Ward: Implementing Arbitrary Hierarchical Policies using Packet Resubmit in Programmable Switches

Mojtaba Malekpourshahraki
University of Illinois at Chicago
mmalek3@uic.edu

Brent Stephens
University of Illinois at Chicago
brents@uic.edu

Balajee Vamanan
University of Illinois at Chicago
bvamanan@uic.edu

ABSTRACT

Datacenters in major cloud providers host thousands of competing tenants and applications. Network operators must ensure that available resources are fairly shared and isolated among tenants to meet Service Level Agreements (SLA). Moreover, operators must be able to meet application requirements inside each tenant to provide end-user satisfaction. Providing isolation among tenants, and enforcing application policies require deep, hierarchical policies to isolate tenants and applications separately. Current state of the art approaches cannot enforce deep, hierarchical policies due to the switches' resource limitations. In this paper, we propose *Ward*, a practical approach to enforce deep hierarchical network policies using packet resubmit in programmable switches. Packet resubmit allows switches to reuse network resources in enforcing complex traffic policies. Our empirical results in a sample hierarchical policy with two levels show that *Ward* could enforce tenant isolation and strict priority.

CCS CONCEPTS

• **Networks** → **Packet scheduling**; **Data center networks**; *Packet classification.*

ACM Reference Format:
Mojtaba Malekpourshahraki, Brent Stephens, and Balajee Vamanan. 2019. *Ward*: Implementing Arbitrary Hierarchical Policies using Packet Resubmit in Programmable Switches. In *The 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19 Companion), December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3360468.3366780

1 INTRODUCTION

Today's production datacenters host a large number of competing tenants and applications. Network operators in major cloud providers must ensure that network resources (e.g., bandwidth) are fairly shared among tenants. Similarly, tenants must share the assigned resources among applications to optimize intra-tenant policies such as tail FCT. As such, today's networks incorporate rich policies in both control [5] and data planes [15].

Providing fine-grain traffic isolation in multi-tenant dataceters requires a complex hierarchical policies as the number of comparison increases with the number of flows and tenants. Figure 1 shows an example of a hierarchical policy using restricted directed acyclic graph (DAG) [15]. A DAG determines how to share the bandwidth among tenants and applications. The Network must ensure that two tenants fairly share bottleneck capacity (*WFQ*), and Memchached traffic is prioritized over Hadoop traffic in tenant 2 (*Pri*).

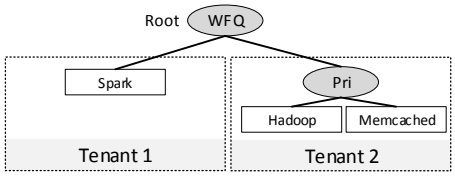


Figure 1: Sample of DAG policy

Unfortunately, enforcing a nested, deep policy graph with different scheduling algorithm is challenging due to the limited resource in tracking the large number of flows and tenants separately. Recent proposals either provide an approximation of the policies [12] or they provide a single or limited number of scheduling policies [6, 8, 10–12] to tackle the limitation in network resources. PIFO [14] and PIEO [13] can implement any complex hierarchical programmable scheduling policy; however, they need a large number of queues or new hardware design for a policy graph with too many vertices. Ether [7] provides a hierarchical policy using multi-sided queues, but the accuracy of the Ether in policy enforcement depends on the policy graph, and the number of queues. In this paper, we present *Ward*, a practical approach to enforce complex network policies in programmable switches. Each packet follows the vertices in policy graph from the root to the corresponding leaf. If the number of stages is not enough to traverse the policy graph, *Ward* leverages *packet resubmit* [3] to tackle the resource limitation. In the worst case, each vertex in the policy graph requires at least one packet submission to the switch pipeline. When a packet leaves the switch pipeline, it carries rate control metadata for the processed policy. If the packet still needs to be considered for a sub-policy, it enters to the switch pipeline for the second time (resubmit), and merges metadata rate control decisions to the result of the current processing policy. For instance, in Figure 1, first packet submission decides on whether the packet is violating the WFQ, and it applies some rate-limiting policy if needed. In the second submission, the switch decides if the packet is violating strict priority among the flows inside tenant 2. Note that there are only 32 stages in programmable switches [4] which that could handle more than one policy in each submission. Thus, packet resubmit is needed to enforce deep policy graphs. As a proof-of-concept, we evaluate *Ward* on BMv2 target [2] and ns3 [1], for a simple scenario shown in Figure 1 and a simple rate limitation policy (ECN mark).

CoNEXT '19 Companion, December 9–12,2019, Orlando, FL, USA

ACM ISBN 978-1-4503-7006-6/19/12.
https://doi.org/10.1145/3360468.3366780

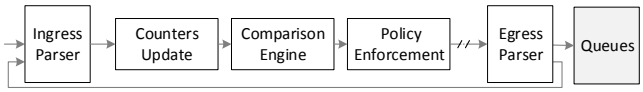


Figure 2: Packet resubmit in *Ward*

We observe that *Ward* could provide isolation among two tenants with the strict priority in Tenant 2.

2 DESIGN

Ward has three main components (see Figure 2): (i) counter update, (ii) comparison engine, and (iii) policy enforcement. When a packet arrives, the switch traverses all vertices of the policy tree, from the root to the leaf that represents the current processing packet. In each step, the comparison engine compares all traffic classes that are included in the current policy vertex, and decides if the packet's traffic violates the processing vertex policy or not. It also determines the rate-limiting mechanism (e.g., ECN mark or drop) based on the severity of the violation, and send it to the policy enforcement via metadata. Policy enforcement decides how to merge the information from the previous step with the policy from the last submit, and it enforces the final decision.

Table 1 shows an example of *Ward* procedure. Assume that *Ward* is processing a packet from Hadoop in Tenant 2 that violate both *WFQ* and *Pri*. Since the packet belongs to Hadoop, *Ward* needs to traverse *WFQ* and *Pri* in the policy graph. In first submit, as packet violates WFQ (comparison), then *Ward* marks the packet (enforcement) and resubmit the packet for *Pri*. If the link is fully utilized, the comparison step marks the packet as drop as the other flow in the policy can saturate the link. Finally, the packet is sent to the last step, and it drops the packet. We explain each step as follow:

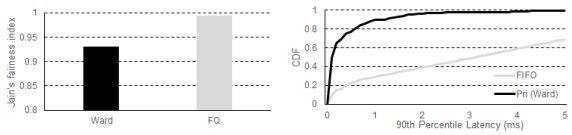
Counter Update. Switches must track the sending rate of tenants and flows to guarantee network isolation. Programmable switches provide counters and sketches to count the number of packets that each flow or tenant sends. *Ward* bypasses this step for resubmitted packets.

Comparison Engine: In this step, *Ward* looks up the counters and decides what kind of rate limitation is required for the packet to meet the policy, and it sends it to the next step as packet metadata. Table 2 shows rate-limiting approaches that are used for comparison engine. Shortest Job First (SJF) scheduler is possible in *Ward*. We plan to study SJF in our future extended version of the paper.

Policy Enforcement: This module decides how to combine the results of the policies in previous vertices in the policy tree (e.g., from the root) with the current processing policy. If rate control decision from previous submits does not match with the current decision, the most strict policy will be applied to the packet. For instance, the dominant policy between drop and ECN mark is drop.

3 EVALUATION AND FUTURE WORK

We evaluate the performance of *Ward* in Mininet/BMv2 [2] and ns3 [1]. We set up a leaf-spine topology with four servers, two leaves, and a spine switch. We ran three different types of flows from two tenants, and we consider a simple DAG policy in Figure 1. For simplicity, we uses ECN mark as rate control mechanism. Logical *AND* operation is used on ECN values in compare engine to aggregate previous values.



(a) Jain's fairness index **(b) 90 percentile FCT**

Figure 3: *Ward* performance evaluation

Table 1: Example of policy enforcement in *Ward*

	Policy	Counters	Comparison	Enforce
Submit	WFQ	Update	ECN	(ECN,ϕ)=ECN
Resubmit	Pri	-	Drop	(ECN,Drop)=Drop

Table 2: Summary of policies and rate control mechanisms

	Rate Control Policy
WFQ	ECN, Drop
Pri	Drop
SJF	ECN, Priority Queue,Drop

Figure 3 shows the bottom-line performance evaluation in terms of both fairness and tail FCT. To evaluate the performance of *Ward* in enforcing WFQ, we used an ideal fair queuing (FQ) as our compared scheme. Figure 3a shows the Jain's fairness index of overall throughput in both tenants in BMv2. This figure shows that *Ward* can provide fairness among different tenants with only 6% lower fairness index compared to the ideal fair queuing. Note that in the experiment, the only traffic control mechanism is ECN mark and other traffic control mechanisms such as packet drop could improve the fairness as senders react faster to it. Unfortunately, Mininet does not support realistic bandwidth; thus, we run the same experiment in ns3 to evaluate second level of hierarchy (*Pri*). Figure 3b shows results of the scenario with higher bandwidth. This figure shows the CDF of the 90 percentile of the short flows in tenant 2 (Memcashed). *Ward* achieves a shorter 99 percentile for the short flow in tenant 2, compared to the case where no strict policy is considered.

Authors in [9] show that the total throughput of the switch slightly reduces when packet resubmit is used; however, the proposed approach in this paper requires more number of packet resubmit. As future work, we plan to study the effect of packet resubmit in the performance of the *Ward*, and the overall throughput of the switch, since ns3 and BMV2 cannot measure the performance of the switch when packet resubmit is used. Moreover, we plan to optimize the design of the *Ward* by using a parallel architecture in each packet submit. Since *Ward* stages are simple, *Ward* can repeat the stages in the same pipeline without resubmit, which remove or reduce the number of required packet resubmit.

REFERENCES

[1] [n.d.]. NS-3 network simulator. http://www.nsnam.org/.

[2] [n.d.]. P4 Language Consortium. 2018. P4-BMv2. https://github.com/p4lang/behavioral-model.

[3] [n.d.]. Portable Switch Architecture (PSA). https://p4.org/specs/.

[4] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.

[5] Y. Chang, A. Rezaei, B. Vamanan, J. Hasan, S. Rao, and T. N. Vijaykumar. 2017. Hydra: Leveraging functional slicing for efficient distributed SDN controllers. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*. 251–258. https://doi.org/10.1109/COMSNETS.2017.7945384

[6] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert Greenberg, and Changhoon Kim. 2013. EyeQ: Practical network performance isolation at the edge. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*. 297–311.

[7] Mojtaba Malekpourshahraki, Brent Stephens, and Balajee Vamanan. 2019. Ether: Providing both Interactive Service and Fairness in Multi-Tenant Datacenters. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*. ACM, 50–56.

[8] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C Mogul, Yoshio Turner, and Jose Renato Santos. 2013. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 351–362.

[9] Ting Qu, Raj Joshi, Mun Choon Chan, Ben Leong, Deke Guo, and Zhong Liu. 2019. SQR: In-network Packet Loss Recovery from Link Failures for Highly Reliable Datacenter Networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE.

[10] Hamed Rezaei, Muhammad Usama Chaudhry, Hamidreza Almasi, and Balajee Vamanan. 2019. ICON: Incast Congestion Control using Packet Pacing in Datacenter Networks. In *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 125–132.

[11] Hamed Rezaei, Mojtaba Malekpourshahraki, and Balajee Vamanan. 2018. Slytherin: Dynamic, network-assisted prioritization of tail packets in datacenter networks. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–9.

[12] Naveen Kr Sharma, Ming Liu, Kishore Atreya, and Arvind Krishnamurthy. 2018. Approximating fair queueing on reconfigurable switches. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 1–16.

[13] Vishal Shrivastav. 2019. Fast, scalable, and programmable packet scheduler in hardware. In *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 367–379.

[14] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable packet scheduling at line rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 44–57.

[15] Brent Stephens, Aditya Akella, and Michael Swift. 2019. Loom: Flexible and Efficient {NIC} Packet Scheduling. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 33–46.