Dynamic Flow Rules in Software Defined Networks

Qing Wei, David Perez-Caparros, Artur Hecker
Huawei Technologies - European Research Center, Riesstrasse 25,80992 Munich, Germany
{qing.wei, david.perez.caparros, artur.hecker}@huawei.com

***Abstract*—Software Defined Networking (SDN) architecture enables centralized control of the forwarding behavior of individual network elements. While SDN brings many well-known benefits, such as manageability and adaptability, it also poses some challenges. Scalability becomes an issue in highly dynamic, large scale networks, where the forwarding rules of single elements must be updated at a high pace by a central controller. This work proposes a novel type of flow rule to tackle this issue, the Dynamic Flow Rule (DFR). DFR enables the network elements to change their forwarding behavior locally, according to predefined instructions set up by the central controller. This paper introduces the DFR concept, discusses several implementation options and examines its performance in different use cases. The performance analysis shows that DFR effectively increases the programmability and adaptability of SDN network. It reduces the control plane signaling, reduces the network reaction time to changes, and alleviates the computing requirements at the controller, while retaining the central control of the network. And most importantly, since DFR leverages on the current processing capability of SDN switches, it provides a general and scalable control solution without additional performance penalty.**

I. Introduction

Mobile communication industry is facing the challenge to satisfy the continuously increasing network capacity requirements due to the fast growth of mobile traffic. Meanwhile, the next generation mobile network needs to provide diverse networking services (e.g., ultra-low latency, massive connectivity, etc.) for different use cases. This pushes the mobile communication industry to seek for solutions towards a flexible 5G system [1]. Software Defined Networking (SDN) is considered as one of the foundations of 5G infrastructure platform innovation [2]: it decouples the network control and forwarding planes, and makes the network programmable from a logically centralized controller according to different service requirements.

The SDN controller can host many control applications that decide the forwarding behaviour of the underlying infrastructure. The desired behaviour is installed as flow rules in the network forwarding elements (i.e. SDN switches) by the controller. Today, the most widely used protocol to perform this kind of reconfiguration in SDN switches is OpenFlow [3]. Except for the group table, the SDN switch maintains a chain of flow tables (pipeline processing). Flow rules are specified in the forwarding table as table entries that include match fields and actions. An incoming packet is processed by the table pipeline according to the matching fields and actions. The set of possible actions include forwarding the incoming packet to an output port, modifying fields in the packet header, redirecting the packet to another forwarding table for further processing, etc. After the flow rules are specified by the controller, they are statically installed at the SDN switches. If a flow rule needs to be modified (e.g. change matching fields or actions), the controller needs to interact with the given switch to reconfigure it. In case the matching condition of a packet changes (e.g. packet arriving at a different port in the switch due to node mobility), if there are no flow rules installed for this new condition, the switch sends the incoming packet to the controller wrapped into an SDN control message (PACKET_IN[3]). The controller analyses this packet and updates the existing flow rule accordingly. Such round trip of control signaling becomes signaling overhead. In a network with many mobile nodes (e.g., a LTE base station has 3-6 cells, where each supports 200 active mobiles, and a network could have 100,000+ base stations), the control signaling due to mobility events could generate congestion of the control channel and huge processing load at the controller [4].

This work proposes a novel type of flow rule - Dynamic Flow Rule (DFR) - to address the above challenges. The performance of DFR is examined in different use cases and shows that DFR effectively increases the programmability and adaptability of an SDN network. Particularly, DFR exhibits three unique, exclusive features:

- It is the first proposal to address the scalability issue of SDN by proposing a new type of flow rule.
- It does not require a local control agent in the switch, as it leverages on existing processing capabilities. Therefore, it avoids both architectural debates and the performance vs. costs tradeoffs.
- DFR implementation can be adapted to the processing capabilities of the target SDN switch; it therefore can be used in a hybrid SDN environment.

These features enable DFR to provide a simple and general, yet scalable control solution.

This paper is organized as follows, Section II discusses related work. Section III introduces the concept of DFR. We exemplify the usage of DFR in different use cases in Section IV, and shows that DFR can adapt its implementation according to the actual processing capability of the SDN switches in Section V. The performance of DFR is examined and discussed in Section VI. Finally, Section VII presents the conclusions.

II. Related Work

The advantages of logically centralizing the control intelligence of the network in a controller are well-known [5]. However, it also presents some drawbacks that are often

neglected, especially when applied to large scale, highly dynamic networks, such as a country-wide mobile networks. The burden of determining, installing and reconfiguring flow data paths for millions of tracked mobile devices may exceed the computation capabilities of a central controller. In this scenario, the performance limitations of the current SDN architecture can be classified in three major areas:

- **Scalability**. The performance of the SDN controller is closely related to the number of flow data paths to be computed, installed and managed in parallel;
- **Latency**. The response time to changes in the network topology depends on the distance between the controller and SDN switches and the amount of simultaneous events to be processed;
- **Resiliency**. Since the controller is the brain of the SDN network, resilience of the controller itself and the control channels from switches to the controller greatly affects the correct operation of the complete SDN network.

In the literature, there are many proposals to overcome the limitations related to the three areas mentioned above.

Some works propose a distributed control architecture to overcome such challenge. Authors in [6] propose a hierarchical control architecture at the expense of increasing the synchronization signaling between the distributed controllers. Some approaches focus on agent-based solutions, where a local agent in each SDN switch takes over some of the tasks from the applications on top of the central controller [7]. However, agent based approaches require additional application layer processing capabilities at the SDN switch, which might not always be feasible.

Other works try to introduce more flexibility by enriching the flow rules. The P4 language specification [8][9] defines the concept of action profile that enables several entries in the forwarding table to share the same action set. Action profiles can also be dynamically bound to a match entry by using an action selector. The P4 action selector chooses a particular action profile entry for each packet by either pseudo-randomly or deriving the decision from header fields and/or meta data. While this option brings dynamicity to the forwarding tables of the SDN switches, the set of actions from which the action selector can choose is still operating only on the data plane (e.g. modify packet header, forward to next forwarding table, etc.). In contrast, as will be explained later, DFR proposes the operations on the control plane (e.g. install new flow rule, flow rule association, etc.) instead.

The *learn* action in Open vSwitch [10] allows modifying an existing flow table based on the content of the flow currently being processed at the switch. since it is a special kind of action based on a set of logical operations (e.g. store/load values in a custom internal switch registry), its usage is limited to certain type of switches like Open vSwitch. Meanwhile, the *learn* action can only operate on flow rules that were produced by the same action. As it will be discussed in the following sections, DFR does not have such limitations.

OpenState [11] implements a Finite State Machine (FSM) at the SDN switch. The switch performs different actions (e.g. reconfiguring the forwarding tables) based on the local state at the switch. DFR can be seen as a simplified FSM with only two states (”new” or ”old”). DFR has different implementation possibilities and does not require any state table.

III. Dynamic Flow Rule

A. Concept

Dynamic Flow Rule (DFR) is a special type of flow rule based on the current OpenFlow specification. Similar to OpenFlow defined flow rules, which we call Static Flow Rules (SFR) in this paper, DFR consists of two main elements: match fields and instruction set. The difference with SFR is represented in the following three aspects (see Figure 1).

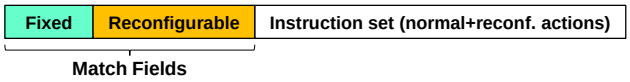


Fig. 1. Dynamic Flow Rule structure

1) Match fields: In contrast to the fixed match fields in SFR, DFR includes a new type of match fields, the reconfigurable match fields. The reconfigurable match fields follow the same format as the fixed match fields as defined in the current OpenFlow specification. Their initial value is also specified by the controller. However, different to the fixed match field, the reconfigurable match field value can be changed by the SDN switch itself by invoking the so called matched reconfigure actions.

2) Instruction set: The DFR instruction set includes data plane actions, as currently specified in OpenFlow, e.g. forward the packet to a port, modify packet by decrementing the time-to-live field, etc. In addition, DFR features reconfigure actions, which act also on the forwarding tables of the SDN switch, e.g. modification of the reconfigurable match fields, or modification/generation of an associated flow rule. The SDN switch triggers reconfigure actions, when a change in the current value of the reconfigurable match fields is detected.

3) Rule association: A DFR can have an associated flow rule e.g. the downlink flow rule belonging to a mobile node can be associated with its uplink flow rule. A new generated flow rule can be associated with the DFR that originated it. Such associated flow rule can be changed or installed as specified in the reconfigure action of the DFR. The associated flow rule can be either a SFR or a DFR. In SFR, there is no explicit rule association concept but a rule identification method that uses the cookie field of the flow entries. For instance, controller may use such cookie field to identify a certain group of flow rules. In contrast, in DFR, the flow rule association provides a hierarchical flow rule relationship chart: e.g. the generating DFR is the parent of the generated flow rule. Such semantic associations of flow rules facilitate the local control operations at the SDN switches, such as deleting/changing the child flow rule when the parent rule is deleted/changed.

B. Procedures and Mechanisms

The SDN controller defines the DFRs and installs them in the target SDN switches. This section explains the related procedures to prepare, install and execute DFRs.

1) Rule Specification: The controller specifies the contents of a DFR based on the requirements of the requested connectivity service (by some internal component or via its northbound interface). For instance, if there is a request for supporting end hosts mobility, the controller may leverage on DFR to reduce the control plane signaling that the mobility events will generate. The controller sets the boundaries of what a given DFR can configure in the target switch by defining its fixed and reconfigurable match fields, optional normal actions, reconfigure actions and the associated flow rules, if applicable.

2) Rule Installation: Depending on the type and packet processing capabilities of the target switch, a given DFR might be installed using different approaches (see section V). Therefore, a DFR might need to be converted into a usable set of flow installation instructions according to the target SDN switch. This conversion can be done at the controller, or even directly at the target SDN switch, with the support of some kind of local agent.

3) Rule Execution: After being installed at an SDN switch, the DFR is executed in the following manner when an incoming packet is being processed:

- If there is a match with both fixed and reconfigurable match fields values, the normal actions in the instruction set are applied.
- If the match is only with the fixed match fields(methods to perform such partial matching is explained in section V), the reconfigure actions (and optionally the normal actions) are applied.
- If there is no match with the fixed match fields, the DFR flow entry is considered as a miss and no action is applied for such entry.

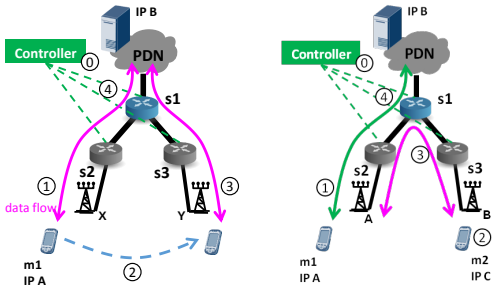
In case the controller explicitly asks for enabling the DFR reporting option, after a reconfigure action is applied, the switch will send a notification to the controller with details on the result of such action and the affected forwarding tables. This feature makes the controller aware of any change in the switch forwarding tables at the cost of increasing the signaling traffic switch-controller. The controller, once notified, can either overrule the locally installed rules at the switch or perform further optimization, e.g., enrich the rule by redirecting the traffic in corresponding queue.

IV. Use Cases

As discussed in previous sections, DFR enables self-adaptation at individual network elements, which proves quite useful in many real world use cases. This feature is crucial in highly dynamic environments, where the flow table of an SDN switch needs to be frequently reconfigured due to changes in the network. In this section, we use two examples (mobility management and traffic bridging) to describe how the DFR concept can be applied in different scenarios.

A. Mobility Management

The mobility management is the major function in a mobile network; in a nutshell, it ensures that services are properly delivered to subscribers, independently of the changes in their location. Here, we show how DFR can be used to automatically reconfigure the existing flow paths of user devices (UE) after a mobility event directly at the SDN switch. Herein, DFR not only reduces the load at the controller, specifically when many mobility events happen simultaneously, but also reduces the flow setup latency. Taking as a reference the scenario shown in Figure 2a, a procedure using DFR to handle mobility events in the data plane is explained below. Here $s2$, $s3$ are SDN switches with general forwarding rules (e.g., tag the data packets from the wireless port and forward to the wired port).



(a) Mobility Management (b) Traffic Bridging

Fig. 2. Use cases

1) Step 0: The controller installs a DFR $\#A$-B in the SDN switch $s1$ to handle traffic between $m1$ with IP address A and a service located in an external Packet Data Network (PDN) with IP address B. In practice, B can also be the address of the LTE packet gateway (PGW). Such DFR is installed in $s1$ during the configuration process of the attachment procedure of $m1$. A flow rule matching downlink traffic for $m1$ (IP $B \rightarrow$IP A), identified as $\#B$-A, is also installed in $s1$. The DFR is specified as shown in Figure 3.

DFR ID: **#A-B**

src: **IP A** dst: **IP B**	in_port: **s2**	Normal action: **output => PDN** Reconf. action: **update flow #A-B (new *in_port*)** **update flow #B-A (new fwd *action*: output => new *in_port*)**

Fig. 3. DFR $\#A$-B mobility management

2) Step 1: Incoming packet from $m1$ arrives to $s1$ [IP src: A, IP dst: B, in_port: $s2$]. The packet is forwarded to the PDN, as specified in the flow table of $s1$ [out_port $=>$ PDN].

3) Step 2: Now, $m1$ moves to a new location ($X \rightarrow Y$) and reattaches to a neighbour access point.

4) Step 3: The incoming packet in $s1$ from $m1$ has [IP src: A, IP dst: B, in_port: $s3$]. The installed DFR in $s1$ detects that the input port for the flow IP $A \rightarrow$ IP B has changed and the reconfigure action is executed. The DFR updates the current flow rule to match the tuple [IP src: A, IP dst: B, in_port: $s3$] and also the associated flow rule $\#B$-A with the action [out_port $=>$ $s3$].

5) Step 4: $s1$ reports the flow rule update to the controller.

B. Traffic Bridging

One other use case for DFR is enabling local bridging at the SDN switch. This can be quite useful for cases like M2M communication. In the scenario depicted in Figure 2b, $m2$ wants to communicate with $m1$ which has an established data path to PDN. $s1$ is the anchor point for both $m1$ and

$m2$. A DFR (see Figure 4) is used at $s1$ to automatically set up a new flow path between $m1$ and $m2$ without the intervention of the controller at the moment of traffic arrival. The detailed procedure is similar to the Mobility Management case as shown in Figure 2a.

DFR ID: **#A-B**

dst: **IP A**	src: **IP B** in_port: **PDN**	Normal action: **output => s2** Reconf. action: **install flow rule #C-A with new values for *src* and *in_port* match fields and action output => s2**

Fig. 4. DFR $\#A$-B traffic bridging

C. Discussion

In both use cases above, by applying the same set of principles as specified in Section III, the DFR concept allows to effectively decouple the actually critical data plane event treatment (Step 3) from the controller update (Step 4). Localizing the treatment of time-critical events (e.g. handovers) is the key concept that DFR brings to achieve much better performance at the controller.

V. Implementation

Depending on the capabilities offered by the target SDN switch, there are alternative DFR implementations. We assume that the controller is capable of decomposing a DFR into a set of flow rule installation messages that are recognized by the target SDN switch, and the SDN switch is able to install another rule according to a rule action.

A. Priority-based Approach

One option for DFR implementation in the SDN switch is to leverage on flow rule priorities. Priority-based implementation installs DFRs as two flow entries with different priorities: the high priority rule matches all the specified matching fields, including both fixed and reconfigurable match fields; with a lower priority than the previous entry, the second rule only matches the fixed match fields. It includes a reconfigure action that installs or modifies the specified associated flow rules with the values of the reconfigurable match fields that were not matched by the high priority flow rule.

The DFR in use case shown in Figure 2a can be implemented as two flow rules (e.g., $\#A$-B-$1/2$) with different priorities as follows:

Flow ID	Prio	Match fields	Instruction set	
			Normal Action	Reconfigure Action
#A-B-1	1	src: IP A dst: IP B in_port: s2	output => PDN	
#A-B-2	2	src: IP A dst: IP B in_port: ANY	output => PDN	Modify flow #A-B-1, new match: in_port: [= #A-B-2 in_port] Modify flow #B-A, new action: output => [= #A-B-2 in_port]
#B-A	3	src: IP B dst: IP A	output => s2	

Fig. 5. DFR #A-B priority-based

Flow entry $\#A$-B-2 will only be matched, when the input port changes with regard to $\#A$-B-1's. As action, the input port in $\#A$-B-1 is updated to the new value registered by $\#A$-B-2 and, at the same time, the associated flow entry $\#B$-A is updated with the new value as output port.

B. Pipeline-based Approach

A DFR implementation can also leverage on OpenFlow pipeline processing [3]. In this case, the fixed match fields are matched by a flow rule installed in the first flow table, as action, the packet is forwarded to the second table for further processing. In this second table, the reconfigurable match fields are matched following the same approach described in V-A. Pipeline based implementation separates the fixed match fields and reconfigurable match fields into different tables. Therefore, it can be used to improve the performance of an SDN switch by moving the flow rule (re)configuration actions to a different table. Using this implementation option, the DFR in use case shown in Figure 2a could be implemented as a pipeline of flow tables as shown in Figure 6.

Table 1

Flow ID	Prio	Match fields	Instruction set	
			Normal Action	Reconfigure Action
#A-B	1	src: IP A dst: IP B	Go to Table 2	
#B-A	2	src: IP B dst: IP A	output => s2	

Table 2

Flow ID	Prio	Match fields	Instruction set	
			Normal Action	Reconfigure Action
#A-B-1	1	in_port: s2	output => PDN	
#A-B-2	2	in_port: ANY	output => PDN	Modify flow #A-B-1, new match: in_port: [= #A-B-2 in_port] Modify flow #B-A in Table 1, new action: output => [= #A-B-2 in_port]

Fig. 6. DFR $\#A$-B pipeline-based

C. Parallel Matching Approach

DFR can also be implemented as a single flow entry, in case the SDN switch supports matching different subsets of match fields in parallel (e.g. using multiple matching lines of TCAM [12]). In this case, the switch explicitly differentiates matching results of fixed and reconfigurable match fields and applies the appropriate type of action.

VI. Performance and Discussions

This section discusses the performance of DFR with respect to signaling overhead, latency, number of required flow entries at the SDN switches and also some other design considerations. Numerical analysis compares DFR with SFR in mobility management and local traffic bridge use cases.

A. Flow Setup Latency

The response latency of an SDN network to a dynamic event, i.e. the overall flow setup latency, T consists of 4 parts: the transport latency T_{Trans}, the queuing time of the PACKET_IN (PI) at the controller T_{Queue}, the processing latency at the controller $T_{ProcCtrl}$, and the flow installation time at switches T_{Conf}. T_{Trans} is the switch to controller round trip time of the SDN control plane signaling. $T_{ProcCtrl}$ is the time to process a PI and to generate the PO and FM messages. In case several SDN switches need to be configured for the same flow path, FM messages will follow a certain order in order to avoid transient inconsistencies in the network. Therefore, T_{Conf} is lower bounded by the configuration/processing time at an individual SDN switch

T_{ProcSW} and upper bounded by ρT_{ProcSW}, where $\rho \geq 1$ is the number of communication rounds between controller and switch to configure one flow path.

For this analysis, we assume optical fiber transport, proportional transport latency to the distance, the PO sent together with FM, and that any SDN switch can be an anchoring point. With this, the response latency in DFR case can be calculated as: $T = 2D_{ctrlsw}/S_{opt} + T_{Queue} + T_{ProcCtrl} + T_{ProcSW}$, where D_{ctrlsw} is the distance between the controller and the target SDN switch, $S_{opt} = 2\times10^8$ m/s is the transport speed of optical fiber. Figure 7 compares the response latency for DFR and SFR in the mobility and traffic bridging cases. Assuming $T_{ProcCtrl}$ together with T_{Queue} is $1 \sim 2ms$, T_{ProcSW} is comaprable to the normal flow installation time - 10ms [13], with $D = 500km$ between the controller and access SDN switch, the response latency can be reduced by around 7 ms in the mobility case and 4.5 ms in the local traffic bridging case. As shown in Figure 7, when configuring multiple switches for one flow path, SFR may double the response latency even with an optimized configuration sequence (i.e., $\rho = 3$ [14]). In contrast, DFR is not affected due to its distributed decision, and in turn achieves more than 70% of latency saving. In case of none ideal, out-of-band control channel (e.g. multi-hop, in-band signaling, congestion, etc.), further savings can be expected.

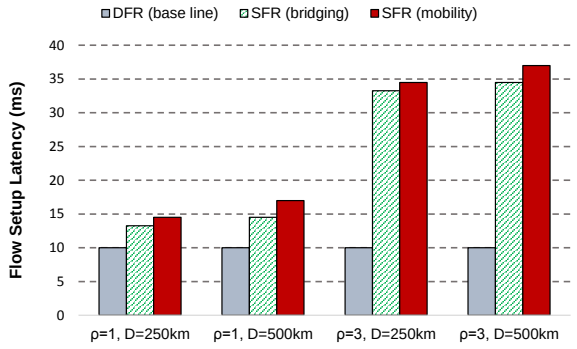


Fig. 7. Comparison of response latency

B. Signaling Overhead

The signaling overhead refers to PI, PO and FM messages between the controller and the SDN switches. Therefore, $OH = (P_{PI} + P_{PO} + N_{sw}P_{FM})T_i\lambda$, where λ is the arrival rate for controller event (e.g. changes of traffic), T_i is the evaluation time period, N_{SW} is the number of concerned SDN switches per control event and P_{PI} ,P_{PO}, P_{FM} denote the payload sizes of PI, PO and FM messages respectively.

For simplicity, we assume $N_{SW} = 1$ in the calculations below. Since the flow path of one mobile node only needs to be configured once as far as it is attached to the same access SDN switch, the overall signaling overhead can be reduced by up to $\frac{\lambda T_i}{1+\lambda T_i}$. In local traffic bridging case, the saving in overhead is very similar. The only difference is that λ in local traffic bridge case depends on the traffic pattern between the users instead of individual user mobility. Even if the SDN switch needs to report to the controller for each local change, the number of signaling messages between the controller and the SDN switch can still be decreased by 67% for each event.

C. Controller processing load and end to end latency

As discussed in the above sections, DFR decouples the local adaptation of data plane forwarding from the controller decision process, and eliminates the unnecessary signalling to the controller. These help to greatly reduce the processing load at the controller and end to end latency at the data plane. We modeled the DFR concept and ran a set of simulations to analyze its performance. Figure 8 and Figure 9 compare the queue length at the controller, and the end to end packet latency in the mobility case assuming a tree topology and 10ms flow setup latency. We change the cluster size c (number of access points connected to one SDN switch), number of mobile UEs u and number of hierarchical levels l. Each UE randomly generates flows with the flow length between 1 to 10 packets, and moves to another access point after transmitting a packet. Figure 8 shows that DFR effectively reduce the controller processing load by ca. 25%. The mean end-to-end packet delay is reduced by 30-40% for a cluster size between 2 and 4, as shown in Figure 9.

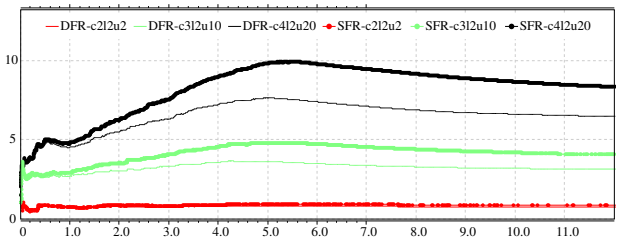


Fig. 8. Controller Queue Length

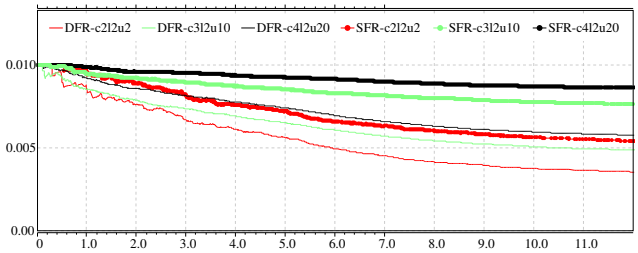


Fig. 9. End to End Packet Latency (s)

D. Number of flow entries

The forwarding performance of SDN switches is antiproportional to the number of installed flow entries N_{Fe}. $N_{Fe} = N_{FeU} + N_{FeD}$, where N_{FeU} is the number of flow entries for upstream data flows, and N_{FeD} is the number of flow entries for downstream data flows. In the mobility case, $N_{FeU} = 1$, assuming all the upstream mobile traffic share the same flow rule. $N_{FeD} = \rho N_{PortD}$ assuming SFR and preconfigured flow path. Here N_{PortD} is the number of downstream logical ports of the access SDN switch and $0 < \rho < 1$ is the aggregation ratio of logical ports to flow entries. For DFR, N_{FeD} is independent of N_{PortD}. It is always 2 for priority-based matching implementation and 1 for parallel matching

implementation, as explained in section V. The savings in flow entries can be easily calculated. In case N_{PortD} = 3, mobility case needs only $75\%/50\%$ of flow entries compared to pre-configured flow entries using priority based/parallel matching implementation. In the traffic bridging case, the savings depend not only on the network topology (i.e. the number of ports of the anchoring node), but also on the traffic pattern (number of flows sharing the same destination/source). Therefore, $N_{FeU} = \rho N_{EntryU}$, $N_{FeD} = \rho N_{EntryD}$, where N_{EntryD}, N_{EntryU} is the number of downlink/uplink flow entries, and $0 < \rho < 1$ is the reusability ratio of flow entries for local bridging.

E. Other design considerations

In both use cases, the computation load is moved from the controller to the SDN switches, where DFR is installed. In the traffic bridging case, the computation complexity is further reduced, because the controller does not need to find the most suitable anchoring SDN switch anymore. The SDN switches at the topology aggregation point can self recognize the need for anchoring by using DFR.

Note that in contrast to the previous work, which also delegated some tasks to the switches [15], [16], in DFR, the reconfiguration privilege of an SDN switch is always authorized in advance by the controller, and the flow abstraction is preserved. Triggers, reconfiguration actions and fields in certain flow rules that can be reconfigured are all predefined by the controller, using essentially the same OpenFlow messages with the same syntax. The controller can therefore ensure the network consistency and synchronization by the appropriate definition of DFR. The local changes will be reported to the controller, so that the controller is still aware of the complete network configuration and is able to change the DFR at any time; however the reporting is different to the reactive flow setup, because it is decoupled from the actual critical event. All in all, DFR does not break the centralized design philosophy of the SDN architecture.

VII. Conclusion

This paper proposes an SDN model-conforming strictly flow rule-based mechanism for delegation of functionality to SDN switches, which achieves controller offloading and a certain flexibility at the SDN switches by adapt their forwarding behaviour locally. A special type of flow rule (Dynamic Flow Rule) is specified, based on the existing OpenFlow specification. Different from current fixed rules, a DFR includes two different types of match fields (fixed and reconfigurable), novel reconfigure actions and associated flow rules. This enables DFR to "modify itself" and/or to generate/modify associated flow rules, thus providing an embedded control architecture in the flow rules to distribute the control handlers from the logical centralized controller to the local SDN switches.

This paper describes the detailed procedure to prepare and process a DFR, exemplifies three possible implementations of DFR (Priority, Pipeline and Parallel matching based implementations) and shows that DFR can be applied to different kinds of SDN switches according to their processing capability. The performance of DFR is examined with respect to latency, signaling overhead and flow table size in two use cases (user equipment mobility management and traffic bridging). The results show that DFR increases the programmability of SDN switches and improves the efficiency of the network, especially in large scale, dynamic environments. With a small extension of the current OpenFlow specification, DFR is able to provide a simple, general and scalable control solution with no data plane performance penalty.

As future work, we plan to implement DFR in Open vSwitch to prove its performance gain with respect to existing solutions. We will also evaluate its efficiency to offload controller processing in realistic mobile network environments.

VIII. Acknowledgement

This work has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No. 671551.

References

[1] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and D. Soldani, "SDN-based 5G mobile networks: architecture, functions, procedures and backward compatibility," *Trans. Emerging Telecommunications Technologies*, vol. 26, no. 1, pp. 82–92, 2015.

[2] A. Hakiri and P. Berthou, "Leveraging SDN for The 5G Networks: Trends, Prospects and Challenges," *CoRR*, vol. abs/1506.02876, 2015.

[3] "OpenFlow switch specification v1.5.1," April 2015. [Online]. Available: https://www.opennetworking.org/sdn-resources/technical-library

[4] Oracle Corporation, "The New Diameter Network: Managing the Signaling Storm," December 2013.

[5] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Palo Alto, CA, USA, White paper, Apr. 2012.

[6] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A Hybrid Hierarchical Control Plane for Flow-Based Large-Scale Software-Defined Networks," *Network and Service Management, IEEE Transactions on*, vol. 12, no. 2, pp. 117–131, June 2015.

[7] A. Y. Ding, J. Crowcroft, and S. Tarkoma, "Poster: SoftOffload: A Programmable Approach Toward Collaborative Mobile Traffic Offloading," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14. New York, NY, USA: ACM, 2014, pp. 368–368.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[9] "The P4 specification version 1.0.2," March 2015. [Online]. Available: http://p4.org/spec

[10] "Open vSwitch manual." [Online]. Available: http://openvswitch.org/support/dist-docs/ovs-ofctl.8.txt

[11] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.

[12] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 99–110.

[13] "ONOS Blackbird performance evaluation." [Online]. Available: https://wiki.onosproject.org/display/ONOS/Master-Performance+and+Scale-out

[14] X. An, D. Perez-Caparros, and Q. Wei, "Consistent Route Update in Software-Defined Networks," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 84–89.

[15] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. –, Aug. 2010.

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.