proposed method is able to minimise the adverse effects of the latency caused by processing operations when current operational flow rules are modified. The proposed mechanism substantially reduces the traffic loss caused by the delay in transitioning from one flow rule to another. Since the mechanism is fully software-based, and is applied at the SDN controller, it can be easily applied without any hardware modification in SDN switches.

III. Priority-based Flow Control (PFC)

Unlike traditional packet forwarding methods, a flow rule for packet forwarding in SDN is determined by not only the flow *match* fields but also the flow *priority* field, which affects packet forwarding performance. Using the flow priority field makes it possible to easily locate the next matching flow rule ($flow_{nm}$) and select it as the prospective active flow rule before actual modification of the current operational flow rule ($flow_{cm}$). In doing so, the latency in locating $flow_{nm}$ can be reduced by effective assignment of priority values to flow rules and reducing this latency can also prevent the traffic disruption while the flow modification process takes place. In light of this observation, the main idea of PFC is to temporarily adapt the priority of flow rules in order to minimise disruptions caused by processing delay. PFC searches $flow_{nm}$ and temporarily assigns a higher flow priority level to it in the flow table, which can then quickly service a matching traffic, effectively avoiding packet drops. PFC is performed as a flow control functionality in the SDN controller. Next, the SDN architecture with PFC is presented.

A. SDN Architecture and Framework with PFC

In SDN, the network infrastructure consists of a set of programmable network devices that perform packet forwarding in the data plane, as shown in Fig. 2. These network devices can communicate with an SDN controller on its southbound API (generally OpenFlow implementation). The SDN controller provides its northbound API to applications; this API is based on Representational State Transfer (REST) and Remote Procedure Call (RPC). The controller has a set of core functions which can be classified into two types: (i) *a network abstraction function* which provides global information on the network, collected by modules that perform monitoring of network conditions and network topology, and (ii) *an interpretation function* which translates flow commands submitted to the controller via its northbound API into corresponding southbound API (OpenFlow) commands. Thanks to these capabilities provided by the SDN controller, various network applications can be easily implemented on top of the controller, without directly interacting with the physical hardware.

In this study, PFC is proposed as a basic function in the SDN controller, which acts as an interpretation function, as shown in Fig. 2. Applications or other functions in the controller

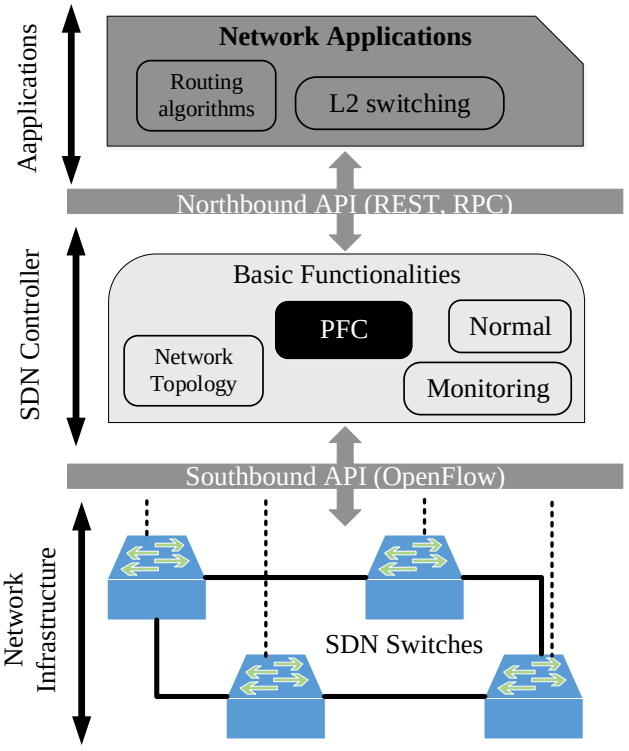

Fig. 2. SDN architecture and framework.

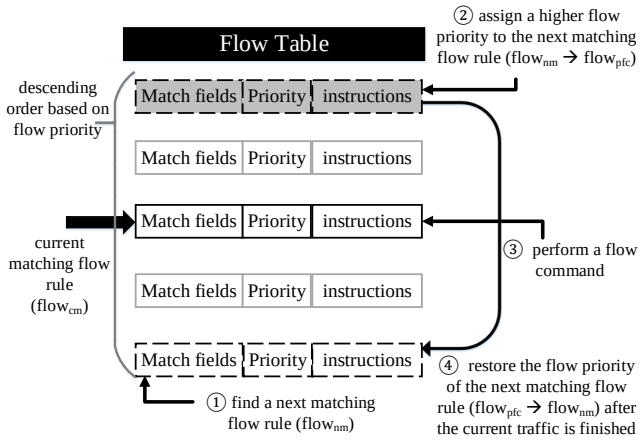

Fig. 3. General operation of PFC.

can still choose between PFC and the default interpretation function of the controller (tagged as "normal" in the figure). When the default interpretation function is in use, it simply translates flow commands into the southbound API (i.e. flow command translation). If applications (or other functions of the SDN controller) select PFC instead, PFC then performs flow command translation and additionally applies priority-based flow control. Note that PFC is also applicable to advanced SDN switch architectures, such as RMT [13], since it is a controller-based solution, and can be operated regardless of the particular SDN switch architecture. Furthermore, PFC may have more flexibility in evolved control protocols, such as P4 [14]; this extension is out of scope in this paper, and needs further study.

B. PFC Operation

Figure 3 shows the general operation of PFC. In this example, a specific network traffic is served by its current matching flow rule ($flow_{cm}$). When a *flow_delete* (or *flow_mod*) command is submitted in order to delete (or to modify) $flow_{cm}$, the following events take place: (1) PFC first searches the next matching flow rule ($flow_{nm}$) in the flow tables. If there is a $flow_{nm}$, then (2) PFC temporarily assigns a higher flow priority to $flow_{nm}$ before performing the *flow_delete* (or *flow_mod*) command. Note that this assigned priority value should be higher than that of $flow_{cm}$. The flow rule which has a temporary higher priority value assigned by PFC is called the *PFC flow rule*, denoted by ($flow_{pfc}$). After $flow_{pfc}$ has been generated, the network traffic is forwarded by $flow_{pfc}$. This operation can reduce processing delay caused by the modification of $flow_{cm}$ because $flow_{pfc}$ can now be easily matched and applied owing to its higher flow priority. In the meantime, (3) the *flow_delete* (or *flow_mod*) command is applied but it does not affect the traffic forwarding behaviour as the traffic forwarding is being handled by $flow_{pfc}$. Lastly, (4) $flow_{pfc}$ is set back to $flow_{nm}$ after the network traffic has been processed. The details of PFC are as follows.

1) Flow Rule Classification to Search Next Matching Flow Rule: When the SDN controller receives a flow command (such as a *flow_delete* or a *flow_mod* command), the first procedure of PFC is to classify flow rules in order to determine $flow_{nm}$ to replace $flow_{cm}$. The flow rules which have a lower priority than $flow_{cm}$ are searched in descending order of priority, and then PFC classifies the flow rules into five types: compatible, superset, subset, overlapping, and disjoint.

- *Compatible* flow rules are the flow rules that have the exact same match fields with $flow_{cm}$. This means that compatible flow rules can be used as a substitute for $flow_{cm}$ because they can only handle the network traffic matched by $flow_{cm}$.
- The match fields of *superset* flow rules are a superset of the match fields of $flow_{cm}$. In other words, superset flow rules can affect not only the network traffic matched by $flow_{cm}$ but also other network traffic, although a superset flow rule can be just $flow_{nm}$ if there are no other flow rules between $flow_{cm}$ and the superset flow rule.
- In contrast to superset flow rules, the match fields of *subset* flow rules are a subset of the match fields of $flow_{cm}$, which means that subset flow rules may or may not be $flow_{nm}$ depending on the network traffic matched by $flow_{cm}$.
- In case of the *overlapping* flow rules, their match fields can overlap the match fields of $flow_{cm}$ and also have at

least one unrelated match field that is not present among the match fields of $flow_{cm}$. Therefore, overlapping flow rules may or may not be $flow_{nm}$.
- Finally, *disjoint* flow rules have match fields which have no overlap with the match fields of $flow_{cm}$.

The case scenarios for these flow rules are as follows. If a network operator modifies the traffic route of a traffic flow by adding a new flow rule which has the same match fields with the previous flow rule but a different flow action set, the previous flow rule becomes a compatible flow rule. In the superset case, a flow rule with a larger granularity-level than $flow_{cm}$ is a superset flow rule. On the contrary, subset flow rules can be utilised when the network operator wants to split a traffic flow into multiple flows with smaller sets of match fields.

During the classification process, if flow rules are classified to be in one of these categories, i.e. subset, overlapping, or disjoint types, PFC then continues flow rule classification. When PFC discovers a compatible flow rule or a superset flow rule, it stops the flow rule classification process and proceeds to its priority control mechanism using the flow rules that have been classified up to that point. In this paper, these flow rules are called the *classified flow rules*. The procedure of the flow rule classification is presented in Alg. 1.

Algorithm 1 Flow Rule Classification

Term
* X_flows is X flow rules

$flows$ = Collect all flows in SDN switch
$flow_{cm}$ = find a current matching flow rule ($flow\_command$, $flows$)
$flows$ = **Filter** ($flows$, lower priority than $flow_{cm}$)
$flows$ = **Sort** ($flows$, descending order of priority)

for $flow$ **in** $flows$ **do**
    -Insert $flow$ into $classified\_flows$
    $type$ = **Compare** ($flow$, $flow_{cm}$)

    if $type == compatible$ **then**
        -Insert $flow$ into $compatible\_flows$
        -**Break**
    else if $type == superset$ **then**
        -Insert $flow$ into $superset\_flows$
        -**Break**
    else if $type == subset$ **then**
        -Insert $flow$ into $subset\_flows$
    else if $type == overlapping$ **then**
        -Insert $flow$ into $overlapping\_flows$
    else if $type == disjoint$ **then**
        -Insert $flow$ into $disjoint\_flows$
    end if
end for

Return [$classified\_flows$, $compatible\_flow$, $superset\_flow$, $subset\_flows$, $overlapping\_flows$, $disjoint\_flows$]

2) Modes of operation in PFC: After flow rule classification, PFC performs its priority control with the classified flow rules, in order to assign the current traffic to the next matching flow rule ($flow_{nm}$). The priority control operation depends on the composition of the classified flow rules, which fall into four categories as defined in Sec. III.B.1, and as a result, one of four modes of operation is activated: compatible, superset, subset, and disjoint mode. These are described as follows.

a) Compatible mode: This mode is activated if the classification operation is terminated by the discovery of a compatible flow rule[1]. There are a few cases in this mode, depending on the types of other flow rules in the set.

Case 1: There is a compatible flow rule and all other flow rules are disjoint flow rules. PFC then assigns the highest flow priority to the compatible flow rule. The priority values of the disjoint flow rules are not altered.

Case 2: There is a compatible flow rule as well as subset flow rules, but no overlapping flow rules. Higher priority values are assigned to the subset flow rules than the priority of $flow_{cm}$ and that of the compatible flow. The compatible flow rule also have a higher priority than that of $flow_{cm}$.

Case 3: If the set of the classified flow rules contains both a compatible flow rule and a number of overlapping flow rules, then PFC assigns a higher flow priority to all the classified flow rules than the priority of $flow_{cm}$. In doing so, PFC should aim to preserve the order of priority among all the classified flow rules so as to prevent an unwanted flow match, because the overlapping flow rules may not only be matched by the current traffic but also affect other traffic flows.

b) Superset mode: This mode is activated if there is no compatible flow rule, but a superset flow rule has been detected during the flow rule classification procedure[1]. In this case, PFC assigns a higher flow priority to all the classified flow rules than the priority of $flow_{cm}$, regardless of the types of the classified flow rules. In doing so, PFC should aim to preserve the order of priority among all the classified flow rules so as to prevent an unwanted flow match, because the superset flow rule may not only be matched by the current traffic but also affect other traffic flows.

c) Subset mode: The subset mode is activated if there are only subset flow rules in the classified rules, or there is a combination of subset and disjoint flow rules after the classification operation has finished. In this case, PFC assigns a higher flow priority to the subset flow rules than the priority of $flow_{cm}$. The priority values of the disjoint flow rules are not altered.

d) Disjoint mode: The disjoint mode is activated when all the classified flow rules are of type disjoint, or a combination of disjoint and overlapping. In this case, priority control is skipped and the flow command is directly translated into the southbound API because there are no alternative flow rules. When there are also overlapping flow rules besides disjoint flow rules, the overlapping flow rules are regarded as disjoint flow rules because their match fields do not exactly match those of $flow_{cm}$, and hence may not be qualified to become $flow_{nm}$, and other traffic flows may be served by these overlapping flow rules.

The priority control mechanism of PFC is outlined in Fig. 4 and Alg. 2.

[1]Note that once the flow rule classification procedure detects a compatible flow rule or a superset flow rule, the flow rule classification is terminated.

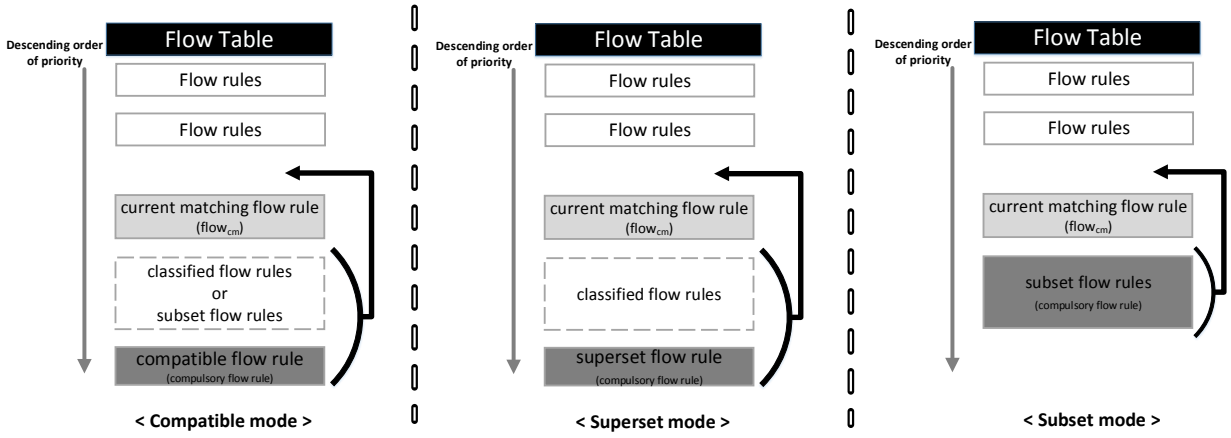

Fig. 4. Prioirty control of PFC.

Algorithm 2 Priority control mechanism of PFC (Modes of operation in PFC)

Term
* X_flows is X flow rules
* **N** (X_flows) is the number of X_flows

1: **if** $classified\_flows > thold_{PFC}$ **then**
   * Restriction for a single flow command
2: -Terminate PFC and perform a normal flow
3: **else**
4: **if** $compatible\_flow$ **then**
5: **if** $overlapping\_flows$ **then**
6: -Assign higher flow priority to $classified\_flows$
7: -Insert $classified\_flows$ into PFC_flows
8: -Terminate PFC operation
9: **else if** $subset\_flows$ **then**
10: -Assign higher flow priority to $subset\_flows$ and $compatible\_flow$
11: -Insert $subset\_flows$ and $compatible\_flow$ into PFC_flows
12: -Terminate PFC operation
13: **else**
14: -Assign the highest flow priority to $compatible\_flow$
15: -Insert $compatible\_flow$ into PFC_flows
16: -Terminate PFC operation
17: **end if**
18: **else if** $superset\_flow$ **then**
19: -Assign higher flow priority to $classified\_flows$
20: -Insert $classified\_flows$ into PFC_flows
21: -Terminate PFC operation
22: **else if** $subset\_flows$ **and** $overlapping\_flows$ == **None** **then**
23: -Assign higher flow priority to $subset\_flows$
24: -Insert $subset\_flows$ into PFC_flows
25: -Terminate PFC operation
26: **else**
27: -Terminate PFC and perform a normal flow operation
28: **end if**
29: **end if**

3) Principles in priority assignment with PFC: There are a few principles that PFC follows when assigning a higher priority to the classified flow rules.

- In the case that PFC performs its priority control with a compatible flow rule and (or) subset flow rules (Case 1 and 2 of Compatible Mode, and Subset Mode), PFC can use priority values from the highest priority, but should keep the flow priority order of the flow rules unchanged.
- When PFC assigns priority values to all classified flow rules (Case 3 of Compatible Mode and Superset Mode), PFC should keep the flow priority order unaltered. Also, the assigned flow priorities by PFC should not exceed the priority of existing flow rules which already have a higher priority than $flow_{cm}$ before performing PFC. This is to preserve the same match order before PFC operation. In the case that there is a small priority difference between $flow_{cm}$ and the existing flow rules, PFC temporarily increases flow priority of the existing flow rules to expand this priority difference.

4) Priority assignment methods: There are two priority control methods to assign a high (or the highest) flow priority to a classified flow rule.

- *Modification Method (MM)* directly modifies the flow priority of a classified flow rule.
- *Duplication Method (DM)* installs a new flow rule which has a set of flow match fields and a set of flow actions which are identical to the corresponding sets of a classified flow rule. This new flow rule has a higher (the highest) flow priority than the priority of $flow_{cm}$.

5) Restrictions of PFC operation: In order to avoid excessive processing overhead (delay) caused by PFC in both an SDN controller and SDN switches, there are restrictions applied to the PFC operation: the number of flow rules inserted by PFC is limited.

a) Number of flow rules added by PFC for a single flow command: The number of $flows_{pfc}$ generated by a single flow command is restricted in PFC because numerous flow rules could be generated inadvertently. The maximum number of $flows_{pfc}$ generated by a single flow command is determined by a threshold denoted by $thold_{PFC}$. This is a predefined value which is used in the priority control procedure (lines 1-2 of Alg. 2). When the priority control procedure is performed, PFC firstly compares the number of classified flow rules to $thold_{PFC}$. If the number of the classified flow rules exceeds $thold_{PFC}$, then PFC operates as the normal interpretation function of the SDN controller (i.e. the priority

control procedure is skipped and the flow command is directly translated to the southbound API). Otherwise, the priority control operations as outlined in lines 4-28 of Alg. 2 are followed.

b) Total number of flows added by PFC in an SDN switch: The total number of flow rules in $flows_{pfc}$ in an SDN switch is also restricted in order to prevent potential excessive processing overhead (delay) caused by PFC. A considerable number of flow rules could be generated by multiple PFC operations in a short time frame, although the maximum number of $flows_{pfc}$ generated by a single flow command is already limited to the threshold ($thold_{PFC}$). Before PFC performs its flow rule classification procedure, it first checks the number of existing flow rules in $flows_{pfc}$ in the SDN switch. If the number of existing rules in $flows_{pfc}$ exceeds γ * $thold_{PFC}$, PFC is operated as the normal interpretation function. Here, $\gamma > 1$ is a predefined value. This happens, when multiple PFC operations are performed in a short time [2].

If PFC receives a flow command for a $flow_{pfc}$, PFC should not be applied to $flow_{pfc}$, and hence such a command is directly converted to the controller's southbound API.

The algorithm of the restrictions is presented in lines 1-2 of Alg. 2 and Alg. 3.

Algorithm 3 Restrictions of PFC operation: total number of flows added by PFC in an SDN switch and PFC flow rules

Term
* X_flows is X flow rules
* **N** (X_flows) is the number of X_flows

1: **if** $flow\_command \notin$ flow change commands **or**
N (PFC_flows) > γ * $thold_{PFC}$ **or**
$flow\_command$ is for PFC_flows **then**
2: -Terminate PFC and perform the normal flow operation
3: **end if**

6) Performing a Flow Command and Restoring a Flow Priority: After priority assignment operation has been completed, PFC waits for a certain amount of time, called the *guard time*, and then starts to perform the flow command. This guard time has the purpose of preventing traffic disruption due to the processing delay caused by priority assignment in an SDN switch. This is necessary, because although the flow command is applied after the priority assignment takes place in the SDN controller, the actual assignment of new priority values in the SDN switch may be still in progress.

It is desirable that the guard time should be set to a value which is as small as possible in order that the controller can quickly perform the flow command. However, it should also be large enough to apply the flow command after the completion of the priority assignment in the SDN switch. Thus, it is necessary to have a suitable guard time to prevent the traffic disruption whilst ensuring that flow commands are not postponed excessively [3].

In order to determine a suitable guard time, the controller periodically monitors switch status and PFC estimates the processing time (denoted by T_p) in the SDN switch at each instance. T_p is defined as:

$$T_p = T_c - RTT_{cs}, \tag{1}$$

where T_c refers to the control latency, which is the time interval between sending a control message from the SDN controller and receiving a response from the SDN switch. T_c can be measured when the controller sends control messages. RTT_{cs} is the round trip time (RTT) between the SDN controller and the SDN switch, and it is also periodically measured by a monitoring function. Based on T_p, the guard time (T_g) is defined by:

$$T_g = \beta * T_p, \tag{2}$$

where β is a predefined value for T_g, which can be set by the network operator. In Section IV-D, the impact of β on network performance is explicitly discussed.

After applying the flow command, the last PFC operation is to delete $flows_{pfc}$ (or to restore $flows_{pfc}$ to $flow_{nm}$) in order to restore flow tables to the same state when the flow command is performed without PFC. Because $flows_{pfc}$ may serve the current network traffic, PFC should delete (or restore) $flows_{pfc}$ after the network traffic has been handled in the SDN switch. In the case that MM is applied as the priority control method, PFC first checks whether the network traffic served by $flows_{pfc}$ exists or not. PFC restores the flow priority of $flows_{pfc}$ if there is no traffic served by $flows_{pfc}$. The served traffic can be detected by checking the statistics of $flows_{pfc}$. When DM is used as the priority control method, the restoring process can be simplified by setting an idle timeout value when $flows_{pfc}$ are inserted. This idle timeout value is set to a predefined value (denoted by T_i). The traffic flow rules ($flows_{pfc}$) can be removed when there are no matching packets related to $flows_{pfc}$ for an uninterrupted time period equal to the idle timeout value [15].

The algorithm for the performing a flow command and the restoring a flow priority is presented in Alg. 4 and the flow chart for the overall PFC operation is shown in Fig. 5.

Algorithm 4 Performing a flow command and restoring a flow priority

T_c and RTT_{cs} are measured by a monitoring function
1: $T_p = T_c$ - RTT_{cs}
2: $T_g = \beta * T_p$
3: **Wait** (T_g)
4: **Perform** ($flow\_command$)
5: **if** Modification Method is used **then**
6: **while** there is traffic served by PFC_flow **do**
7: **Wait** (monitoring period)
8: **end while**
9: **Restore** PFC_flow
10: **end if**

IV. Measurement Results

A. Measurement Environment

In this section, we evaluate the performance of PFC using measurement taken using our SDN testbed. We implement PFC in Ryu which is a component-based SDN framework [16]. PFC is executed when the SDN controller receives flow change

[2]The flow rules in $flows_{pfc}$ which are not matched by any ongoing network traffic can be removed (or restored) shortly after PFC operation.

[3]The benefit of PFC is that since a new flow rule $flow_{pfc}$ is set up, which services the ongoing traffic, the modification of the $flow_{cm}$ does not affect performance.

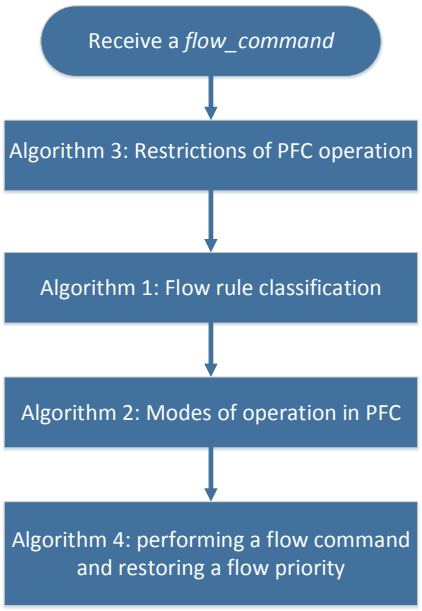

Fig. 5. Overall operation of PFC.

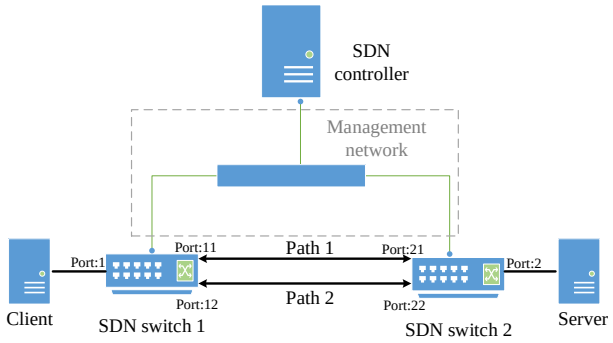

Fig. 6. Measurement Environment: testbed network topology.

commands (*delete* command or *modification* command). Duplication Method (DM) is applied for the priority control of PFC. The idle timeout (T_i), PFC threshold ($thold_{PFC}$) and γ are set to 10 sec, 10 and 10 respectively. T_c and RTT_{cs} are periodically measured by a monitoring function in the SDN controller and the monitoring period is set to 3 sec. In Section IV.B and Section IV.C, there are two cases of PFC: PFC without guard time and PFC with guard time. The PFC without guard time performs the flow change commands without guard time. In case of PFC with guard time, β is set to 3. In Section IV.D, several guard methods are used to consider the performance effect according to the guard time methods. For comparison, the normal flow operation which translates flow commands to OpenFlow commands in Ryu framework is used. We constructed a network topology using hardware SDN switches, an SDN controller, and end terminals. Two Cisco switches (Nexus 3000 series) which support OpenFlow 1.3 are applied as SDN switches. The Cisco switches use double-wide TCAM carving configuration for a 12-tuple match, which supports a maximum of 700 flows for each switch. The SDN controller and the end terminals are operated on PC (i5-2500U @ 3.3GHz with 8GB of RAM) running 64-bit Ubuntu 14.04. The network topology is shown in Fig. 6. The client and server are connected through the two SDN switches, and there are two links between the SDN switches. The SDN switches are connected with the SDN controller through the management network which is used for control messages. All network interfaces in this topology have same bandwidth (1-Gbps port). TCP and UDP are used as transport network protocol and Cubic congestion control [17] is applied in TCP. Measurements are repeated 10 times and all results in Section IV.B and Section IV.C are averaged. The parameters for the measurements are summarized in Table I.

B. Performance Evaluation in Priority-based Flow Control

First, we investigated the effect of path change events on PFC and the normal flow operation during packet forwarding. For the path change events, the traffic path for ongoing traffic is changed to another path by deleting $flow_{cm}$ during packet forwarding and the number of path change events varies between 1 and 8. In case of the multiple path change events, the traffic route periodically fluctuates between path 1 and path 2 which are shown in Fig. 6. Transmission disruption time and packet loss due to path change events are firstly evaluated and then, the effect of path change events is considered on TCP throughput performance.

Figure 7 shows the transmission disruption time of a current traffic due to a path change event. The traffic passes through path 1 and the traffic path is changed to the path 2 once during packet forwarding. In this measurement, UDP is used as transport layer protocol. The vertical line with an arrow indicates the starting point of the path change event and the vertical line with square represents the end point of the transmission disruption time when the normal flow operation is used for the path change. The vertical line with circle presents the end point of the disruption time when PFC without guard time is applied for the path change. The transmission disruption time is measured by analyzing traffic interval at end terminals. The results show that the transmission disruption

TABLE I
MEASUREMENT PARAMETERS

PFC parameter	value
idle timeout (T_i)	10
PFC threshold ($thold_{PFC}$)	10
monitoring period	3 sec
β	1~3
γ	10
topology parameter	**value**
SDN switch model	Cisco Nexus 3000 series
SDN controller	Ryu
OpenFlow version	1.3
link bandwidth	1 Gbps
maximum segment size	1500 bytes

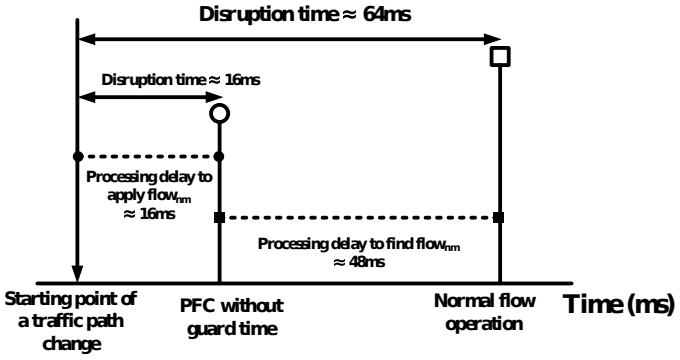

Fig. 7. Transmission disruption time due to a path change event.

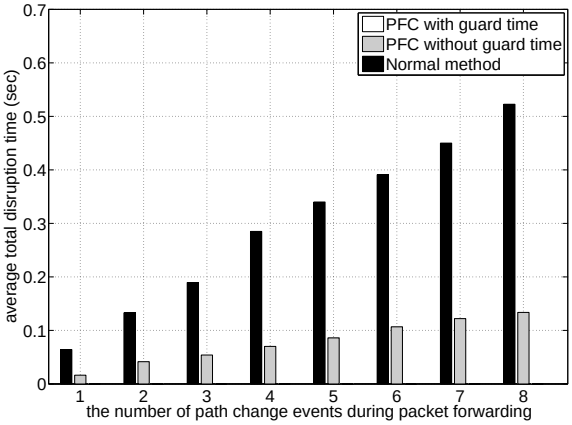

Fig. 8. Average transmission disruption time versus the number of path change events.

time occurs in both the normal flow operation and PFC without guard time cases. In case of the normal operation, the path change causes the disruption time for approximately 64ms. When PFC without guard time is applied for the path change, the disruption time is roughly 16ms. The disruption time of the normal flow operation is related to the total processing delay for the path change in SDN switch. In other words, this disruption time is affected by both the processing delay to search $flow_{nm}$ and the processing delay to apply the current traffic to $flow_{nm}$ in SDN switch. In case of PFC without guard time, the disruption time is associated with the processing delay to apply the current traffic to $flow_{nm}$ in SDN switch because PFC without guard time can eliminate the processing delay to search $flow_{nm}$ in SDN switch. On the basis of both the disruption times, the processing delay to search $flow_{nm}$ can be calculated by subtracting the disruption time of PFC without guard time from the disruption time of the normal flow operation. In Fig. 7, there is no vertical line for PFC with guard time because the current traffic does not experience the traffic disruption when PFC with guard time is used for the path change, which signifies that PFC with guard time can prevent both the disruptions in SDN switch.

Figure 8 shows the average transmission disruption time of a current traffic according to path change events. In this measurement, UDP is also used as transport layer protocol.

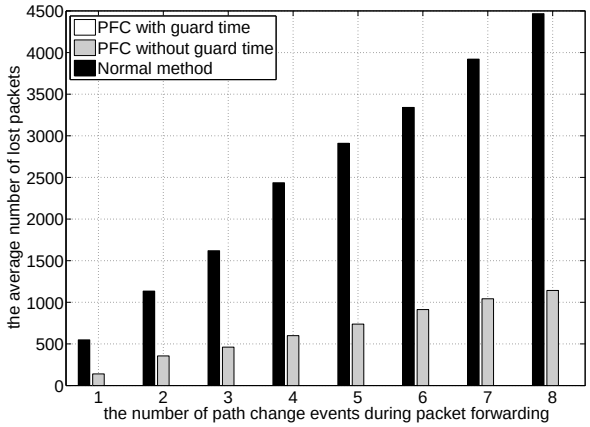

Fig. 9. Average number of packet loss versus the number of path change events.

In Fig. 8, the black bar graph indicates the results of the normal flow operation. The gray bar graph depicts the results of PFC without guard time, and the white bar graph represents the results of PFC with guard time. The results show that the transmission disruption time in both the normal operation and PFC without guard time linearly grows with an increase in the number of path change events. This is because the transmission disruption occurs whenever the ongoing traffic path is modified, and the amount of the disruption time due to a path change event is almost consistent: this amount of the disruption time is almost the same as the results in Fig. 7. However, PFC with guard time does not cause the disruption time regardless of the number of path change events because it eliminates all disruption factors caused by path change events. In other words, a current traffic path can be changed without the disruption time by using PFC with guard time.

Figure 9 shows the average number of packet loss of a current traffic according to path change events. The measurement scenario and the representation of the results are the same as in Fig. 8. The results demonstrate that packet loss events occur when the normal operation and PFC without guard time are used to modify a traffic path of the current traffic. The results also have a similar shape with Fig. 8, which indicates that the packet loss events are proportional to the disruption time. This is because the packet loss events in this scenario are caused by the transmission disruption in SDN switch and UDP traffic is directly affected by the disruption time. Moreover, this packet loss is independent of network conditions. However, PFC with guard time does not experience packet loss events regardless of the number of path change events, which means PFC with guard time support a lossless packet transmission. This is because it eliminates the all disruption factors caused by a path change event, which is also shown in Fig. 8.

In Fig. 10, the effect of path change events is considered on TCP throughput performance. The representation of the results is the same as in Fig. 8. In this scenario, an end terminal generates TCP traffic during 20 sec and path change events occur during the packet transmission. The results indicate that the TCP throughput performance in both the normal operation

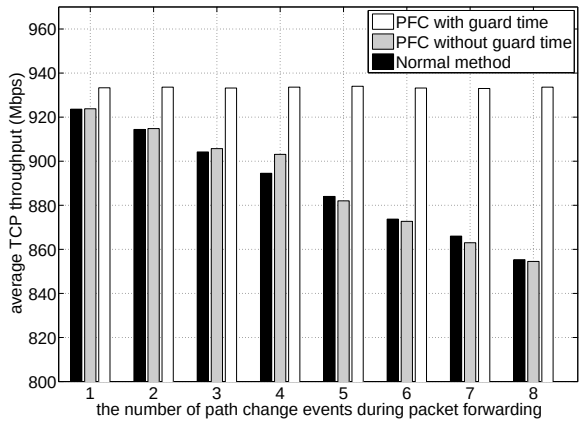

Fig. 10. Average throughput versus the number of path change events.

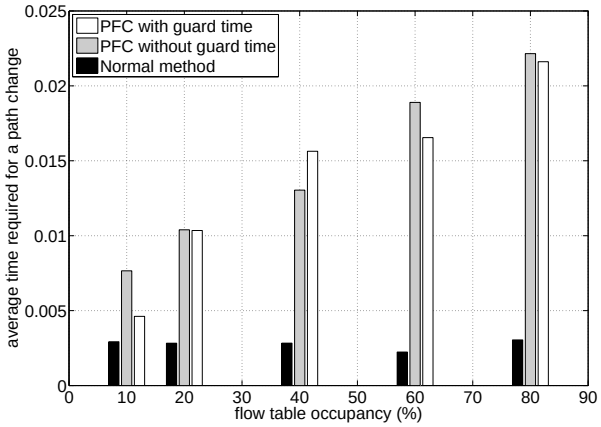

Fig. 11. Average time required for a path change.

and PFC without guard time decreases as the number of path change events increases. This is a natural consequence of the packet loss as shown in Fig. 9 because packet loss shrinks a TCP congestion window, which diminishes TCP throughput. However, the normal operation and PFC without guard time have the similar throughput performance although the normal operation loses more packets than PFC without guard time in Fig. 9. This means that the amount of lost packets due to path change events rarely influences TCP throughput performance: when burst packet loss occurs, the number of packet loss events mainly influences TCP performance. In case of PFC with guard time, there is no performance degradation according to path change events because PFC with guard time can eliminate both the disruption time and the packet loss as shown in Figs. 8 and 9.

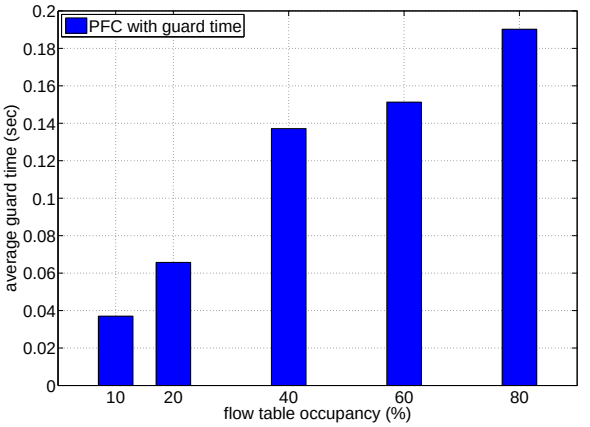

Fig. 12. Average guard time versus flow table occupancy.

C. Overhead of Priority-based Flow Control

In this chapter, we investigate the overhead of PFC according to flow table occupancy and the number of PFC flow rules. The effect of flow table occupancy on PFC is firstly considered and then a processing delay and additional control packets generated by PFC are examined according to the number of PFC flow rules ($flows_{pfc}$).

Figure 11 shows the average time required for a path change according to flow table occupancy in SDN switch. In this measurement, the number of $flows_{pfc}$ is fixed to 10 but the flow table occupancy varies between 10% and 80%. The representation of the results is the same as in Fig. 8. In case of the normal operation, the time required for a path change is almost constant regardless of the flow table occupancy because the normal operation is independent of the flow table occupancy: it simply performs flow commands. In the cases of PFC without guard time and PFC with guard time, the required time for a path change grows with the increase in the flow table occupancy. This is due to that PFC (with/without guard time) firstly performs the flow rule classification for a path change and the processing delay for the flow rule classification increases with the growth of the number of flow rules in SDN switch. The required time for a path change is similar in both PFC without guard time and PFC with guard time because a path change is conducted by the priority control of PFC in both the cases, which means the guard time of PFC does not affect the time required for a path change.

Figure 12 shows the average guard time of PFC according to flow table occupancy in SDN switch. The measurement scenario is the same as in Fig. 11. The results show that the average guard time grows with increasing the flow table occupancy. The reason for this is that T_c which is the main factor to determine guard time increases in proportion to the flow table occupancy. However, note that the guard time of PFC only generates additional latency before performing a flow command, which does not influence the time required for a path change in PFC mechanism.

Figure 13 shows the average processing delay of PFC versus the number of $flows_{pfc}$. In this measurement, the number of total flow rules in each SDN switch is fixed to 120 but the number of $flows_{pfc}$ varies between 1 and 100. The dotted lines represent the average processing delay in SDN switches. The processing delay in SDN switches is measured by $T_{control}$ - $RTT_{control}$. $T_{control}$ is the time interval between sending flow commands with a control message from the

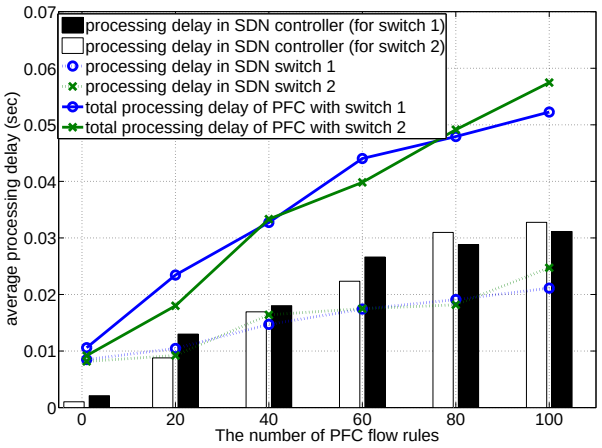

Fig. 13. The average processing delay of PFC.

SDN controller and receiving the response of the control message from SDN switch. In this measurement, the barrier messages are used as the control message. $RTT_{control}$ is the RTT between SDN controller and SDN switch. The bar graph indicates the processing delay of PFC in SDN controller. The black bar represents the processing delay when PFC inserts $flows_{pfc}$ into SDN switch 1 and the white bar shows the processing delay when PFC inserts $flows_{pfc}$ into SDN switch 2. The lines represent the total processing delay which is the sum of the processing delay in SDN controller and the processing delay in SDN switch. Note that these processing delays indicate the additional latency due to inserting $flows_{pfc}$ by PFC before performing the flow command.

The results show that the all processing delays grow with increasing the number of $flows_{pfc}$. It is also shown that although the processing delays in the SDN switches are slightly different between the SDN switches, both SDN switches have similar processing performance because they are the same model. In this measurement, the processing delay in SDN switch is a dominant factor of the total processing delay of PFC when small $flows_{pfc}$ are controlled by PFC. In the case that there are a few $flows_{pfc}$ (the case that the number of $flows_{pfc}$ is under 10), the processing delay in SDN controller is negligible in comparison with the processing delay in SDN switch. However, in the case that the number of $flows_{pfc}$ is greater than 20, the processing delay in SDN controller exceeds the processing delay in SDN switch. These results indicate that the processing performance of SDN controller can be a dominant factor in the total processing delay of PFC when PFC handles considerable $flows_{pfc}$. This phenomenon is because the increment of the processing delay according to $flows_{pfc}$ is different between SDN controller and SDN switch. As shown in Fig. 13, the amount of the increment of the processing delay in SDN switch is less than that of the processing delay in SDN controller although both the processing delays linearly increase with the growth of the number of $flows_{pfc}$.

Table II shows the control messages of the normal flow operation and PFC to perform a flow command. In this table, only control messages which are related to performing a flow command are considered. PFC with DM and PFC with MM indicate the PFC using Duplication Method (DM) and the PFC using Modification Method (MM), respectively. In case of the normal flow operation, the smallest amount of the control messages is required: one control message and one TCP ACK packet are required to perform a flow command. This is because the normal flow operation just sends a flow control message and receives an ACK. The amount of control messages sent by PFC are different depending on the priority assignment methods (DM and MM) as well as the number of $flows_{pfc}$.

In case of PFC with DM, *flow_add* messages are firstly required to add $flows_{pfc}$. α indicates the number of *flow_add* messages which is equal to the number of $flows_{pfc}$. Each *flow_add* message generates a TCP ACK packet for the acknowledgment of the *flow_add* message. After *flow_add* messages are handled, the control messages for a flow command are then used, which is the same as the normal flow operation: one control message and one TCP ACK packet. Thus, the total number of the control messages of PFC with DM to perform a flow command is $2 * (\alpha + 1)$.

In case of PFC with MM, the required control messages to perform a flow command is the same as that of PFC with DM ($2 * (\alpha + 1)$) but additional control messages are required to restore the priorities of $flows_{pfc}$. Firstly, additional flow status information may be required in PFC with MM because PFC with MM needs to monitor a current traffic unlike PFC with DM. τ indicates the number of additional monitoring messages to check the flow status of the current traffic and τ can be zero if PFC relies on a monitoring function to monitor the flow status. If PFC with DM uses the additional monitoring messages, TCP ACK packets are also generated for the acknowledgment of the additional monitoring messages: the required number of control messages for the monitoring is $(2 * \tau)$. After the current traffic is processed, PFC with MM uses flow control messages (*flow_mod*) to restore $flows_{pfc}$ to $flows_{nm}$. These control messages also cause TCP ACK packets: the required number of control messages for this restoration equals twice the number of $flows_{pfc}$ $(2*\alpha)$. Thus, the total number of control messages of PFC with MM is $2 * (2 * \alpha + \tau + 1)$.

TABLE II
CONTROL MESSAGES FOR A FLOW COMMAND

Methods	Control Messages	The Number of Packets
Normal	flow delete	1
	TCP ACK	1
	ALL	2
PFC with DM	flow add	α
	flow delete	1
	TCP ACK	α+1
	ALL	2*(α+1)
PFC with MM	flow mod (add)	2*α
	flow delete	1
	flow stats request	τ
	flow stats reply	τ
	TCP ACK	2*α+1
	ALL	2*(2*α+τ+1)

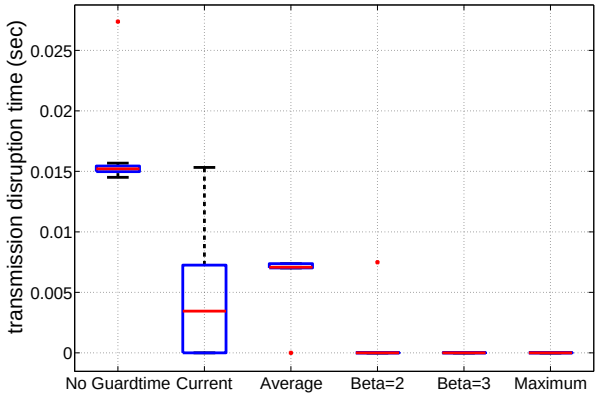

Fig. 14. Transmission disruption time according to the guard time methods.

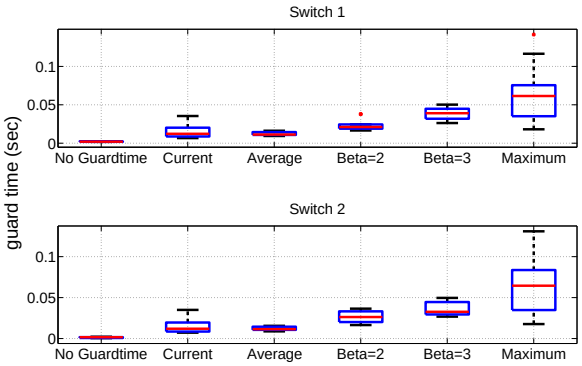

Fig. 15. Guard time of PFC.

D. Effect of Guard Time in Priority-based Flow Control

In this chapter, we investigate the effect of guard time on PFC. In order to evaluate the guard time effect, several guard time methods are applied in PFC. Because the purpose of guard time in PFC is to wait for the completion of the priority control in an SDN switch, guard time is related to the traffic disruption due to the processing delay in conveying a current traffic to $flows_{pfc}$ in the SDN switch.

Figure 14 shows the transmission disruption time of PFC according to guard time methods. The results are represented by box-and-whisker plots and the measurement scenario is the same as in Fig. 7. The current method uses purely T_p for guard time and the average method utilizes the average value of T_p. The beta methods use T_g with their beta values and the maximum method utilizes the maximum value of T_p during operation of PFC. In the results, the no guard time method causes the longest transmission disruption time among the guard time methods. This is because the processing delay in conveying a current traffic to $flows_{pfc}$ in SDN switch fully affects packet forwarding. In the current method, the transmission disruption time is relatively variable than the other methods, which means solely utilizing T_p is not suitable for guard time. The average method also generates the transmission disruption time although this disruption time is less than that of the no guard time method. Thus, the average method can only mitigate the transmission disruption time. When PFC uses the beta methods, the disruption time can be eliminated: only one experiment causes the transmission disruption when the beta value is set to two. The maximum method can also eliminate the transmission disruption time. The elimination of the transmission disruption in the beta methods and the maximum method is owing to the sufficient guard time which is shown in Fig. 15.

Figure 15 shows the guard time of PFC according to the guard time methods. The upper subgraph indicates the guard time for SDN switch 1 and the lower subgraph represents the guard time for SDN switch 2. Note that guard time is assigned to each SDN switch based on its processing performance. The representation of the results and the measurement scenario are the same as in Fig. 14. The no guard time method naturally does not use guard time, which means there is no additional latency for guard time. However, this method makes PFC vulnerable to the processing delay in conveying a current traffic to $flows_{pfc}$ in the SDN switch. In case of the current method, the guard time is relatively distributed to near 15ms which is the average of the guard time in the current method. Because the disruption time of the current method shown in Fig. 14 also fluctuates, both the results signify that the variation of the guard time cannot reflect the variation of the processing delay in SDN switch, which also indicates that the guard time cannot be optimized by using T_p directly. The average method has on average slightly less guard time (roughly 12ms) than the current method and this guard time cannot fully eliminate the disruption time as shown in Fig. 14. The guard time in the beta methods grows with the increase of the beta value and the maximum method utilizes the longest guard time. However, as shown in Fig. 14, the beta methods and the maximum method can eliminate transmission disruption time in SDN switch.

To summarize the results of the effect of guard time, there is a trade-off between guard time and the transmission disruption time within a certain threshold. In cases of the current method and the average method, the transmission disruption time increases with the decrease of guard time. On the other hand, if the guard time exceeds a certain threshold, the transmission disruption can be prevented. In the cases of the beta methods and the maximum method, the disruption time is almost eliminated regardless of the guard time.

V. Conclusion

In this paper, we present a novel flow control mechanism for SDN in order to prevent transmission disruption, which is caused by the processing delay due to a flow modification process in an SDN switch. PFC is a software-based solution in SDN controller, which searches $flow_{nm}$ and shifts a current traffic from $flow_{cm}$ to $flow_{nm}$ by using priority control. Through the measurement in our testbed, we determined that PFC can dynamically change a traffic path without transmission disruption although additional latency and control messages can be generated to perform a flow command.

Consequently, PFC can achieve higher TCP throughput than the normal flow operation, which experiences a throughput degradation due to transmission disruption. As part of our future work, an evolved framework and mechanism for multiple SDN controllers will be considered. Lastly, we believe that PFC can contribute to dynamic traffic management in SDN.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of the University of Surrey's 5G Innovation Centre (5GIC) (http://www.surrey.ac.uk/5gic) members for this work.

REFERENCES

[1] N. Mckeown, How SDN will shape networking, Oct. 2011.
[2] S. Schenker, The future of networking the past of protocols, Oct. 2011.
[3] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks", SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69-74, Mar. 2008.
[4] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, M. Thottan, J. Rexford, A. Vahdat, "Measuring control plane latency in sdn-enabled switches", Proc. the 1st ACM SIGCOMM Symposium on Software Defined Networking Research SOSR'15, pp. 25:1-25:6, June 17-18, 2015.
[5] M. Kuzniar, P. Peresini, D. Kostic, M. Canini, "Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches", Computer Networks, Volume 136, Pages 22-36, 2018.
[6] A. R. Curtis et al., "DevoFlow: Scaling flow management for high-performance networks", Comput. Commun. Rev., vol. 41, no. 4, pp. 254-265, Aug. 2011.
[7] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation", Proc. 13th Int. Conf. Passive Active Meas., pp. 85-95, 2012.
[8] D. Y. Huang, K. Yocum, A. C. Snoeren, "High-fidelity switch models for software-defined network emulation", Proc. Second ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw., pp. 43-48, 2013.
[9] M. Kuzniar, P. Peresini, D. Kostic, J. Mirkovic, Y. Liu, "What you need to know about SDN flow tables" in Passive and Active Measurement, Cham, Switzerland:Springer-Verlag, vol. 8995, pp. 347-359, 2015.
[10] S. Luo, H. Yu, L. M. Li, "Fast Incremental Flow Table Aggregation in SDN", Proc. 23rd Int'l Conf. Computer Communication and Networks (ICCCN 2014), pp. 1-8, Aug. 2014.
[11] M. Moshref, A. Bhargava, A. Gupta, M. Yu, R. Govindan, "Flow-level state transition as a new switch primitive for SDN", Proc. 3rd Workshop Hot Topics Softw. Defined Netw., pp. 61-66, 2014.
[12] S. Song, H. Park, B. Y. Choi, T. Choi and H. Zhu, "Control Path Management Framework for Enhancing Software-Defined Network (SDN) Reliability," in IEEE Transactions on Network and Service Management, vol. 14, no. 2, pp. 302-316, Jun. 2017.
[13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN", Proc. ACM SIGCOMM Conf., pp. 99-110, 2013.
[14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, "P4: programming protocol-independent packet processors" ACM SIGCOMM Comput Commun Rev, 44 (3), pp. 87-95, 2014
[15] OpenFlow Switch Specification Version 1.5.1 [Online]. Available: https://www.opennetworking.org/images/stories/downloads/ sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf
[16] Ryu SDN Framework Community [Online]. Available: http://osrg.github.io/ryu/.
[17] I. Rhee and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," in Proc. PFLDNet, pp. 1-6, Feb. 2005.