CacheNet: Leveraging the Principle of Locality in Reconfigurable Network Design

Chen Griner and Chen Avin
School of Electrical and Computer Engineering
Ben Gurion University of the Negev, Israel
griner@post.bgu.ac.il, avin@cse.bgu.ac.il

***Abstract*—Emerging optical communication technologies support the dynamic reconfiguration of datacenter network topologies depending on the traffic they serve. However, to reap the benefits of such demand-aware networks, a control logic is required which allows to quickly learn and adapt to traffic patterns. This paper presents CacheNet, a novel approach to efficiently control demand-aware networks. CacheNet leverages temporal and spatial locality in the traffic by managing the reconfigurable links of the optical switches as a links-cache. Network traffic, in turn, can be served either by a link from the link-cache component or by a demand-oblivious topology component. We study several classic caching algorithms and provide an analytical model which captures their performance benefits compared to an all demand-oblivious topology. Our analytical results show that based on the hit ratios and the links-cache size, our hybrid design can outperform designs that are based only on demand-oblivious topology.**

I. Introduction

Traditional datacenter networks have in common that they rely on a topology which is demand-oblivious, i.e., independent of the current traffic pattern it serves. Recently, reconfigurable optical technologies have introduced an intriguing alternative to design datacenter networks, allowing to dynamically establish shortcuts, depending on the demand [1]–[3]. However, designing demand-aware networks is challenging. Existing architectures , including *Helios* [1], *RE-ACToR* [2] and *Mordia* [3], among others, are based around creating a schedule of reconfigurations for a snapshot of the traffic matrix. More specifically, most existing reconfigurable optical technologies allow to provide dynamic matchings between a set of endpoints (e.g., top-of-rack switches), and throughput can be optimized by cleverly scheduling a sequence of such matchings. This paper presents a novel approach to design demand-aware and self-adjusting networks, which is inspired by the success of leveraging the locality principle [4] in other computing systems using caching (e.g., CPU, memory, web caches). That is, rather than aiming to collect information patterns explicitly, we propose an implicit approach in which the different optical switches manage their reconfigurable links as a *distributed cache*. The links in the cache serve communication requests with very low latency and high capacity, and are adjusted in an online manner, according to the changing demand in the network. We propose *CacheNet*, a hybrid architecture which consists of both demand-aware links, that can be realized as a distributed link cache, and demand-oblivious links. Ideally, the demand-aware component adjusts to changing network demand patterns to serve large flows at lower overhead, while the demand-oblivious component handles any remaining traffic which the cache cannot handle (e.g., all-to-all shuffle traffic). In particular, the demand-oblivious component of *CacheNet* relies on RotorNet [5], which has serve shuffle traffic particularly well; We provide a formal analysis of *CacheNet* which allows us to shed light on the optimal distribution of demand-oblivious and demand-aware links in the datacenter. We further complement these insights with an empirical evaluation, considering both important synthetic and real-world workloads. Our results reveal that *CacheNet* can greatly benefit from its hybrid design. To the best of our knowledge, we are the first to establish a connection to explore the opportunities of a distributed link cache to enhance an otherwise demand-oblivious topology. The complete technical details will appear in the full version of the paper.

Annex to ISBN 978-3-903176-39-3

II. Hybrid Architecture Model

We consider a hybrid architecture model which will allow us to compare trade-offs between demand-oblivious and demand-aware networks. In particular, in this paper, we will use an abstract view RotorNet [5] as the demand-oblivious network, henceforth denoted as *rotor-net*. For the demand-aware network, we will use *cache-net*, also described in this section, which is based on our distributed link cache approach. We will refer to our hybrid architecture combining *rotor-net* and *cache-net*, as *CacheNet*. To this end, we will assume that for the design of *CacheNet*, we are given a *link budget* (or synonymously edge budget) of m edges (optical links) to serve the communication between n nodes (i.e., possible source or destinations). In a data-center network, sources and destinations could be different ToR switches (as in our empirical traces), but more generally they may represent any type of network nodes (e.g., hosts). Each of these edges can be assigned to either the *rotor-net* component or to the *cache-net* component. In the following, we will denote the number of edges assigned to either *rotor-net* or to the cache as m_r and m_c respectably, and $m_r + m_c = m$. We will first introduce the two networks in turn and then describe *CacheNet*.

A. The Demand-Oblivious Network: rotor-net

We consider an abstract model of *RotorNet* [5] which we denote by *rotor-net*. In a nutshell, *rotor-net* is simply a reconfigurable network which cycles periodically through a sequence of matchings, in a demand-oblivious manner. *rotor-net* has m_r links and is operated by cycling in a round robin manner through all $n(n-1)$ possible links of the all-to-all *directed* complete graph. In every time slot *rotor-net* connects a set of m_r links and disconnects the previous set of m_r links, until all possible links have been covered in a single full *cycle*.

B. The Demand-Aware Network: Cache-Net

cache-net has a total budget of m_c links. Each link in the cache is either connected and ready to be used for packet transmission, or it is being reconfigured, and therefore currently unavailable. When a packet from u to v is sent and the corresponding physical link (u, v) is in the cache, we have a *cache hit*. The decision of when to reconnect a link (insert it to the cache and remove another link from the cache) is left to a *cache replacement policy* (a.k.a. caching algorithm).

C. The CacheNet System

We assume that *all* packets are sent either on *rotor-net* or on *cache-net*. When a packet arrives to the *CacheNet* system, if an appropriate *cache* edge is available, the packet is sent immediately on that edge to its destination, in a single hop. Otherwise, the packet is sent using *rotor-net*, and the system's cache is updated as necessary. During the reconfiguration time both the new and the old links are not usable, and all messages for those links are transmitted using the *rotor-net* system.

Given some edge budget m and a traffic pattern σ, which is a sequence of requests, our goal is to find a partition of the edges into m_r and m_c such that the performance of the network is optimized.

We study both the hit ratio of basic caching algorithms and the optimal partition of the total budget m (to m_r and m_c) that will maximize the performance of *CacheNet* analytically. One extreme of our system is a completely demand-oblivious network, that is $m_c = 0$ and $m_r = m$: a pure *rotor-net*. Another extreme is a pure caching system where $m_c = m$: a pure *cache-net*.

III. Overview of Analysis of CacheNet

We evaluate the performance of *CacheNet* on a ToR-to-ToR network, by analyzing at the *average delay* for a packet from the moment it first reached the source ToR, until it arrives at its destination ToR. To derive a concise formula for this delay we assume all packets are of the same size and type, and differ only by their timestamps, source and destination nodes. All packets are transmitted using direct single hop connections. Consider the three main elements of the average delay: that is, the delay for packets transmitted on the *cache-net* subsystem and *rotor-net* subsystems, and the cache hit ratio of the hybrid system, denoted as t_r, t_c and h respectively. Let us first consider t_c. When a packet is transmitted on the *cache-net* subsystem, it's sent using a single hop so the average delay is $t_c = t$ i.e, where t is the transmission time. For the *rotor-net* subsystem the average delay t_r, must be greater than t, as it is also a function the number of links m_r. More links would reduce the length of time which is required for *rotor-net* to cover a complete graph. A full analyses of this is reserved for the full version of this paper. Lastly, the hit ratio $0 \leq h \leq 1$, namely, the fraction of times that a packet arrived when the correct link is already in the cache.

This is the ratio between the number of packets sent using the cache, and the number of total packets sent, i.e., the length of the trace $|\sigma|$. The average delay per packet in the entire *CacheNet* system $AD(h)$ can therefore be seen as the weighted average of the delay of each subsystem according to the hit ratio h: $AD(h) = ht_c + (1-h)t_r$. In order to have a useful metric for the functionality of *CacheNet* we compare our system to a baseline system, which is completely demand oblivious, *rotor-net*, with the same link budget m as the total budget of our system. For this pure *rotor-net* we denote it's expected delay as t_r^* . To compare both systems and see what improvement can be gained by using our system, we define the *effectiveness ratio* as the ratio of the average packet delay of both systems, assuming that t is negligible:

Definition 1 (Effectiveness ratio):

$$\zeta_m(m_r, h) = \frac{\text{Average delay } \textit{CacheNet}}{\text{Average delay pure } \textit{rotor-net}} = \frac{AD(h)}{t_r^*} \quad (1)$$

Assuming that the transmission time is negligible this ratio can be summed up to the following result.

Claim 1: The Effectiveness ratio of *CacheNet* is:

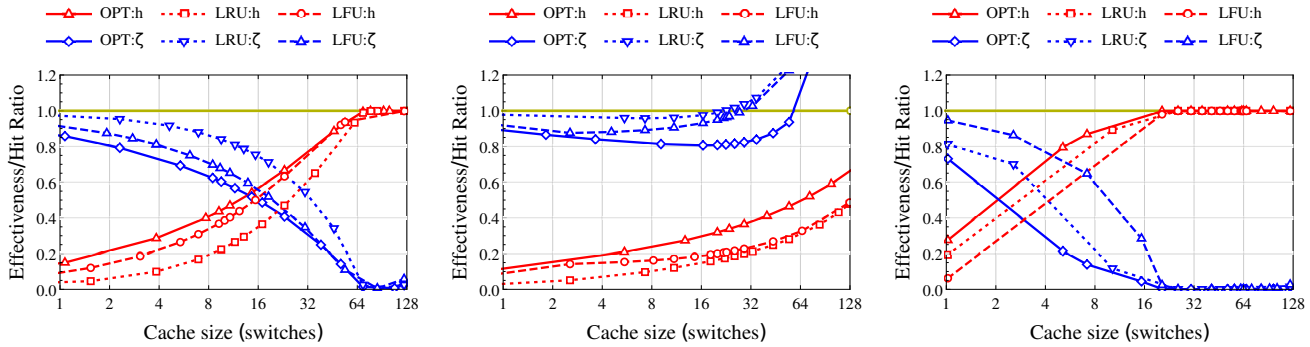
$$\zeta_m(m_r, h) = (1-h)\frac{m}{m_r} \quad (2)$$

The values of the effectiveness ratio can range from 0 to ∞, and desired values lie in the range 0 to 1. A value less than 1 means that *CacheNet* improves on the baseline system in our case, pure *rotor-net*.

IV. Empirical Results

First, let us shortly discuss the dataset used in our evaluation. We tested *CacheNet* on the a set of six traces three rack level Facebook traces [6], two HPC traces, MultiGrid and Mocfe [7], and a pFabric trace [8]. The full details of the datasets are found in the full version of this paper, here we present the results of three traces. We consider the performance of three basic cache replacement policies: Least recently used (LRU), Least-frequently used (LFU), and OPT (optimal). The OPT policy (a.k.a. Belady's algorithm) can be obtained offline and it always discards the item (a link) that will not be needed for the longest in the future. Reconfiguration times of the cache were not taken into account when evaluating the hit ratios.

This means that after an event of a cache miss, when a new link is reconfigured , it will be ready by the next time a packet with the same source and destination arrives. This replacement assumption increases the hit ratios in the results, and therefore, the results should be viewed as an *upper* bound on the performance of *CacheNet*. Figure 1 presents the *effectiveness plot* for three traces along with the hit ratio results



(a) DB trace (b) Hadoop trace (c) MultiGrid trace

Fig. 1: The effectiveness plot for three traces with link budget set to 1024, $m = 128n$

obtained by using three caching algorithms. The plot's goal is to show the potential benefits of using *CacheNet* by converting *rotor* links to *cache* links. Each such plot considers a trace and a link budget m and contains three curves. The first is the effectiveness ratio which is denoted by the blue line. The second red curve, is the hit ratio. The third yellow curve, is a "reference line" set to $y = 1$. The Y axis measures both the hit and the effectiveness ratios. The X axis (logarithmic scale) shows the size of the *cache*, i.e., the $m_c = kn$ links in the cache (recall that a switch has n links). For example when $k = 64$, the cache size is $m_c = 64 \cdot n$ links. The *link* budget m, for each trace is always equal to 128 switches as was used in the original *RotorNet* paper [5]. The yellow line acts as a useful boundary in each plot. If the value of the effectiveness ratio is greater than one, e.g. 1.1 it means that *CacheNet* with the current cache size, $m_c = kn$, has average packet delay that is 10% worse than a *rotor-net*, if the effectiveness ratio is below the line, e.g. 0.9 is means that *CacheNet* outperforms a pure *rotor-net* by 10% in average packet delay. To produce an effectiveness ratio curve, the values of the empirical hit ratio are fed into the formula presented in Eq. (1). Typically, when k is small, close to 1, the effectiveness ratio will be close to 1 as a *CacheNet* with a very small cache is similar to a pure *rotor-net*. When k is close to 128 the effectiveness will often tend to grow towards ∞, as any miss will result in a substantial delay for messages served by *rotor-net*, unless we have a 100% hit ratio. when the effectiveness ratio has one global minimum in the range $k \in [1 \dots 128]$, it corresponds to an optimal division of m_r and m_c. Figure 1 (a) of the DB trace shows an almost best case example for *CacheNet*. All tested values for the size of the cache m_c were able to improve on the performance of *rotor-net*. With all three algorithms reaching nearly 100% improvement with m_c of about 70 switches. These results can be explained as a consequence of a relatively high amount of structure in the trace.

Figure 1 (b) shows the results for the Hadoop trace. They present a case were *CacheNet* was able to improve on *rotor-net*, but not nearly as significantly as for the DB trace. In particular we see that LFU and LRU were able to reach about 10% improvement with a small cache of about 5 to 10 switches, while the improvement brought by OPT is more significant at around 20%. These results with Hadoop are surprising, since the Hadoop trace lacks *significant* structure [6], which should lead to negligible improvement. However, looking at the hit ratio curves where $x > 16$ the hit ratio seems to grow at a slightly faster rate, which may indicate some structure that LFU and OPT are able to use. Figure 1 (c) shows an HPC trace of the MultiGrid application [7]. The effectiveness plot shows that the hit ratio of LRU is superior to LFU. However, all three algorithms reach a hit rate of about 100% with 20 switches. One possible explanation for the under-performance of LFU is that while the HPC trace distributions are skewed, they are only skewed in the sense that they are sparse; that is, only a small part of the possible communicating pairs appear in the trace. The pairs that do appear in the trace are (relatively) uniformly distributed.

Acknowledgement: We would like to thank Stefan Schimd for many discussions and his insightful feedback. This project received funding by the European Research Council (ERC), grant agreement no. 864228, Horizon 2020, 2020-2025.

REFERENCES

[1] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 339–350, 2011.

[2] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter, "Circuit switching under the radar with reactor," in *USENIX NSDI*, (USA), pp. 1–15, 2014.

[3] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *ACM SIGCOMM CCR*, vol. 43, pp. 447–458, 2013.

[4] P. J. Denning and P. J., "The locality principle," *Communications of the ACM*, vol. 48, p. 19, jul 2005.

[5] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proc. of ACM SIGCOMM*, pp. 267–280, 2017.

[6] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM CCR*, vol. 45, pp. 123–137, 2015.

[7] U. DOE, "Characterization of the DOE mini-apps." https://portal.nersc.gov/project/CAL/doe-miniapps.htm, 2016.

[8] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM CCR*, vol. 43, pp. 435–446, 2013.