PARD: Hybrid Proactive and Reactive Method Eliminating Flow Setup Latency in SDN

Michał Rzepka¹ · Piotr Boryło¹ · Artur Lason¹ · Andrzej Szymanski¹

Received: 4 March 2019 / Revised: 18 June 2020 / Accepted: 23 June 2020 / Published online: 9 July 2020
© The Author(s) 2020

Abstract

Advantages of Software Defined Networking are unquestionable and are widely described in numerous scientific papers, business white papers and press articles. However, to achieve full maturity, crucial impediments to this concept and its shortcomings must be overcome. One of the most important issues regards significant setup latency of a new flow. To address this issue we propose PARD: a hybrid proactive and reactive method to manage flow table entries. Additional advantages of the proposed solution are, among the others, its ability to preserve all capabilities of Software Defined Networking, utilization of multiple flow tables, a possibility to employ fine-grained traffic engineering and, finally, compatibility with existing protocol and hardware design. It is shown that the proposed solution is able to significantly reduce latency of first packets of a new flow, which directly impacts packet loss and perceived throughput. Thus, our solution is expected to enable a wide deployment of Software Defined Networking concept without any need for protocol changes or, what is extremely important, hardware modifications.

Keywords SDN · Network management · Flow setup · Control plane · OpenFlow · Network performance

✉ Michał Rzepka
mrzepka@agh.edu.pl

Piotr Boryło
borylo@agh.edu.pl

Artur Lason
lason@kt.agh.edu.pl

Andrzej Szymanski
szymans@agh.edu.pl

¹ Department of Telecommunications, AGH University of Science and Technology, Krakow, Poland

1 Introduction

The main paradigm of Software Defined Networking (SDN) is to separate a control plane from a data plane. The control plane is implemented in a form of a logically centralized entity called an SDN controller while the data plane is composed of simplified forwarding devices, called SDN switches. Hence, a communication channel between the planes and an appropriate protocol is required. The OpenFlow Protocol (OFP) is designed for this purpose. Through the OFP the SDN controller sends, among the others, entries to be placed in switches' flow tables in order to implement globally scoped routing policies, which is in contrast to the suboptimal hop-by-hop approach. Thanks to centralized nature of the SDN, the policies may easily be managed by a network administrator or automatically programmed by external applications integrated with the controller [1]. Programmability, combined with controller's global knowledge about network state, together with an abstraction of network layer exposed to the external applications, becomes the main advantage of the SDN concept which follows the network *softwareization* process. The aforementioned architecture is illustrated in Fig. 1.

Despite opening attractive perspectives for network operators, SDN also raises numerous scientific and engineering concerns. The most important issues are

Fig. 1 The architecture of an SDN network with the OpenFlow Protocol

Fig. 2 Flow matching in an SDN switch according to the OFP

related to the scalability of its architecture based on a separate and logically centralized control plane. In this paper we focus solely on the problem of reducing latency of a new flow setup process while preserving programmability and elasticity of the SDN. This issue is one of the most significant impediments to the widespread SDN deployment [2, 3].

1.1 Flow Path Establishment Process in SDN

Any OpenFlow-compatible SDN switch stores entries provided by the controller in a flow table. Each entry is composed of various fields, such as priority, cookie, statistics, and a list of actions, e.g. sending a packet to a particular output port. However, the key element of each entry is a *match* field which defines packet parameters, usually related to a packet header, required to assign a packet to a flow. During packet forwarding, selected fields are matched to an entry with the highest priority in the table, and actions associated with the entry are performed. The important assumption is that when a packet could be matched by multiple entries in flow tables, the switch performs an action associated with an entry with the highest priority. Such OpenFlow-specific packet processing pipeline ensures that a flow may be processed in its entirety only by a single flow table entry at a time. This feature may be used to provide accurate network monitoring as statistics of all traffic related to a single flow are stored together with its corresponding unique flow entry. Figure 2 illustrates the flow matching process according to the OFP. As SDN switch is bereft of control plane, when it receives a packet that cannot be matched to any entry in its flow table, it may drop the packet or hand over the packet to the controller to determine appropriate action. Such process is called a reactive one and in most cases leads to establishment of the path for the new flow. The latter scenario requires sending at least a header of the packet to

the controller in an *OFPT_PACKET_IN* message. This action is usually provided by so-called *TABLE MISS* entry which is a flow entry with the lowest possible priority. The controller prepares an adequate flow entry reflecting the routing policy and sends it back to the node. However, the node is not able to handle packets of the new flow until it receives a response from the controller. In [4] the authors measured that Round-Trip Time (RTT) between data plane and control plane can be four times higher in SDN than in traditional networks, even if request processing time in the controller is neglected. The latency in serving new flows may be further increased by control plane overload or internal switch latency in generating *OFPT_PACKET_IN* messages [2, 3, 5, 6]. A negative impact shows up in out-of-order packets, decrease of throughput of TCP flows (especially important for short flows), or increased UDP packet loss.

In order to mitigate these issues a proactive flow path establishment process may be deployed. Proactive denotes installing flow entries in network nodes before the first packet of a new flow arrives. However, a common approach is to define flows using 5-tuples: source and destination IP address, source and destination port, and transport layer protocol. As a consequence, it is not possible to install all rules in advance due to a huge number of possible combinations. Therefore, wildcards are used to install aggregated entries in nodes' flow tables. A widely used approach is to define aggregates based on ranges of IP addresses. Unfortunately, such an approach comes with numerous limitations. Its granularity and elasticity in terms of network optimization is strongly limited, as defining routing policies solely based on IP addresses prevents introducing application-aware network optimization, which is one of the most attractive advantages of SDN. The lack of granularity affects also the process of gathering OFP statistics in a flow-based manner as only aggregated statistics will be available. What is more, flow entries installed in advance cannot be formulated based on, for example, the size or inter-arrival times of a first few packets of a new flow. Finally, any modification of an aggregate entry results in rerouting of all transmissions handled by this aggregate, which may further cause network instabilities, losses and retransmissions [7].

In this work we try to mitigate issues of both fully reactive and fully proactive approaches. In a survey addressing the problem of placing flow entries in OpenFlow networks [8] the authors stated that a hybrid approach is an expected way to handle traffic. However, it is still an open question how to combine proactive and reactive modes efficiently. In our opinion, the solution should:

- Eliminate flow setup latency by simultaneously providing proactive handling of traffic at the aggregate level and applying reactive mechanisms to create detailed flow table entries with higher priorities.
- Allow the use of static optimization methods to prepare proactive routing policies.
- Ensure that sophisticated dynamic mechanisms may be used to handle detailed flows. Possible time demanding mechanisms include application-aware [9] and energy-aware routing [10, 11] or integration with cloud [12].
- Ensure fine-grained programmability of a control layer.

- Ensure that changing policy of handling new flows will not affect any flows being serviced, e.g. for dynamic traffic flow steering purposes in Network Function Virtualization (NFV) scenarios [13].
- Ensure that only well known and widely deployed protocols are used (i.e. OFP), with little or no modifications.
- Avoid any impediments to *softwareization* process caused by dedicated hardware.
- Eliminate a problematic requirement of defining and detecting *elephant* flows.
- Retain advantages of distributing entries between multiple tables within a network node.

We designed our solution to adhere to those guidelines. We propose¹ an improvement to the flow path establishment process that eliminates latency in handling new flows while preserving all of the valuable capabilities of the SDN concept. In the proposed solution flows are managed in a hybrid manner using the OFP without any modifications. The novelty of the approach is a proper combination and deployment of the aforementioned mechanisms so that SDN capabilities and advantages are preserved. By combining an appropriate assignment of priorities to aggregate (coarse-grained) entries and detailed (fine-grained) entries, and a proper distribution of those entries in single or multiple tables, it is possible to develop hierarchical and efficient packet processing. On the one hand, aggregates are installed in advance based on static network optimization which eliminates latency in flow setup process. On the other, in order to ensure programmability, detailed flow entries are prepared based on dynamic network optimization.

Additionally, one of the proposed mechanisms, Proactive Aggregate and Reactive Detailed-Multiple Tables (PARD-MT), takes advantage of multiple flow tables available in many OpenFlow devices. The selected benefits of a multiple table approach are as follows:

- Detailed flows are grouped in tables to maximize the gain from efficient queries for OpenFlow statistics,
- Ternary Content Addressable Memory (TCAM) memory may be effectively utilized by using it to store aggregates,
- Actions over tens of thousands of entries like modifications or garbage collection are easily manageable [3].

The Proactive Aggregate and Reactive Detailed (PARD) solutions are applicable in various use-cases. The brief overview of how the mechanisms may be tailored to the needs of a specific scenario is presented in Sect. 3.4.2, along with examples of their usage in data center, Internet Service Provider (ISP) network and a generic layer-3 routing deployment.

¹ The proposed mechanism is a subject of European Patent, number EP3531622A1, publication date: 08.01.2020.

2 Related Work

Solutions that address the flow setup latency problem, present in the literature, may be classified with regard to the plane on which they operate, e.g. data plane, control plane or hybrid. Simultaneously, another important taxonomy divides solutions based on the scope of their operation, considering network wide and single node perspectives. Only few works can be considered as directly related to our solution.

In [5] the authors carefully investigated latencies introduced by control plane. The research is conducted using four SDN switches and three measurement-driven latency mitigation techniques are proposed. All of the solutions are deployed in the control plane and operate in a network wide perspective. The authors concluded that their solution may not be sufficient for the most demanding network applications.

It should be noted that the optimization problem of placing flow table entries is usually solved with the aim of reducing memory consumption in SDN devices or limiting the signalling overhead. However, as a side effect, flow setup latency is often minimized or eliminated.

First of the examples, the *DIFF* solution presented in [14], implements a routing scheme that solves the problem of unbalanced flow table utilization in SDN switches. The aim is to evenly distribute flow entries across network nodes regardless of their location in the topology (central or edge). As a result, flow table overflow is less probable and performance improvement (also related to flow path establishment latency) is observed. In addition, large volume flows are detected and rerouted in a way that satisfies their resource demands.

An alternative *K Similar Greedy Tree (KSTG)* algorithm aims to prevent flow table starvation as well [15]. In the mechanism, Multiprotocol Label Switching (MPLS)-based routing is implemented to reduce the number of required flow table entries by aggregating traffic on certain path segments within a common MPLS label. In the end, a reduction of the number of flow table entries by up to 60% is achieved. A similar concept was implemented in the *JumpFlow* mechanism that uses VID to identify aggregated flows instead of a MPLS label [16].

In *MINNIE*, simple aggregated flow table entries are created, based on wildcarded source and/or destination addresses in packet headers [17]. The mechanism performs load-balanced routing that can be easily deployed in SDN environment. Although the utilization of flow table is significantly reduced, the fully-proactive approach implemented in this mechanism limits controller's ability to dynamically react to changes in the network traffic.

A more sophisticated method of aggregation is proposed in [18]. While handling the issue of flow table overflow, Quality of Service (QoS) classes of the traffic are preserved in newly created aggregated flows to provide desired end-to-end latency. However, the mechanism implements a proactive approach only and does not alleviate issues related to new flow establishment latency experienced when the first packet of a new flow arrives at an interface of an SDN switch.

The concept of flow aggregation is further explored in the *AggreFlow* mechanism designated to operate in power-efficient Data Center Network (DCN) [19]. It reduces flow table occupancy and related control messaging overhead by aggregating

fine-grained flows into coarse-grained flows. At the same time constraints related to power consumption are considered, as paths determined by the routing algorithm determine links that are dynamically turned on or off by the SDN controller. In addition, the mechanism ensures that link capacities are not exceeded.

The *SwitchReduce* mechanism, proposed in [20], operates in the control plane and considers a network wide perspective. Its main aim is to reduce flow table size by aggregating entries directing traffic to the same output port. As those entries are installed in a proactive manner, it also reduces latency of the first packets of a new flow. However, its benefits come at a cost of a few limitations. Multiple nesting of VLAN headers increases data plane overhead and may raise MTU issues in larger networks. Simultaneously, OpenFlow statistics may be collected only with granularity of the aggregates. Additionally, hardware SDN switches rarely support necessary actions, while commodity network adapters may filter out packets with multiple VLAN tags when software switches are considered.

It may be observed that all of the aforementioned solutions try to implement a routing mechanism that designates optimal paths with regard to use-case specific constraints such as flow table occupancy, link utilization, power consumption, control channel load or QoS policies. Reduction of the necessary flow table capacity achieved by aggregating flow table entries may indeed positively impact the overall network performance and the flow path establishment latency. However, design choices that improve performance of those mechanisms in specific scenarios make them less versatile at the same time. Moreover, due to incorporating a network wide perspective in the decision algorithms, the aforementioned solutions don't focus on the flow entry creation process itself and leave issues related with reactive traffic handling for further studies.

A proactive approach is used in [21] for networks hosting numerous Internet of Things (IoT) sensors. In such an environment, periodic traffic flows are possible to predict and estimate. As a result, appropriate flow entries may be created in advance to eliminate control channel overloading and path establishment latency caused by a reactive approach.

The mechanism proposed in [22] designates network segments as core or edge. A different flow aggregation strategy is applied to each of the segments. While edge devices handle a lower number of fine-grained flows that are related to end-user traffic, core nodes handle coarse-grained aggregated flows that convey large amounts of traffic. Flow path establishment in the core is not required due to a proactive approach and, as a result, traffic does not suffer from flow path establishment latency. Meanwhile, traffic handled by reactive approach in the edge is slightly less affected by control channel load and path establishment latency, as single edge nodes forward a lower amount of traffic.

The common assumption in these solutions is that traffic on a single node may be handled in just a single way—either reactively or proactively. Each of the choices introduces some shortcomings. The reactive approach may increase flow path establishment latency, while the proactive approach requires prior calculation of paths and accurate prediction of traffic. This conclusion is a foundation for hybrid mechanisms that enhance SDN network operation by creating new flow entries in a hybrid reactive-proactive manner [23].

In [4] the authors proposed *DevoFlow* mechanism which operates in the data plane of a single network node and aims at reducing signalling overhead in the control plane. As a side effect *DevoFlow* reduces latency in handling new flows. A hardware modification of an OpenFlow switch enables it to identify significant flows that should be handled by the network controller while all other flows are handled locally by the switch. Not only hardware modifications are necessary but also extension to the OpenFlow Protocol is required in the form of an additional action flag: *CLONE*. The benefits of this solution come also at a cost of losing the complete knowledge about flows in the central controller. It collides with the concept of SDN in terms of elasticity in changing traffic handling policies and QoS policies, maintaining network security or gathering statistics.

The *Flow-split* mechanism proposed in [24] is an extension to [4]. The *CLONE* flag is substituted by the *flow-split* action which enables setting OpenFlow attributes, in addition to cloning the match fields. The greatest advantage of this approach, compared to the *DevoFlow* mechanism, is its possibility to change rules of handling future requests without affecting flows that are already present in the network. However, the *Flow-split* mechanism retains many disadvantageous properties of its predecessor, such as a requirement of hardware modifications of SDN switches and a requirement to extend the OpenFlow protocol, which limit practical deployments of this solution.

Finally, the solution presented in [25] aims at reducing memory consumption of flow tables by utilizing a hierarchical network topology specific for data center networks. No modifications in network nodes are required. Small flows are aggregated in a proactive manner while large flows are handled reactively. Thus, in this work proactive and reactive approaches are combined. However, its main disadvantage is that flows' sizes must be discovered in advance and this process may introduce some additional latencies.

All of the works described above are briefly summarized in Table 1, with major similarities to our approach underlined. As observed, most solutions discussed in the section operate in network scope and aim to prevent flow table starvation by introducing flow aggregation and specific routing mechanisms. These mechanisms effectively result in reducing flow path setup latencies. However, only few works focus on key concepts considered in PARD: a single-node scope and the flow entry installation process itself. Both *DevoFlow* [4] and *Flow-split* [24] attempt to introduce a hybrid reactive-proactive approach discussed in [23]. However, in contrast to PARD, both solutions require modifications of SDN switches and the controller, due to use of non-standard OpenFlow extensions. The main aim of our research is to provide an alternative solution that eliminates flow setup latency by tweaking flow installation process, while remaining out-of-the-box compatible with all OpenFlow-conformant hardware and software. PARD preserves the most important capabilities of the SDN, thanks to a combination of proactive and reactive mechanisms, allowing to apply varied policies for fine-grained and coarse-grained components of network traffic. In addition, the proposed solution retains a possibility to distribute flow entries in multiple tables of an SDN switch. The solution operates in the scope of a single network node, is fully programmable and implemented in the control plane.

Table 1 Main features of related works

Solution	Scope	Main contributions
<i>DIFF</i> [14]	Network	Flow entry distribution, routing
[5]		Flow entry distribution, flow entry installation order
[18]		Flow entry distribution, flow aggregation
<i>KSGT</i> [15]		Flow entry distribution, flow aggregation, routing
<i>JumpFlow</i> [16]		Flow entry distribution, flow aggregation, routing
<i>MINNIE</i> [17]		Flow entry distribution, flow aggregation, routing
<i>SwitchReduce</i> [20]		Flow aggregation
<i>AggreFlow</i> [19]		Flow aggregation, energy-efficient routing
[22]		Differentiated traffic flow aggregation in edge and core network
[25]		Differentiated traffic flow aggregation based on hierarchical network topology
[21]		Proactive routing mechanism based on traffic prediction
[23]	Node	Survey of <u>flow entry installation mechanisms</u>
<i>DevoFlow</i> [4]		OpenFlow extension for hybrid <u>flow entry installation mechanism</u>
<i>Flow-split</i> [24]		OpenFlow extension for hybrid <u>flow entry installation mechanism</u>

3 Flow Path Establishment Approaches

The approaches proposed in this paper exploit selected principles of the SDN concept together with the OFP while utilizing both proactive and reactive mechanisms. As a result, they are able to effectively manage traffic in the SDN architecture, ensure reliability and dynamically react to infrastructure changes and user requests. In this section, a few features of the proposed mechanisms are discussed along with a detailed description of each approach considered in the paper. Section 3.1 provides a description of the purely-reactive reference approach, while Sects. 3.2 and 3.3 introduce proposed approaches for single and multiple tables, respectively. Both of the PARD approaches make use of proactively-added coarse-grained (aggregate) flow entries and reactively-added fine-grained (detailed) flow entries at the same time. Section 3.4 concludes the preceding sections by highlighting and discussing key concepts related to the proposed solution. The presented PARD scheme, together with the mechanisms described in subsequent sections, are the main contributions of this paper.

The following notation is used: $P1$ denotes the first packet of a new flow; $P2$, $P3$, ... denote subsequent packets of the flow; A , B , ... are separate aggregate entries; $A.1$, $A.2$, ... are detailed entries corresponding with an aggregate A , i.e. they are a subset of the aggregate but provide more specific match fields.

3.1 Reactive Detailed Entries in a Single Table (The Reference Scenario)

This reference solution represents a basic way of processing traffic in the SDN network. It handles all the incoming flows reactively and creates a detailed flow entry for each new flow. When a packet of a new flow arrives at the SDN switch, it is redirected to the controller and handled no earlier than an action for the new flow is determined. Flow entries are placed in a single table as presented in Fig. 3a. There is no need to make a distinction between aggregate (coarse-grained) and detailed (fine-grained) flow entries as all of them apply just a single output action.

The first packet of a new flow (*P1*) is not matched by any of the flow entries with priority higher than a *TABLE MISS* entry. Thus, according to this entry *P1* is sent to the controller in an *OFPT_PACKET_IN* message. The controller decides the action (usually output port) and using *OFPT_FLOW_MOD* message installs a detailed entry in the flow table. Subsequent packets (*P2*, *P3*) of the flow are matched by the newly inserted entry.

Such a reactive approach ensures that routing policies may be adapted for a new flow without affecting any of the existing flows. The controller is able to prepare an entry for each new flow using any optimization technique (e.g. static or dynamic). Simultaneously, all the existing flows are handled by entries already present in flow tables which are not modified until their timeout expires. However, the advantage comes at a cost of flow setup delay. For each new flow a network node must exchange messages with the network controller. As a result, the packet handling time is affected by round-trip transmission latency between the network node and the controller. Another factor contributing to the total delay is a decision process at the controller that may require running complex path computation algorithm or database lookup. The issue is further amplified if the controller is congested.

3.2 Proactive Aggregates and Reactive Detailed Entries in a Single Table (PARD-ST)

Proactive Aggregate and Reactive Detailed-Single Table (PARD-ST) is the first of the proposed approaches. It introduces flow entries with different levels of aggregation that contain dynamically (reactively) determined actions for single fine-grained flows or default proactively-created actions for coarse-grained flow aggregates. Moreover, PARD-ST allows to quickly handle packets belonging to new flows by using actions that forward packets to the default output port before the final decision is made by the controller. The mechanism creates proactive aggregate entries and reactive detailed entries. As a result, new flows are handled in a hybrid, both reactive and proactive manner. While traffic is forwarded based on already created rules, the controller is notified about new flows and capable of changing the routing policy for selected flows by modifying or adding new reactive rules. In opposition to PARD-MT presented in the next section of the paper, this approach uses only a single flow table.

Fig. 3 Packet processing workflows

Table 0

(a) Detailed entries in a single table

Table 0

(b) Aggregates and detailed entries in a single table (PARD-ST)

Table 0

Table 1

(c) Aggregates and detailed entries in multiple tables (PARD-MT)

In PARD-ST, all the entries are placed in a single flow table while priorities are used to group detailed entries according to the aggregates that correspond to them, as presented in Fig. 3b. In each group detailed entries (e.g. $A.1$, $A.2$, ...) have higher priority than a corresponding aggregate entry (A). The aggregate entry A initiates two concurrent actions: the first is to send packet to the selected output port, the second one is to send packet to the special *OFPP_CONTROLLER* port which is equivalent to sending packet to the controller.

The first packet of a new flow (PI) is not matched by any of the detailed entries (i.e. $A.1$, $A.2$) but is matched by the aggregate entry e.g. A , and is immediately handled using actions associated with the A entry. Therefore, PI is sent to the selected output port and, simultaneously, sent to the controller in *OFPT_PACKET_IN* message. The controller decides the action (usually output port) and using the *OFPT_FLOW_MOD* message installs a new detailed entry ($A.3$) in the flow table. A proper priority assignment is crucial in this step. Subsequent packets of the flow ($P2$, $P3$) are matched by the newly inserted entry $A.3$ which has a higher priority than A and are handled without any controller participation.

Extending the action list of the aggregate entry eliminates flow setup latency as the network node is able to immediately handle traffic in the data plane and simultaneously send it to the control plane for further processing. Proactive aggregate entries may define output ports based on sophisticated static optimization mechanisms as this optimization does not affect packet latency. What is more, a change of the aggregate entry A does not affect existing flows matched by the detailed entries $A.1$, $A.2$ or $A.3$. The method does not imply significant overhead in terms of flow table size as only carefully chosen additional aggregate flow entries are added. Moreover, the number of *OFPT_PACKET_IN* messages sent to the controller is no higher than in the reference case (as discussed in Sect. 3.1). All of the SDN capabilities related to network optimization are preserved thanks to the dynamic, application-aware mechanism preparing detailed flow entries. This process also does not have very strict latency requirements.

3.3 Proactive Aggregates and Reactive Detailed Entries in Multiple Tables (PARD-MT)

The PARD-MT is the second proposed approach, which extends PARD-ST by distributing entries in multiple flow tables. Both of them are based on the same principle of using different levels of aggregation in proactively-created aggregate entries and reactively-created detailed entries present in the OpenFlow pipeline. Just as in the PARD-ST, both aggregate (coarse-grained) and detailed (fine-grained) flows are represented by multiple flow table entries. Groups of detailed entries are placed in separate tables, corresponding with aggregates, as presented in Fig. 3c.

The first packet of a new flow (PI) is compared against aggregate entries in the first table (A , B , ...). Each of these entries invokes the *OFPT_GOTO_TABLE* instruction which directs the packet to a table containing detailed entries corresponding with the aggregate. In other words, if an aggregate A matches packet PI , the packet is further compared against $A.1$, $A.2$, ... entries placed in a separate

table. If none of the detailed entries matches the *PI* packet then the packet is handled according to the action associated with the aggregate entry *A* (i.e. it is usually sent to the selected output port). Simultaneously, based on an instruction associated with a *TABLE_MISS* entry of the detailed table the packet is sent to the controller in an *OFPT_PACKET_IN* message. Analogously to the previous approach, the controller decides the action (usually output port) and using an *OFPT_FLOW_MOD* message installs a new detailed entry (*A.3*) in the appropriate flow table. Subsequent packets of the flow (*P2*, *P3*) are processed in a similar way and are matched by the newly inserted entry *A.3*. No further involvement of the controller is required.

The presented approach preserves all advantages of its predecessor, the PARD-ST approach, while, additionally, it benefits from distributing flow entries in numerous tables. The advantages of such a solution were addressed in Sect. 1. However, the network node must be capable of supporting multiple tables and *OFPT_GOTO_TABLE* instruction while both are optional according to the OpenFlow 1.3.5 and 1.5.1 specifications. The maximum number of flow tables supported in OpenFlow is limited to 255, requiring the user to sparingly designate tables for aggregates. However, the number of flow entries dedicated to each aggregate and placed in flow tables is limited only by software or hardware capabilities of the SDN switch. In addition, excessive aggregates may be accommodated by placing more than one aggregate in a single flow table.

3.4 Overview of Proposed Solutions

Both PARD-ST and PARD-MT utilize the concept of hybrid reactive-proactive approach to creating new flow table entries and distinction of flow aggregation levels provided by various entries. In this section, some of the related caveats are discussed together with comprehensive example of a use-case for the PARD solution.

3.4.1 Hybrid Reactive-Proactive Approach

In PARD mechanisms, packets belonging to a new flow are sent to the controller and forwarded to the desired output port at the same time. This behavior is caused by the default action defined by the aggregate (coarse-grained) rule. Such an approach allows to combine features of both proactive and reactive traffic handling, instead of having to choose either of them. Therefore, PARD should be assessed in reference to both purely reactive and purely proactive approaches to fully acknowledge its benefits achieved by reaching tradeoff between flexibility and performance of these two.

When the controller decides that a subclass of traffic should be handled in a specific way, a corresponding detailed (fine-grained) flow entry is added that results in forwarding packets without issuing further *OFPT_PACKET_IN* messages. One must note, that the controller is still aware of each new flow arriving in the network and is able to make decisions based on the applicable traffic handling policies. This involves additional controller load caused by processing asynchronous OpenFlow messages. However, the number of *OFPT_PACKET_IN* messages (and increase in the controller's load caused by processing these

messages) is no higher than in case of the basic reference solution (Sect. 3.1) based on a purely reactive approach, therefore the proposed solution is no worse in terms of imposed controller load. Moreover, the increased load (in comparison to a purely proactive approach) is compensated with more flexible management and monitoring of fine-grained network flows. Impact of the *OFPT_PACKET_IN* on the controller performance may be further reduced by deploying an architecture based on multiple redundant controllers. During the evaluation of the proposed mechanisms, we focused solely on latency and packet loss ratio metric, assuming that satisfactory metric values imply proper controller operation under the imposed load. It should also be noted that PARD offers additional path protection in case of controller overload, as the default (coarse-grained) flow table entries may handle traffic (forward packets to the output port) even in the event of controller unavailability.

To better illustrate benefits of the PARD approach, a simple analogy between hybrid reactive-proactive approach and speculative execution mechanism present in modern Central Processing Units (CPUs) may be drawn. The CPU is capable of guessing and following the most likely execution path before the final decision about the execution path is determined. This stems from the fact that access to the system memory is considerably slower than access to the CPU cache and it takes a few hundred CPU cycles to retrieve uncached values from the memory. Instead of just waiting for the preceding operation to finish, the CPU may start prematurely executing the next instructions, following its guess about the expected execution path. This way, idle cycles are utilized to perform calculations that could be useful in the future. When the desired value is finally retrieved from the memory, the CPU is able to verify its guess. If the guess was wrong, results of the operations performed in the waiting time are not useful at all and the CPU returns its state to the previously saved checkpoint. The performance in this case may be compared to the case in which speculative execution was not present in the system at all. However, if the guess was right, some of the operations in the flow have been already performed and may be skipped, resulting in a performance gain [26].

The principles of the hybrid reactive-proactive approach in the PARD solution are similar. When a packet arrives at the SDN switch, its forwarding is not delayed by communication with the controller. The packet is immediately forwarded to the output port instead of remaining idle in the buffer. At the same time, an asynchronous message is also sent to the SDN controller. The controller determines the correct action for the new flow and responds to the switch. If the action determined by the controller differs from the default action applied by the switch, all of the subsequent flow packets are handled in accordance with controller's decision. Previously sent packets are also expected to successfully arrive at the destination, as default actions apply different but correct paths. The only possible side-effect is packet reordering in the initial part of the flow. However, if the controller's decision was consistent with the already applied action, packets have been forwarded properly without any unnecessary delay. This constitutes one of main advantages of the PARD solution over its alternatives.

3.4.2 Choice of Aggregation Level

The distinction between aggregate (coarse-grained) and detailed (fine-grained) flow table entries is the major principle of the PARD concept. The former ones are assumed to be created in a proactive manner, while the latter ones are created reactively.

As an example, a use-case that involves proactive mechanisms creating aggregates from source and destination addresses belonging to the same layer-3 networks may be assumed. The rationale for such a behaviour is that each of those destinations is reachable through the same SDN node. Required topology discovery mechanisms are available in the literature and are out of scope of this work. The aggregates are prepared in the background during normal network operation and in advance of a specified timestamp at which they will be applied (inserted into flow tables). Thus, at the desired time, actions associated with a given aggregate may change. The actions are supposed to be a result of a complex optimization which takes into account, for example, current and predicted traffic demands between network nodes. For an effective prediction, fine grained OpenFlow statistics are needed, thus it is an additional advantage that the proposed methods support gathering those statistics.

Simultaneously, reactive mechanisms dynamically take into account up-to-date network state including failures, unexpected increase of traffic demands, or characteristics of the flow being considered (e.g. distribution of sizes of the first n packets of a new flow). As a result, fine-grained (detailed) entries that cover a part of the traffic belonging to the aggregate may apply different actions than coarse-grained flow entries containing default action for the whole aggregate. This allows e.g. to route regular traffic of a specific subnet via a chosen network node, but use an alternate path for just a group of services available in the subnet.

To further illustrate how aggregation level choice affects flow table contents and applicable traffic policies, two examples are presented in Table 2. Both options differ by granularity of flow entries on each level and, therefore, the number of flow entries generated in each case. In the Option 1 flows are aggregated by IP prefix pairs and IP host pairs, while in the Option 2 flows are aggregated by IP host pairs and 5-tuples. The number of flow entries provided in the table was estimated with assumption that the network traffic is generated by 15 hosts, each 5 hosts in a separate subnet, and each host has 3 outbound http sessions to all other hosts.

A decision on which of the flows should be handled by detailed (fine-grained) or aggregate (coarse-grained) entries is beyond the scope of PARD mechanism which was designed solely as the flow entry provisioning scheme. A variety of use-cases, different than the one presented above, may be developed. In each case, the aggregation level of the entries should be determined on the basis of requirements for the routing policy in a specific use-case. The decision algorithm could use OpenFlow traffic statistics or other data source (e.g. sFlow) to detect elephant flows (flows that convey large volumes of traffic) and handle them in an alternative way, while handling the remaining traffic using the default rule [14]. The detailed rule may be also used to retain a specific processing policy (i.e. routing, queueing) of an existing high-priority flow, while some modifications are introduced to the aggregate entries based on long term offline optimization. Both of these approaches could be

Table 2 Example aggregation levels

	Flow entry type	Aggregation level	Flow entry example	Flow entry count
Option #1	Coarse-grained	IP prefix pair	ip_src=192.168.10.0/24 ip_dst=192.168.11.0/24 ip_proto=* tcp_src=* tcp_dst=*	6
	Fine-grained	IP host pair	ip_src=192.168.10.50 ip_dst=192.168.11.1 ip_proto=* tcp_src=* tcp_dst=*	210
	Both types	–	–	6 + 210 = 216
Option #2	Coarse-grained	IP host pair	ip_src=192.168.10.50 ip_dst=192.168.11.1 ip_proto=* tcp_src=* tcp_dst=*	210
	Fine-grained	5-tuple	ip_src=192.168.10.50 ip_dst=192.168.11.1 ip_proto=6 tcp_src=54433 tcp_dst=80	630
	Both types	–	–	210 + 630 = 840

used by an ISP or a data center operator [19, 25] to ensure constant QoS, prevent disruptions of a chosen traffic class (mass event video stream, VM migration traffic) and still let the other flows to adapt dynamically. For the sake of brevity, a simple approach that installed detailed (fine-grained) entries for all new flows reported in *OFPT_PACKET_IN* messages was implemented.

4 Evaluation

In order to assess and validate the proposed mechanisms, several experiments were conducted in the dedicated environment. The results are presented in this section along with the description of methodology used. In addition, a functional analysis of key features provided by flow path establishment approaches (including PARD-ST, PARD-MT, reference fully-reactive and fully-proactive approach) and discussion on performance caveats related to flow table usage and packet reordering is also included. These provide altogether a thorough overview of enhancements provided by the PARD solution and its main advantages over alternative mechanisms.

4.1 Environment

A virtual network was emulated using efficient and well known Mininet² software running inside a virtual machine. An SDN controller was deployed under the host operating system. Communication between the controller and emulated network nodes was provided using a separate control network and OFP ver. 1.3. Among numerous software switches the recent Open vSwitch ver. 2.9.2³ was chosen. The proposed mechanisms are designed to be implemented in the SDN controller. After a thorough analysis, the Ryu⁴ controller was chosen.

Due to the fact that both the proposed PARD mechanism and reference mechanisms operate in the scope of a single node, a very simple topology was assumed. In the experimental topology, three end hosts were connected to separate interfaces of a single OpenFlow switch. Simulated traffic between end hosts traversed the switch, which was responsible for forwarding packets between input and output ports according to rules (flow entries) installed in its flow tables. The decision to choose such a scenario stems from the fact that in a multi-node network packets are processed by the OpenFlow pipeline in a similar way in each node on their path from source to destination.

The performance of traffic forwarding suffers from latency introduced by mechanisms that determine and apply the desired action for the packet. In this context, the choice of PARD, fully reactive (reference) or proactive approach determines the time necessary to forward the packet just like the choice of CPU, Application Specific Integrated Circuit (ASIC) or Network Interface Card (NIC). The impact of those factors is best observed on a single node, as simple latency measurement between just two ports of the device eliminates external factors present in more complex topologies that could distort the results. Conclusions on how application of a specific flow path establishment mechanism would affect a multi-node, scalable topology may, however, be drawn from the observation of a single node. The presence of coarse-grained flow entries that contain default actions for flow aggregates would result in faster forwarding of initial packets of a new flow regardless of the topology. In case of control plane failures, the coarse-grained flow entries act as a backup to ensure that the traffic is not disrupted. In addition, detailed analysis of *OFPT_PACKET_IN* messages received from multiple nodes across the network may be carried out to verify whether all switches install the new flow entries correctly, and detect any possible issues such as flow table contention. Therefore, PARD is believed to perform no worse than fully reactive or proactive approach. It is expected that overall network performance would benefit from lower path setup times in any topology, while precise performance gains depend on multiple factors such as topology size, simultaneous or sequential flow entry installation on multiple nodes, etc. and therefore their assessment is beyond the scope of this paper.

² <http://mininet.org/>.

³ <https://openvswitch.org/>.

⁴ <https://osrg.github.io/ryu/>.

Fig. 4 The architecture of the proposed application

The following components of applications (Fig. 4), developed on top of the Ryu controller, are specially important:

- *FlowDistributionController* is handling network events (including *EventOFP-PacketIn*) and communication with network nodes.
- *FlowDistributionControllerApi* provides a REST API interface for application configuration.
- *FlowAggregate* represents a user-defined aggregate along with match criteria and actions.
- *FlowPipeline* represents a set of rules to be inserted to a flow table and contains methods to search within those rules.

The environment was properly preconfigured in order to minimize the impact of external factors. Each host had preinstalled static entries in its ARP table. Furthermore, a high priority entry is preinstalled in the flow table to handle responses from the destination node. The rationale is to analyze performance of one way communication.

4.2 Experimental Results: Latency and Packet Loss

Two indicators were used to assess the mechanisms: transmission latency of initial packets in a new flow and packet loss rate for UDP traffic. Such an approach is reasonable as those two factors are directly improved by solutions proposed in this work. However, it is worth to note that both of those parameters further impact throughput, which is commonly used to illustrate performance of the transmission. Namely, initial latency increases the total duration of each transmission while packet loss causes retransmissions in case of protocols with guaranteed packet delivery. Thus, as more time is required to transmit the flow, the overall throughput of transmission deteriorates. It should also be noted that UDP traffic was preferred over TCP in the experiment due to its common usage in real-time, time-critical applications such as online gaming and media streaming. Considering also its application in DNS and DHCP services, RDP or QUIC protocol [27], UDP data streams provide a proper example of traffic vulnerable to increased path setup latency that may benefit from PARD mechanisms presented in the paper. An additional reason for focusing solely on UDP in the experiment was an overhead of TCP traffic related to retransmissions and session establishment that could distort measurement results. However, it is clear that the first packets of the TCP stream, that belong to the 3-way handshake process, would be affected in a same way as packets of a UDP stream. Moreover, increased packet latency observed in the experiment could further decrease throughput of TCP transmission as proven in [28]. This issue is especially burdensome in case of short flows with insufficient time to extend the transmission window.

Three scenarios were evaluated: “Reference”, “PARD-ST” and “PARD-MT”, each of them using mechanisms described in Sects. 3.1, 3.2 and 3.3 respectively. Three variable parameters were introduced in experiments: a number of predefined flow entries ($n = \{1, 100, 200\}$), packet interarrival time ($p_i = \{0.063, 1\}$ [s]) in case of ICMP traffic and transmission rate in case of UDP traffic ($T = \{10, 25, 50\}$ [Mb/s]). The rationale for selecting such values is as follows:

- *The number of predefined flow entries (n)*—The number of flow entries used in the experiment was determined by the fact that in commercially available SDN switches, flow entries may be stored in TCAM due to its high performance [29]. Costs of TCAM modules limit the maximum flow table capacity to only a few thousands of entries [8]. Considering that low-end devices could provide even less TCAM storage, values of ($n = \{1, 100, 200\}$) were chosen. In case of “PARD-MT” scenario, the number of predefined entries n was also the number of flow aggregates and resulted in using ($n_t = \{1, 100, 200\}$) flow tables, respectively. The rationale for choosing such values for n_t was that the maximum number of flow tables specified by OpenFlow is 255. Therefore, $n_t = 1$ is the minimum possible number of flow tables used, while $n_t = 100$ and $n_t = 200$ represent ca. 40% and 80% of available flow tables, respectively. In accordance with schemes presented in Sects. 3.1, 3.2 and 3.3, the “Reference” scenario used only detailed (fine-grained) flow table entries installed reactively, while “PARD-ST”

Fig. 5 Transmission latency of consecutive packets and for different number of predefined entries ($p_i = 1$ s)

Fig. 6 Transmission latency of consecutive packets and for different number of predefined entries ($p_i = 63$ ms)

and “PARD-MT” scenarios used both detailed and aggregate (coarse-grained) flow entries installed reactively and proactively.

- **UDP traffic rate (T)**—Based on the Internet traffic analysis presented in other research, it was assumed that the traffic rate expected from majority of flows is well below the capacity of FastEthernet interfaces configured in the Mini-net emulator [30]. Considering common applications of UDP in both low-rate streams (e.g. DNS queries) and high-rate streams (e.g. media streaming), traffic rates of $T = \{10, 25, 50\}$ [Mb/s] were chosen for the experiment. These resemble 10%, 25% and 50% utilization of FastEthernet link capacity, respectively.
- **ICMP packet interarrival time (p_i)**—The Trace Statistics for CAIDA Passive OC48 and OC192 Traces [31] were used to approximate the mean packet interarrival time in a scope of a single flow. The data from *nyc (dirA)* collected on 2019-01-17 was used as the reference. Using the overall trace statistics, the mean

Fig. 7 Packet loss as a function of the number of entries in flow tables for different traffic rates

packet interarrival time of ca. 63 ms was calculated. In addition, the default 1 s interarrival value of the *ping* tool was also used.

Transmission latency was measured using the unix-based *ping* tool for the first five ICMP packets of a new flow generated with a specified p_i interval. Figures 5 and 6 present the results parametrized with a different number of predefined entries for the interarrival times of 63 ms and 1 s, respectively. In each scenario, the latency of the first packet was significantly higher for the reference approach (reaching up to 10 ms and 45 ms in Figs. 5 and 6, respectively) than for PARD-ST and PARD-MT approaches. For the next packets differences between approaches are negligible and fall below a single millisecond.

The results also allow to assess the latency of the first packet as a function of the number of preinstalled entries (consider the results for the first packet in different subfigures). The most important conclusion is that the number of pre-installed entries does not impact the performance of PARD-ST and PARD-MT mechanisms. However, latency of the first packet in case of the reference scenario depends on the number of predefined entries. It is a result of the fact that the first packet of a new flow must be classified in the controller before a new entry is installed.

A side observation is a slightly higher latency of the second packet for all of the approaches. This results from mechanisms involved in packet classification implemented in Open vSwitch (OvS) architecture and a specific form of flow table management in the assessed approaches. After the initial packet is matched in userspace and examined by the controller, its relevant flow entry is inserted into the flow table. Due to that fact a time-consuming userspace classification has to be applied also for the second packet. However, it allows to create a new entry in kernel cache that significantly improves processing performance of third and subsequent packets.

The loss rate for UDP traffic was measured using a unix-based *iperf* tool. Data streams with a constant duration of 10 seconds and throughput dependent on a simulation scenario were generated between two end devices configured in a client-server

architecture. As datagram lengths remained unchanged throughout all cases, traffic rates were controlled by changing packet inter-arrival times. Figure 7 presents the collected results. The most important observation is a significantly higher loss rate in a reference approach in comparison to the proposed approaches. One of the main reasons for that is packet reordering resulting from a significant latency of the first packet. Also, the number of predefined entries impacts the loss rate in case of the reference scenario. In the most pessimistic case the loss rate was over 0.5% while for the proposed mechanisms the loss rate was never higher than 0.1%.

4.3 Flow Table Usage Analysis

A deployment of the PARD mechanism involves installing flow table entries both reactively and proactively. This results in a slightly higher number of flow table capacity usage to ensure proper handling of network traffic. The difference between initial flow table occupancy in reference mechanisms and in PARD should not, however, result in starvation of memory resources. Each of the cases should be considered separately, depending on intended routing policies and desired configuration of aggregation levels within PARD.

4.3.1 PARD vs Purely-Reactive Approach

In comparison with a purely-reactive approach it is assumed that the controller creates a new flow table entry for each new flow detected and announced by the *PACKET_IN* message. In this case, migrating to PARD would involve creating higher-level aggregate (coarse-grained) flow entries added proactively to support fast forwarding using default actions of the aggregate. The flow table entry overhead introduced by the PARD solution highly depends on the aggregation level chosen for the proactive entries. For most cases, e.g. a detailed flow defined by a single host IP address and an aggregate flow defined by a subnet address, the number of additional flow table entries created by PARD is only a fraction of the initial flow table occupancy. Therefore, the negative impact of PARD on resources and performance should not be expected.

4.3.2 PARD vs Purely-Proactive Approach

When comparing to a purely-proactive approach, it may be assumed that the controller creates a number of higher-level aggregate entries to handle traffic in the default, predefined manner. Proactive creating of fine-grained flow table entries is limited as not all header values may be predicted (e.g. port number, single host IP address) and populating flow table with a series of redundant detailed entries (e.g. containing consecutive port numbers) involves massive overhead. Instead, wildcarded coarse-grained flow entries are installed (a single entry for all hosts in a subnet or a single entry for all ports of a single host, respectively). Migrating to PARD would introduce fine-grained flow entries to be installed dynamically based on traffic detected by the controller. These flow table entries would allow to handle specific flows in

Table 3 Comparison of flow path establishment approach types

Feature/approach	Fully-reactive (reference)	Fully-proactive	PARD-ST	PARD-MT
<i>PACKET_IN</i> on a new flow	Yes	No	Yes	Yes
<i>PACKET_IN</i> until new entry	Yes	No	Yes	Yes
Packet forwarding after <i>PACKET_OUT</i> and/or <i>FLOW_MOD</i>	Yes	No	No	No
Immediate packet forwarding	No	Yes	Yes	Yes
Reactively added rules	Yes	No	Yes	Yes
Proactively added rules	No	Yes	Yes	Yes

a different way or gather accurate statistics of a single flow. Although the most basic approach could result in adding a detailed entry for each flow discovered in the network, the decision algorithm on top of PARD may be adjusted to consider only selected flows based on their volume or a custom-defined priority. As the result the number of additional flow entries made by properly deployed PARD mechanism depends on a use-case and is hardly estimated without making further assumptions. Nevertheless, as flow entry overhead introduced by deploying PARD depends mainly on the adjustments made to the routing policies, it may be tailored to meet both functional and performance requirements.

4.4 Packet Reordering Analysis

As the PARD solution features path changes, rerouting of the flows may cause packet reordering and lead to transient performance degradation due to losses and retransmission. Such an issue will occur if reactive mechanisms prepare a detailed entry that changes the path of a flow being handled by a proactive aggregate. At the time the detailed entry is installed, a few packets of the flow may have already been forwarded using action defined by the coarse-grained flow entry corresponding with the default aggregate.

Although this issue was not the main focus of current research, it may be alleviated by mechanisms implemented in higher-layer protocols and by countermeasures developed in other research [32]. Its severity highly depends on the way traffic aggregates are defined and may be reduced by using properly designed routing strategies that provide some form of consistency between proactive and reactive entries. Moreover, discrepancy between paths defined in default (coarse-grained) and detailed (fine-grained) rules is assumed to occur only in rare circumstances. Reactive mechanisms are expected to choose a different path in case of network failures or in case of congestion, which is infrequent as aggregate entries are prepared based on static network optimization. In these cases, traffic rerouting is imminent and packet reordering is considered to be a trade-off between performance and reliability. Finally, the problem of rerouting in an SDN network may be mitigated using any of the numerous mechanisms, e.g. [33].

4.5 Functional Comparison

In addition to performance metrics presented in the preceding sections, functional features are also important for the proper evaluation of the flow path establishment mechanisms. A few key differences, that should be noted when considering efficiency of a specific approach, have been collated in Table 3. Along with the reactive (reference), PARD-ST and PARD-MT mechanisms evaluated in the paper, the fully-proactive approach was also presented. The features considered in the comparison are:

- *PACKET_IN* on a new flow—indicates whether controller is notified about each new flow by *PACKET_IN* message. This allows the controller to immediately react to dynamic traffic changes in the network. However, the risk of overload caused by huge number of asynchronous messages is involved.
- *PACKET_IN* until new entry—indicates whether *PACKET_IN* messages are sent to the controller until an action (such as adding a new flow entry) is taken. This involves a chance of overload caused by *PACKET_IN* messages.
- Packet forwarding only after *PACKET_OUT* and/or *FLOW_MOD*—indicates whether a controller action (such as *PACKET_OUT* message and adding a flow entry) is required to start handling the traffic of a new flow. In such an approach packets are buffered until an action is determined by the controller. This results in performance deterioration caused by increased path establishment latency.
- Immediate packet forwarding—indicates whether packets may be forwarded before a decision is made by the controller (e.g. based on default flow entries). In such an approach there is no additional latency imposed on packets by communication between the SDN switch and the controller. This results in improved overall performance of traffic forwarding.
- Reactively added rules—indicates whether flow entries are added in a reactive manner.
- Proactively added rules—indicates whether flow entries are added in a proactive manner.

Based on the comparison presented in Table 3, a conclusion may be drawn that PARD mechanisms combine advantages of both reactive and proactive approaches. This comes at a cost of a slightly higher number of flow entries present in the flow table (as discussed in Sect. 4.3). The number may be dynamically adjusted for a specific use-case by properly defining network policies and traffic aggregation levels. In addition, the flow installation mechanism implemented in PARD introduces a tradeoff between performance and traffic management capabilities. While the number of *OFPT_PACKET_IN* messages sent to the controller is higher than in case of the fully-proactive approach and causes increased resource consumption, it remains no worse than in case of the fully-reactive approach and additional features related to fine-grained traffic handling and monitoring are available.

The presented assessment proves that the PARD-ST and PARD-MT mechanisms are superior to the reference one in context of a single SDN node performance. The

impact of controller utilization on the transmission latency is reduced. An improvement in this context becomes even more significant with an increasing number of entries in the flow table. It is, however, important that both proposed mechanisms do not reveal any significant differences in terms of performance.

Therefore, if only an SDN node is able to support multiple tables, the PARD-MT mechanism should be used, taking advantage of the multi-table architecture. The most significant benefit of deploying PARD-MT is storing aggregate (coarse-grained) flow entries in one table and creating detailed (fine-grained) entries in other separate tables. Most hardware SDN switches provide storage for one of its tables in TCAM memory that performs exceptionally well in the process of matching wildcard entries. However, its capacity is limited [29]. The scheme implemented in PARD-MT allows to efficiently use available TCAM storage space for the aggregate entries that use wildcards to match traffic aggregates. In addition, grouping entries related to a single aggregate in one table enables simplified flow table management, as described in Sect. 1.1. Finally, the proposed mechanisms neither affect operation of a network nor its end-users.

5 Conclusions

Two mechanisms proposed in this paper allow to effectively manage entries in flow tables of an SDN switch in order to reduce flow setup latency and enhance routing policy enforcement. In addition to PARD-ST that allows to both quickly handle traffic of a new flow and let the SDN controller introduce more complex routing policies, PARD-MT provides additional benefits by utilizing multiple flow tables (as described in Sect. 4.5).

Both PARD mechanisms fully conform to the SDN concept and the OpenFlow standard. They can be flawlessly deployed in any OpenFlow-compliant network without any need to modify either network devices or the southbound communication protocol. This constitutes one of the most important advantages of the proposed approach over the alternative solutions that require additional changes to the network stack.

The PARD solution may be considered as a framework that manages a process of maintaining flow tables satisfying both performance requirements and fine-grained traffic policy constraints. It utilizes a hybrid proactive-reactive approach that may create rules based on higher-layer components: complex static optimization algorithms, dynamic optimization methods, other network applications, user input—using flow aggregation levels, decision algorithms and policies specified by the network administrator. Ultimately, PARD allows to achieve better network performance by simplifying flow table management, ensuring that routing policy changes will not affect already serviced flows and reducing flow path setup latency for new flows. All of these three areas are considered in our future research plans together with works on incorporating network monitoring features into our solution.

Acknowledgements This work was funded by the Polish National Centre for Research and Development under “SDNRoute: integrated system supporting routing in Software Defined Networks” Project No. LIDER/30/0006/L-7/15/NCBR/2016.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Singh, S., Jha, R.K.: A survey on software defined networking: architecture for next generation network. *J. Netw. Syst. Manag.* **25**(2), 321–374 (2017). <https://doi.org/10.1007/s10922-016-9393-9>
2. Li, S., Han, K., Ansari, N., Bao, Q., Hu, D., Liu, J., Yu, S., Zhu, Z.: Improving SDN scalability with protocol-oblivious source routing: a system-level study. *IEEE Trans. Netw. Serv. Manag.* **15**(1), 275–288 (2018). <https://doi.org/10.1109/TNSM.2017.2766159>
3. Kreutz, D., Ramos, F.M.V., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. *Proc. IEEE* **103**(1), 14–76 (2015)
4. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: DevoFlow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.* **41**(4), 254–265 (2011)
5. He, K., Khalid, J., Das, S., Gember-Jacobson, A., Prakash, C., Akella, A., Li, L.E., Thottan, M.: Latency in Software Defined Networks: Measurements and Mitigation Techniques. In: *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2015)*, pp. 435–436. Portland, OR, USA (2015)
6. Wang, Y., Zhong, Q., Qiu, X., Li, W.: Resource allocation for reliable communication between controllers and switches in sdn. *J. Netw. Syst. Manag.* **26**(4), 966–992 (2018). <https://doi.org/10.1007/s10922-018-9450-7>
7. Hussain, M., Shah, N.: Automatic rule installation in case of policy change in software defined networks. *Telecommun. Syst.* **68**(3), 461–477 (2018). <https://doi.org/10.1007/s11235-017-0404-2>
8. Nguyen, X.N., Saucez, D., Barakat, C., Turetti, T.: Rules placement problem in openFlow networks: a survey. *IEEE Commun Surv Tutor* **18**(2), 1273–1286 (2016)
9. Borylo, P., Lason, A., Rzas, J., Szymanski, A., Jajszczyk, A.: Fitting green anycast strategies to cloud services in wdm hybrid power networks. In: *2014 IEEE Global Communications Conference*, pp. 2592–2598 (2014)
10. Rzas, J., Borylo, P., Lason, A., Szymanski, A., Jajszczyk, A.: Dynamic power capping for multi-layer hybrid power networks. In: *2014 IEEE Global Communications Conference*, pp. 2516–2522 (2014)
11. Dias de Assunção, M., Carpa, R., Lefèvre, L., Glück, O., Boryło, P., Lasoń, A., Szymański, A., Rzepka, M.: Designing and building SDN testbeds for energy-aware traffic engineering services. *Photonic Network Communications* (2017)
12. Borylo, P., Lason, A., Rzas, J., Szymanski, A., Jajszczyk, A.: Green cloud provisioning through-out cooperation of a wdm wide area network and a hybrid power it infrastructure. *J. Grid Comput.* (2015). <https://doi.org/10.1007/s10723-015-9354-7>
13. Callegati, F., Cerroni, W., Contoli, C., Cardone, R., Nocentini, M., Manzalini, A.: Sdn for dynamic NFV deployment. *IEEE Commun. Mag.* **54**(10), 89–95 (2016)
14. Guo, Z., Xu, Y., Liu, R., Gushchin, A., Yin Chen, K., Walid, A., Chao, H.J.: Balancing flow table occupancy and link utilization in software-defined networks. *Fut. Gener. Comput. Syst.* **89**, 213–223 (2018). <https://doi.org/10.1016/j.future.2018.06.011>
15. Jia, X., Jiang, Y., Guo, Z., Wu, Z.: Reducing and balancing flow table entries in software-defined networks. In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pp. 575–578 (2016). <https://doi.org/10.1109/LCN.2016.96>

16. 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.* **92**, 300–315 (2015). <https://doi.org/10.1016/j.comnet.2015.09.030>
17. Rifai, M., Huin, N., Caillouet, C., Giroire, F., Lopez-Pacheco, D., Moulrierac, J., Urvoy-Keller, G.: Too many sdn rules? compress them with minnie. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–7 (2015). <https://doi.org/10.1109/GLOCOM.2015.7417661>
18. Kosugiyama, T., Tanabe, K., Nakayama, H., Hayashi, T., Yamaoka, K.: A flow aggregation method based on end-to-end delay in sdn. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6 (2017). <https://doi.org/10.1109/ICC.2017.7996341>
19. Zehua Guo, Shufeng Hui, Yang Xu, Chao, H.J.: Dynamic flow scheduling for power-efficient data center networks. In: 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), pp. 1–10 (2016). <https://doi.org/10.1109/IWQoS.2016.7590399>
20. Iyer, A.S., Mann, V., Samineni, N.R.: Switchreduce: Reducing switch state and controller involvement in openflow networks. In: 2013 IFIP Networking Conference, pp. 1–9 (2013)
21. Sanabria-Russo, L., Alonso-Zarate, J., Verikoukis, C.: Sdn-based pro-active flow installation mechanism for delay reduction in iot. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2018). <https://doi.org/10.1109/GLOCOM.2018.8647382>
22. Khalili, R., Despotovic, Z., Hecker, A.: Flow setup latency in sdn networks. *IEEE J. Select. Areas Commun.* **36**(12), 2631–2639 (2018). <https://doi.org/10.1109/JSAC.2018.2871291>
23. Ahmed, D., Dawood, A., Rafey, M., Hussain, M.: Flow Installation Mechanisms in SDN (2018). <http://www.uow.edu.pk/ORIC/Publications/4th%20MDSRIC-264.pdf>
24. Yi, T., Li, H.: Flow-split: An approach to reduce flow establish time and invoking of controller in OpenFlow networks. In: Proc. IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC 2016), pp. 241–245. Chongqing, China (2016)
25. Hung, C.H., Huang, C.W., Wang, L.C., Chen, C.: Scalable topology-based flow entry management in data center. In: Proc. 13th IEEE Annual Consumer Communications Networking Conference (CCNC 2016), pp. 85–90 (2016)
26. Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1–19 (2019). <https://doi.org/10.1109/SP.2019.00002>
27. Murray, D., Koziniec, T., Zander, S., Dixon, M., Koutsakis, P.: An analysis of changing enterprise network traffic characteristics. In: 2017 23rd Asia-Pacific Conference on Communications (APCC), pp. 1–6 (2017). <https://doi.org/10.23919/APCC.2017.8303960>
28. Guo, Z., Liu, R., Xu, Y., Gushchin, A., Walid, A., Chao, H.J.: Star: Preventing flow-table overflow in software-defined networks. *Comput. Netw.* **125**, 15–25 (2017)
29. Xiong, B., Wu, R., Liao, Z., Wang, J.: Efficient differentiated storage architecture for large-scale flow tables in openflow networks. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 742–749 (2019). <https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00110>
30. Zhang, X., Ding, W.: Comparative research on internet flows characteristics. In: 2012 Third International Conference on Networking and Distributed Computing, pp. 114–118 (2012). <https://doi.org/10.1109/ICNDC.2012.35>
31. CAIDA UCSD: The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces (2019). https://www.caida.org/data/passive/passive_trace_statistics.xml
32. Leung, K.K., Li, V.O., Yang, D.: An overview of packet reordering in transmission control protocol (tcp): Problems, solutions, and challenges. *IEEE Trans. Parallel Distrib. Syst.* **18**(4), 522–535 (2007). <https://doi.org/10.1109/TPDS.2007.1011>
33. Aibin, M., Walkowiak, K., Sen, A.: Software-defined adaptive survivability for elastic optical networks. *Optic. Switch. Netw.* **23**, 85–96 (2017). <https://doi.org/10.1016/j.osn.2016.06.008>

Michał Rzepka is a PhD student at the Department of Telecommunications, AGH University of Science and Technology, in Krakow, Poland. He received the MSc degree in ICT from the same university in 2017. His area of research covers Software Defined Networking, traffic engineering and network measurements. His interests also include cloud computing and mobile technologies.

Piotr Boryło works at the AGH University of Science and Technology. Received M.S. and Ph.D. degrees in Telecommunications in 2012 and 2016, respectively. Interested in traffic engineering and resource provisioning in Software Defined Networks, for the purpose of cloud services, energy efficiency, 5G, data centers, fog computing. Static and dynamic optimization are within the scope of his interests.

Artur Lason is an assistant professor at AGH University of Science and Technology, Poland. He received M.Sc. and Ph.D. degrees from AGH University in 1992 and 1999 respectively. Currently his main research interests are focused on resource management in Software Defined Networks; virtualization of network functions; privacy and security issues in fog/cloud computing environments.

Andrzej Szymanski received the M.Sc. degree in electrical engineering and the Ph.D. degree in telecommunications from the University of Science and Technology, Krakow, Poland, in 1999 and 2009, respectively. In 1999, he joined the Department of Telecommunications at the University of Science and Technology. His research interests include control plane issues of both packet networks and switched optical networks.