Load Balancing for Deterministic Networks

Shuang Chen†, Jérémie Leguay*, Sébastien Martin*, Paolo Medagliani*
*Paris Research Center, Huawei Technologies Co. Ltd
e-mail addresses: {name.surname}@huawei.com
† Beijing Research Center, Huawei Technologies Co. Ltd
e-mail address: chenshuang23@huawei.com

***Abstract*—With the advent of 5G and the evolution of Internet protocols, industrial applications are moving from vertical solutions to general purpose IP-based infrastructures that need to meet deterministic Quality of Service (QoS) requirements. The IETF *DetNet* working group aims at providing an answer to this need with support for (i) deterministic worst-case latency and jitter, and (ii) zero packet loss for time-sensitive traffic. In particular, this working group is currently specifying Cycle Specified Queuing and Forwarding (CSQF), an extension of Cyclic Queuing and Forwarding (CQF) with multiple transmission queues and support for Segment Routing (SR).**

In this paper, we present a load balancing extension for DetNet that aims at improving network utilization. In order to implement this additional feature, we show how to extend the original SR policy in CSQF. Then, we formulate the optimization problem that a centralized controller has to solve to produce feasible load balancing configurations and we demonstrate on a typical 3-tiers topology that load balancing improves traffic acceptance as the complexity of finding the optimal configurations increases.

***Index Terms*—Deterministic Networking, Routing, Scheduling, Load Balancing.**

I. Introduction

The 5^{th} generation of networks is calling for new latency-critical network services to enable a wide-range of applications like cloud gaming, factory automation, connected vehicles, and smart grids [1]. Traditional IP services cannot provide strict QoS guarantees and even if certain service classes can be given preferential treatment, performance is still statistical. Deterministic performance is now a must to support applications with requirements of low and worst-case latency guarantee.

In the past decade, a collection of IEEE 802.1 Ethernet standards, known as *Time-Sensitive Networking (TSN)* [2], has been developed to support professional applications over Local Area Networks (LAN) with mechanisms such as priority queuing, preemption, traffic shaping, and time-based opening of gates at output ports. While these mechanisms are well suited for static traffic requirements and small networks, they can't scale up to support large-scale IP networks. The IETF DetNet (Deterministic Networking) [1] working group is taking a step further by defining Segment Routing (SR) mechanisms so that Layer 3 can dynamically exploit Layer 2 functionalities for queuing and scheduling to guarantee (i) deterministic worst-case latency and jitter, and (ii) zero packet loss for time-sensitive traffic. In particular, the working group is currently specifying Cycle Specified Queuing and Forwarding (CSQF) [3], a promising extension to Cyclic Queuing and Forwarding (CQF, a.k.a. IEEE 802.1Qch) with more than 2 transmission queues in order to relax tight time-synchronization constraints and to schedule, in a more flexible way, transmissions at each hop. CSQF proposes a scalable solution where transmission cycles at each port repeat periodically thanks to the round-robin opening of multiple queues dynamically selected by IP packets using segment routing identifiers (SIDs), a label stack that determines scheduling and routing at each hop. A network controller decides the proper SR policy (i.e, labeling strategy) for each flow by solving a joint scheduling and routing problem.

In the current CSQF specification [3], flows are routed over a unique *scheduled-path*, a sequence of links with transmission cycles specified at each link. As for traditional IP networks, the use of a single path to route all packets from the same flow can engender a poor utilization of the network bandwidth. This is typically the case when the distribution of flow sizes is skewed and a small set of large flows must be split over multiple paths [4] using well known flow splitting techniques like ECMP [5] or Weighted-Cost Multi-Path (WCMP) [6]. As DetNet systems are intended to support a mix of low-rate control traffic and high-rate sensor-based data streams, we also expect to have the need to spread large flows over multiple paths. Furthermore, as flows may have totally different transmission patterns over cycles, whose periodicity is referred to as *hypercycle*, it may be difficult to combine them in all cases to maximize traffic acceptance. Splitting flows with complicated patterns into a collection of smaller sub-flows with simpler patterns should help making a better use of network resources and accept more traffic.

In this context, we present a load balancing extension for DetNet where flows can be split over multiple paths and multiple cycles. After a necessary reminder of how CSQF works, we propose an extension of the SR policy that needs to be deployed at ingress nodes to enable load balancing. We present three use cases to (i) improve network utilization as well as traffic protection against (ii) busts and (iii) failures. To automatically design load balancing policies, we formulate the DetNet Load Balancing (DN-LB) planning problem to maximize the acceptance of time-triggered flows and we analyze its complexity. Using a typical 3-tiers topology, we demonstrate the benefits of a centralized control plane solving the DN-LB problem to generate feasible load balancing policies over the case where a single scheduled-path routing problem is solved. In particular, we show that traffic acceptance can be improved

Annex to ISBN 978-3-903176-28-7©2020 IFIP

by tens of percents when the complexity of finding the optimal load balancing policy increases, i.e., when the number of flows and the size of the hypercycle increase.

More details about CSQF are given in Sec. II. Relevant related works are discussed in Sec. III. Sec. IV presents the load balancing extension for DetNet. The DN-LB problem is formulated and analyzed in Sec. V. Numerical assessments are shown in Sec. VI. And Sec. VII concludes this paper.

II. DetNet System Model with CSQF

Deterministic performance guarantees are getting momentum thanks to standards like Cycle Specified Queuing and Forwarding (CSQF) [3] for Deterministic Networking (DetNet), allowing the coexistence of deterministic traffic over traditional IP networks. CSQF allows exact packet scheduling—and by consequence deterministic packet delivery—by explicitly stating, via a Segment Routing (SR) label stack, in which cycle (i.e., which queue) each packet should be transmitted after being received and processed. Precise knowledge of the position of a packet inside the network at a generic instant t comes from the fact that, at each node, the worst case forwarding latency is known. Each time a packet arrives at a node, the scheduling of its future transmission is realized by its assignment to one of the inactive queues.

Inside a CSQF-enabled device, each port is equipped with N queues (typically 8), normally used for DiffServ and Best Effort (BE) traffic. In CSQF, the standard defines that out of the N queues, N_{DN} queues (by default 3) are reserved for time-sensitive traffic. These queues are served in a round-robin fashion such that the active queue is *open* for transmission and *closed* for reception. Conversely, the $N_{\mathrm{DN}} - 1$ inactive queues can only accept packets for future transmission. For this reason, the assignment of packets to specific inactive queues defines their transmission schedule and needs to be carefully controlled. Each time-sensitive queue is drained after the activity period and it must be dimensioned to receive all the packets scheduled within a cycle without introducing any packet loss.

Due to the periodic activation of queues, the time at each node is logically divided into cycles, whose duration is assumed to be the same throughout the network. The starting time of the cycles at the different nodes is not synchronized and can present an offset which is measured and known by the controller. CSQF guarantees bounded jitter as all packets experience the same maximum forwarding time, which is deterministic and known in advance by statistically measuring the worst-case delay. In practice, as the real processing delay of each forwarded packet can be smaller than the worst-case, the DetNet-enabled node introduces an artificial additional delay to ensure that the packet forwarding delay is equal to the worst-case.

In Fig. 1, we show an example of how a packet is propagated from node A to node C through node B. Once the packet is sent from A, it is received at B within a cycle (cycle 2 in the figure). As node B decides for immediate packet forwarding, the packet is transmitted in the next cycle. Finally, node C

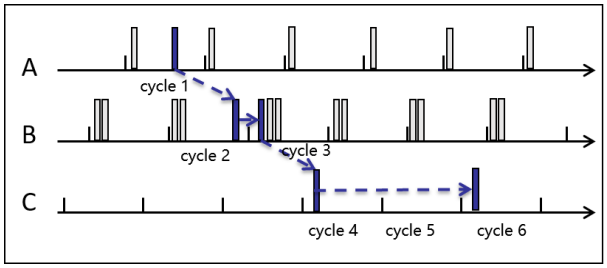


Fig. 1: CSQF packet forwarding. Between nodes A and B, and, B and C, the packet is transmitted in the next cycle, while node C decides to schedule packet transmission two cycles later.

decides for the scheduling of the packet two cycles later, so that the packet will be transmitted at cycle 6. As the same considerations apply if we consider 0 offset between cycles of different nodes, for the sake of simplicity and without loss of generality, we will consider throughout this paper a 0 time offset such that all cycles are aligned at the different nodes.

According to CSQF, a DetNet-enabled device decides how and when a packet is forwarded by consuming the first Segment Routing ID (SID) available in the label stack of packet headers. As a first step, the receiving node maps the SID into the corresponding output port. As a second step, the device uses the same label to select the queue associated with the intended transmission cycle. The SR label stack can be provided by a centralized network controller that (a) computes a feasible path (or set of paths) from source node to destination node, (b) computes the right scheduling within each node traversed by the flow, and (c) distributes the corresponding SR label stack to all the network elements via specific protocols (e.g., PCEP).

In order to ensure deterministic performance, DetNet traffic is time-triggered (TT) and follows a specific pattern that repeats over time. This period is referred to as *hypercycle*. For each cycle, the application specifies how much data will be sent. According to this definition of traffic, the *routing* problem in DetNet consists in finding a path from the source to the destination, while the *scheduling* problem focuses on the mapping of each TT packet inside a specific transmission queue. As a consequence, it is necessary to provide a scheduling and routing decision at each hop and guarantee that enough resources are available. The current limitations imposed on the TT flows are that no load balancing is possible, neither at path level (i.e., a TT flow must use only one path) nor at cycle level (i.e., all the packets of the same flow received in cycle x must be forwarded in the same cycle y). In Section IV, we will show how it is possible to overcome these limitations in order to achieve better network utilization.

III. Related work on Load Balancing

In the literature, a number of load balancing solutions have been proposed for traditional IP networks. Many works advocate skewed traffic distributions and relaxation of the equal cost constraint [7], [8] to increase network utilization.

However, the ECMP (Equal-Cost Multi-Path) [5] extension to OSPF is still the most commonly used technique to load balance traffic. However, research is still focusing on efficient and practical ways to implement unequal cost and uneven flow splitting.

To achieve uneven load balancing problem, extensions of ECMP such as weighted-Cost Multi-Path (WCMP) [6] routing or Niagara [9] have been proposed for uneven flow splitting over equal cost paths. They repeat the same next-hop multiple times in the forwarding table to route different portions of traffic over the paths. As these approaches can lead to a large number of rules, they both propose rules reduction algorithms. However, these two approaches solve a local problem and do not aim at maximizing the overall network throughput. In a past work from us [4], we proposed a global optimization algorithm to configure data plane buckets (i.e;, forwarding rules in group tables) to mitigate this problem. However, in practice, these uneven load balancing solutions are not enough as they are facing the problem that the size of flows is not known in advance and the actual observed rate can deviate the expected ones.

Fortunately, in DetNet flows are either generally described as time-triggered flows or shaped / policed at ingress nodes so that we know their mean rate or worst case burst size. Without loss of generality, we will consider in the rest of this paper time-triggered flows where the number of packets issued by applications in each cycles is known to the network controller. Thanks to this knowledge, an uneven load balancing of DetNet flow can be accurately planned without the risk of problematic deviations. Up to our knowledge, we are the first to propose a load balancing mechanism for DetNet.

IV. Cycle-level Load Balancing with CSQF

In DetNet, similarly to traditionnal IP networks, the use of a single path to route all packets for every flow engenders poor network utilization.

The way load balancing is implemented depends on the way flows are forwarded inside the devices. Traditionally, there exist two different techniques to forward packets in the devices: (i) flow tables and (ii) SR policies. Considering a flow table, each input port is associated with a packet processor that decides, based on forwarding rules, on which output port (and transmission cycle for DetNet) the flow has to be forwarded. In this case forwarding rules have to be installed in all intermediary devices processing a flow. The other approach is based on SR policies and the ingress node attaches the list of SIDs to incoming packets. Intermediate nodes just consume the associated SR label and route the packet using the information contained in the SID. As there is a static mapping between SID and output port, no flow tables are needed.

As CSQF leverages on segment routing to route and schedule packets, in this paper we will focus on the modifications required to the SR policies in order to support cycle-level load balancing. The SR policies are composed by a set of Candidate paths that can be activated by the head-end node that injects the packet into the network. Each Candidate path is composed

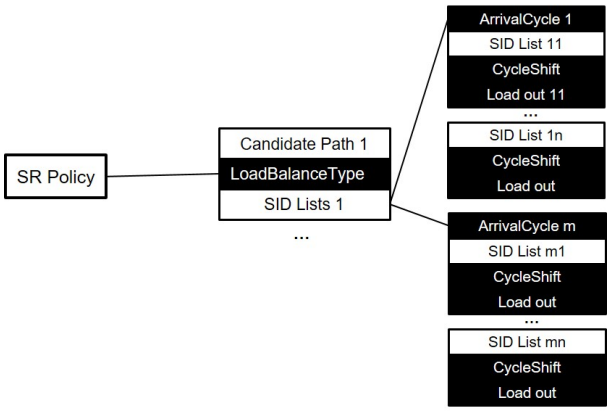


Fig. 2: SR policy structure for multi-path cycle level load balancing (new attributes are highlighted in *black*).

by a set of Segment IDs (SIDs) that are added to the header of each packet and consumed by intermediary nodes in order to forward the packets into the corresponding transmission queue. Note that the selection of the transmission queue induces the selection of the link as each queue is associated to a specific link. In order to implement load balancing, some attributes must be added in the SR policy. First, the policy contains an attribute specific to the input cycle (Arrival cycle), it is used to determine the input cycle to which the policy has to be applied. Second, the policy has an attribute specific to the cycle shift applied to packets (Cycle shift), in case it must be delayed of one or more transmission cycles. Finally, a last attribute (Load out) specifies the target load balancing applied to packets arriving in the input cycle. This latter field can be implemented either using weights (a ratio of the cycle capacity) or as a packet number. A simple way to implement attributes is using TLV fields. In Figure 2, we show the structure of the new SR policy for load balancing in CSQF. For each one of the m arrival cycles, it is possible to define n output paths having each a different cyle shift and load distribution. The new attributes required to handle cycle-level load balancing are shown in black.

A practical example of how to use SR policy to load balance traffic is shown in Figure 3. Beside to each node of the topology on the top figure, we show the SIDs that are assigned to every outgoing queue. For flow 1 (from *a* to *h*) and flow 2 (from *b* to *h*), the corresponding traffic pattern is depicted. The number inside the grey area indicates an active packet for the corresponding flow number, characterizing the active demands at each transmission cycle (3 cycles / queues in the example). For instance at node A there is 1 packet per cycle for flow 1. At node *b* there are 3 packets per cycle for flow 2. Node *b* applies the following routing for flow 2. First, it checks the arrival cycle in order to determine which SID list to apply. For instance, for ArrivalCycle=1, the third list is applied. According to the Load out field, a split 1/3, 2/3 is applied. For 1 packet out of 3, the first SID list (10002, 20002, 30002) is used, while for2 packets out of 3, the second SID

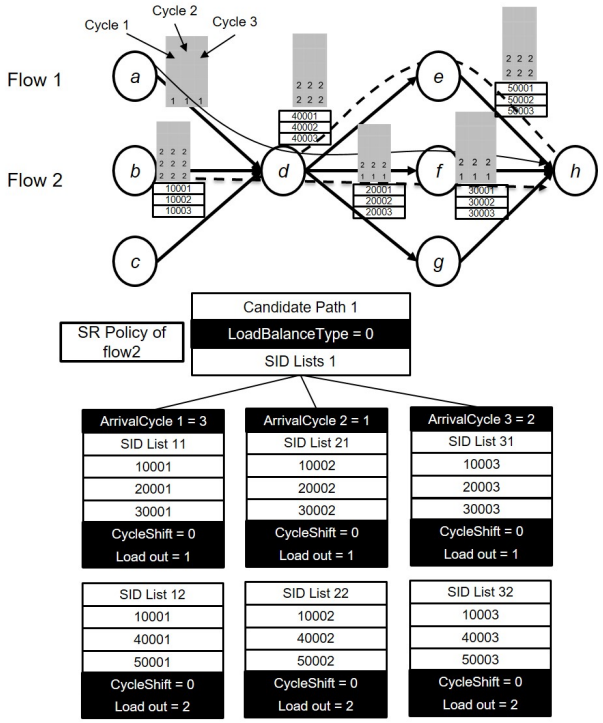


Fig. 3: Example on a 8 node network with two demands from a to h (*flow 1*) and from b to h (*flow 2*), respectively (top figure). SR policy and detailed SR lists for cycle-level load balancing of flow 2 (bottom figure).

list is used (10002, 40002, 50002). As it can be seen from the example, the packets will be split by node *d*, which will forward two packets per cycle of demand 2 on the link *d-e* and 1 packet per cycle on link *d-f*. The same considerations apply for the other arrival cycles.

Support for different use cases. Beyond network utilization improvement, the cycle-level load balancing can also be used to handle bursts of traffic. In case it is not possible to serve the burst using the same path, the SR policy can be used to distribute packets over different links according to the best suitable distribution. If the head-end node, responsible for assigning the SR list to the packets, detects a burst, it can locally decide for the application of the specific load balancing policy. In order to ensure deterministic performance, the controller must take into account the potential presence of bursts while computing the SR list for burst protection for each head-end node. The same considerations also hold for the reaction to failures. The controller can pre-deploy to head-end nodes specific SR policies to react to failures. Once the controller detects a failure in the network, it can instruct the different nodes to activate different SR policies and route traffic elsewhere to avoid the failure. The computation of the policies must be done by the TE controller taking into account performance guarantee for all the existing traffic into the network. In order to support the three different use cases (i.e., load balancing, burst protection, and failure reaction), an additional attribute, referred to as *LoadBalanceType* has been added to the SR policy, as shown in Figure 2. The scope of this attribute is to instruct the head-nodes on the utilization case of the policy.

A drawback of the use of cycle-level load balancing is that it may introduce disordering into the reception of the packets. However, this can be easily compensated in two different ways. First, the paths used for load balancing can be designed to present similar latency characteristics, so that the disordering is minimized. In addition, a buffer can be used to store the packets and order them once received, as already envisioned in DetNet by standards like Frame Replication and Elimination for Reliability (FRER) [10].

V. The DetNet Load Balancing Problem

This section introduces two Integer Linear Programs (ILP) for the routing of traffic in DetNet. First, it formulates the single path routing problem and then it presents the load balancing version where traffic can be split over different paths. In the two formulations we consider a given set of feasible paths for each demand (i.e., flow).

Let's consider the network $G = (V, A)$, a graph where V denotes the set of routers and A the set of links. Each link $a \in A$ has a capacity b_a for each cycle c. A given set of demands D has to be routed over links and cycles. For each demand $d \in D$, bw_c^d bandwidth units are transmitted at the source node in cycle c. We denote $\text{bw}_d = \sum_{c \in C} \text{bw}_c^d$ the total bandwidth for demand d and P_d the set of given feasible paths for d. $\text{bw}_{a,p}^d(c)$ indicates the bandwidth used by the flow d on the link a at cycle c on the path p. A path for a demand is feasible if it respects an end-to-end delay constraint.

a) DetNet routing problem (DN): In the basic DetNet Routing a demand needs to be assigned a unique feasible path and all packets must follow this path to ensure bounded jitter and delay. For each flow $d \in D$ and for each path $p \in P_d$ we consider a binary variable x_p that equals to 1 if the path is used by d and 0 otherwise. The ILP for the DN problem to maximize traffic acceptance is:

$$\max \sum_{d \in D} \sum_{p \in P_d} \text{bw}_d x_p \tag{1}$$

$$\sum_{p \in P^d} x_p \leq 1 \qquad \forall d \in D \tag{2}$$

$$\sum_{d \in D} \sum_{p \in P_d : a \in p} \text{bw}_{a,p}^d(c) x_p \leq b_a \qquad \forall a \in A, \forall c \tag{3}$$

$$x_p \in \{0, 1\} \qquad \forall d \in D, \forall p \in P_d \tag{4}$$

Inequalities (2) ensure that only one path is selected for each demand. Inequalities (3) are the capacity constraints for each link and each cycle.

The DN problem is a NP-hard problem. This problem is harder than a multi-commodity flow problem since if we consider only one cycle the two problems are equivalent.

b) DetNet Load Balancing problem (DN-LB): In the DetNet Load Balancing problem (DN-LB) each cycle of a flow can be routed on a different feasible paths and all frames

of this flow and cycle must use this path. For each demand $d \in D$, each cycle $c \in C$ and each path $p \in P_d$ we consider a binary variable $x_{p,c}$ equal to 1 if the path is used by this demand at cycle c and 0 otherwise. For each demand, we also consider a binary variable y_d equal to 1 if the demand is accepted and 0 otherwise. The ILP for DN-LB to maximize traffic acceptance is:

$$\max \sum_{d \in D} \text{bw}_d y_d \tag{5}$$

$$y_d \leq \sum_{p \in P^d} x_{p,c} \qquad \forall d \in D, \forall c \tag{6}$$

$$\sum_{d \in D} \sum_{p \in P_d : a \in p} \text{bw}^d_{a,p}(c) x_{p,c} \leq b_a \qquad \forall a \in A, \forall c \tag{7}$$

$$y_d \in \{0,1\} \qquad \forall d \in D, \tag{8}$$

$$x_p \in \{0,1\} \qquad \forall d \in D, \forall p \in P_d, \tag{9}$$

Inequalities (6) ensure that if a demand is accepted then one path is selected for each cycle of this demand. Inequalities (7) are the capacity constraints for each link and each cycle.

DN-LB is NP-hard for the same reasons as DN. As mentioned in Sec. IV, an extension of the problem is to consider the split of packets inside every incoming cycle over different paths. In this case, x_p cannot actually be relaxed as all split ratios are not allowed due to packet granularity (capacity constraints need to be satisfied on each path). The problem remains NP-hard in this case.

VI. Numerical Evaluation

This section presents results computed with a C++ environment on a single core 3.0 GHz machine with 10GB RAM. ILPs are solved with IBM CPLEX 12.6.3.

A. Setup

While efficient algorithms can be found for the DN problem [11] and extended to DN-LB, our paper on load balancing sticks to the resolution of the ILP on rather small instances to stay within acceptable computational times. Our goal is to demonstrate, as a proof-of-concept, that cycle-level load balancing can improve network utilization over single path routing when using CSQF.

We considered a typical 3-tiers topology that can be found in IPRAN (IP Radio Access Network) or data center scenarios (e.g., fat trees) with edges nodes, aggregation nodes and core nodes. Edge nodes and aggregation nodes are grouped by domain as depicted in Fig. 4. We consider 2 edge nodes and 2 aggregation nodes per domain. And we have 2 core nodes attached to 2 domains, which makes a total of 16 links.

We consider that each node is running the CSQF standard and up to $N_{\text{DN}} \in [20, 200]$ queues for each output port can be used for DetNet traffic. The capacity of links is defined in number of packets per cycle. The exact link rate is specific to the cycle length and depends on the type of links (e.g., 1GE, 10GE). For our numerical results and without loss of generality, we consider $b_a = 1$ packet / cycle.

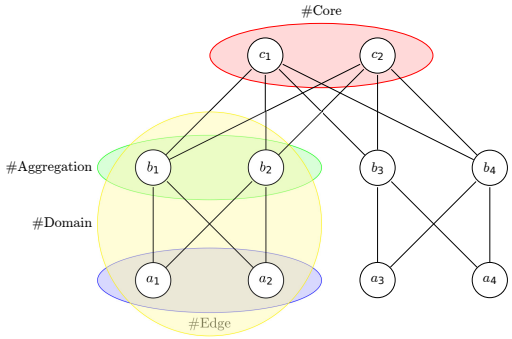


Fig. 4: 3-tiers topology.

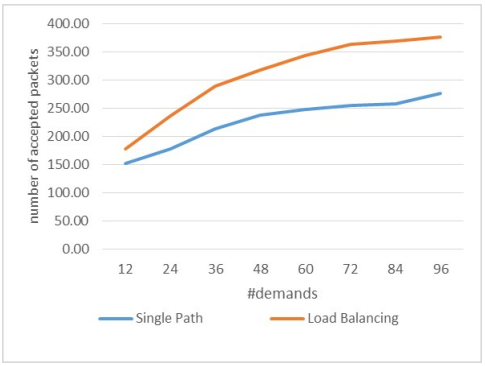
To generate traffic, we consider traffic between all the pairs of edge nodes (12 in total). For each pair, we generate N demands to increase traffic intensity (i.e., we have $12N$ demands in total). As N varies between 1 and 8, the total number of demands varies between 12 and 96. Each demand has an hypercycle of length N_{DN} and issues 1 packet per cycle with a given probability (13% in our case). The idea is to generate random patterns of time-triggered flows with a large-enough probability so that congestion can happen.

In the considered scenario, we pre-compute for each demand all possible shortest paths. In our case, we have in total 16 paths of equal length for every demand, meaning that they have similar latency and packet re-ordering issues at egress nodes are manageable with a small buffer. In practice, the number of paths may be controlled as long as their relative latency. In the next subsection we compare the network utilization when calculating routing with either DN (single path) or DN-LB (multiple paths over multiple cycles).

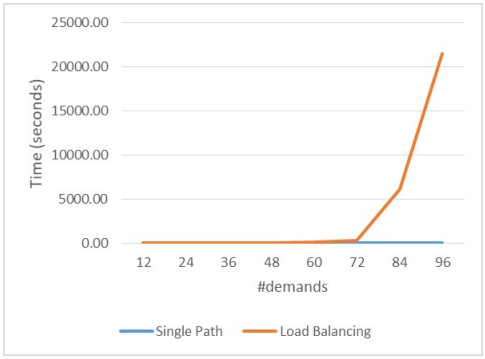
B. Results

Fig. 5a and Fig. 5b show, respectively, the percentage of accepted traffic and the execution time when varying the total number of demands and fixing the hypercycle length (i.e., the number of queues) to 100 cycles. We can see that even for small amount of demands (e.g., 12), load balancing already improves traffic acceptance by 17%. When the traffic increases (up to 96 demands), the gain increases up to 36%. The execution time remains very low below 72 demands and starts to explode after for load balancing. We expect this situation to happen as well for single-path routing when a larger number of demands is considered, as in both cases the ILP is directly solved.

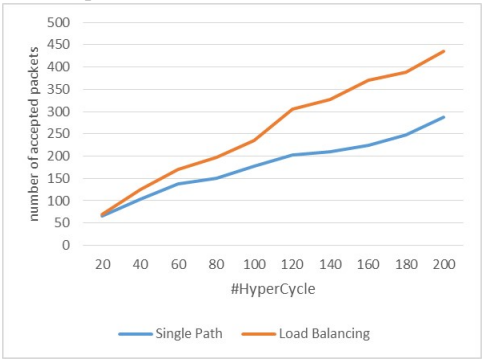
We now analyze the impact of the hypercycle length. Fig. 5c and Fig. 5d show, respectively, the percentage of accepted traffic and the execution time when varying the size of the hypercycle length. We fix traffic intensity to 2 demands for every pair of edge nodes in this case. As before, we observe that the gain in term of traffic acceptance increases with the hypercycle length, up to 51% when 200 cycles are used. For the execution time, as the problem becomes highly combinatorial, the execution time increases with the hypercycle length for load balancing. The explosion is not as high, but the trend is the



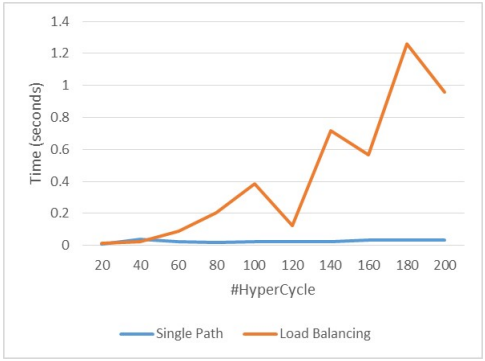
(a) Accepted traffic vs. total number of demands.



(b) Execution time vs. total number of demands.



(c) Accepted traffic vs. hypercycle length.



(d) Execution time vs. hypercycle length.

Fig. 5: Benchmarking results on a 3-tiers topology scenario varying the number of queues (i.e., the hypercycle length) and the number of demands.

same. The execution time will increase if we also increase the number of demands. Actually, increasing the hypercycle length or the number of demands means creating more opportunities for load balancing to improve network utilization, but at the same time it increases the problem complexity. We remind that our goal in this paper is to showcase the load balancing functionality, we leave the development of a scalable algorithm for future work (as an extension of [11]).

VII. Conclusion

In this paper we presented a load balancing extension for DetNet where flows can be split over multiple paths and multiple cycles. In particular, we proposed an extension of the original SR policy for CSQF supporting different use cases to improve network utilization and protect traffic against bursts and failures. We formulated the DN-LB problem as an extension of a multi-commodity flow problem to automatically calculate load balancing configurations. On a typical 3-tiers network, we demonstrated, as a proof-of-concept, gains in terms of traffic acceptance up to 50% when the complexity of finding the optimal solution increases, i.e, when the size of the hypercycle length and the number of flows increase.

For future work, we leave the design of a scalable algorithm to solve the DN-LB problem. We also leave the resolution of several related problems such as the split of individual cycles over multiple paths or the calculation of multiple paths with minimum latency deviation so that packet re-ordering issues can be limited.

References

[1] E. Grossman, "Deterministic Networking Use Cases," RFC 8578, May 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8578.txt

[2] A. Nasrallah, V. Balasubramanian, A. S. Thyagaturu, M. Reisslein, and H. Elbakoury, "Cyclic queuing and forwarding for large scale deterministic networks: A survey," *CoRR*, vol. abs/1905.08478, 2019.

[3] M. Chen, X. Geng, and Z. Li, "Segment Routing (SR) Based Bounded Latency," Internet Engineering Task Force, Internet-Draft draft-chen-detnet-sr-based-bounded-latency-00, Oct. 2018.

[4] P. Medagliani, J. Leguay, M. Abdullah, M. Leconte, and S. Paris, "Global optimization for hash-based splitting," *IEEE GLOBECOM, year=2016*.

[5] D. Thaler, "Multipath issues in unicast and multicast next-hop selection. internet engineering task force: RFC 2991," 2000.

[6] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers," in *Proc. of ACM EuroSys*.

[7] S. Prabhavat, H. Nishiyama, N. Ansari, and N. Kato, "On load distribution over multipath networks," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 3, pp. 662–680, 2012.

[8] G. M. Lee and J. Choi, "A survey of multipath routing for traffic engineering," *Information and Communications University, Korea*, 2002.

[9] N. Kang, J. Reumann, A. Shraer, and J. Rexford, "Efficient Traffic Splitting on SDN switches," Tech. Rep., 2015.

[10] "Ieee draft standard for local and metropolitan area networks – frame replication and elimination for reliability," *IEEE P802.1CB/D2.8, March 2017*, pp. 1–97, Jan 2017.

[11] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng, "Joint Routing and Scheduling for Large-Scale Deterministic IP Networks," in *Arxiv Preprint arXiv:2004.02717.*, 2020.