MARS: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks

VAMSI ADDANKI, Faculty of Electrical Engineering and Computer Science, TU Berlin, Germany
CHEN AVIN, School of Electrical and Computer Engineering, Ben-Gurion University of the Negev, Israel
STEFAN SCHMID, Faculty of Electrical Engineering and Computer Science, TU Berlin, Germany

The performance of large-scale computing systems often critically depends on high-performance communication networks. Dynamically reconfigurable topologies, e.g., based on optical circuit switches, are emerging as an innovative new technology to deal with the explosive growth of datacenter traffic. Specifically, *periodic* reconfigurable datacenter networks (RDCNs) such as RotorNet (SIGCOMM 2017), Opera (NSDI 2020) and Sirius (SIGCOMM 2020) have been shown to provide high throughput, by emulating a *complete graph* through fast periodic circuit switch scheduling.

However, to achieve such a high throughput, existing reconfigurable network designs pay a high price: in terms of potentially high delays, but also, as we show as a first contribution in this paper, in terms of the high buffer requirements. In particular, we show that under buffer constraints, emulating the high-throughput complete graph is infeasible at scale, and we uncover a spectrum of unvisited and attractive alternative RDCNs, which emulate regular graphs, but with lower node degree than the complete graph.

We present MARS, a periodic reconfigurable topology which emulates a d-regular graph with near-optimal throughput. In particular, we systematically analyze how the degree d can be optimized for throughput given the available buffer and delay tolerance of the datacenter. We further show empirically that MARS achieves higher throughput compared to existing systems when buffer sizes are bounded.

CCS Concepts: • **Networks** → **Network architectures**; **Network performance modeling**.

Additional Key Words and Phrases: Datacenter, Throughput, Buffer requirements, Reconfigurable networks.

ACM Reference Format:
Vamsi Addanki, Chen Avin, and Stefan Schmid. 2023. MARS: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 1, Article 2 (March 2023), 43 pages. https://doi.org/10.1145/3579312

1 INTRODUCTION

With the popularity of data-centric and distributed applications, the traffic in datacenters is growing explosively. Dealing with this traffic however becomes increasingly challenging: while cloud traffic roughly doubles each year [1], the capacity increase provided by electrical switches for a given power and cost starts to lag behind. This gap is expected to worsen with the current trend to hardware-driven workloads such as distributed machine learning training [2].

As the *throughput* of datacenter networks is becoming more and more critical for application performance, over the last years, great efforts have been made to increase the capacity of datacenter topologies. A particularly innovative architecture to meet the stringent bandwidth requirements of modern datacenters, are *reconfigurable* (optical) datacenter networks (RDCNs) [2–11], e.g., based on

Authors' addresses: Vamsi Addanki, Faculty of Electrical Engineering and Computer Science, TU Berlin, Germany; Chen Avin, School of Electrical and Computer Engineering, Ben-Gurion University of the Negev, Israel; Stefan Schmid, Faculty of Electrical Engineering and Computer Science, TU Berlin, Germany.

2476-1249/2023/3-ART2 $15.00 https://doi.org/10.1145/3579312

optical circuit switches, tunable lasers, and simple passive gratings [12]. By quickly cycling through a sequence of different topologies—typically matchings between top-of-rack (ToR) switches—RDCNs such as RotorNet [3], Opera [4] or Sirius [2] can provide periodic direct connectivity between rack pairs, at microsecond or even nanosecond granularity. A common property of these systems is that they emulate a *complete graph*, and so avoid the "bandwidth tax" of multi-hop forwarding [4, 13]. Indeed, empirical studies show that periodic reconfigurable datacenter topologies can achieve significantly higher throughput compared to cost-equivalent traditional datacenters based on *static* topologies [2–4].

This paper is motivated by the observation that the existing approach of using reconfigurable technologies to emulate complete graphs comes at a price: long delays and large buffer requirements.

First, at scale, emulating a complete graph can entail *long delays*: the denser the emulated network, the longer the periodic reconfiguration cycle and hence the longer a given rack pair has to wait to be connected again. Second, as we show analytically in this paper, the resulting long reconfiguration cycles require excessive buffering at the ToR switches and end-hosts. The required buffer can intuitively be viewed as *bandwidth-delay* product of dynamic topologies similar to the corresponding notion for static topologies in TCP literature. In practice though, datacenter switches are equipped with shallow buffers. Further, several studies in the recent past show an increasing gap between switch capacity and buffer sizes [14, 15].

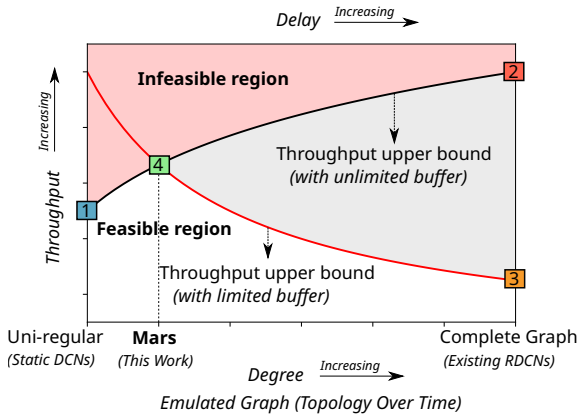
Our main insight in this paper is that accounting for buffer constraints can significantly change the design considerations of reconfigurable datacenter networks.

In particular, we initiate the study of—and make the case for—reconfigurable networks which emulate graphs *of lower node degree*, uncovering an entire spectrum of possible topology designs. We present a systematic and formal analysis of the design spectrum and the tradeoffs that these topologies introduce in terms of throughput, delay, and buffer requirements. Our perspective and the main tradeoffs are intuitively visualized in Figure 1 (left), where on the x-axis (at the bottom) we show the topologies according to the node degree of the graph they emulate. The x-axis (at the top) also represents increasing delay from left to right. The y-axis shows the throughput of the system from low to high. Let us elaborate on the important points in the figure (summarizes in the table on the right):

1 **Static DCNs:** On the very left of the design space are traditional static datacenter networks referred as uni-regular topologies [16] e.g., expander based DCNs [17–19][1]. Such a design in principle incurs the lowest delay and requires the least amount of buffer to achieve its ideal throughput, i.e., the optimal throughput under low delay tolerance. However, since the topology remains static, for scalability reasons, each ToR switch can only connect to a limited number of other ToR switches. This results in long multi-hop paths and hence a high "bandwidth tax" [13], and lower throughput.

2 **Existing periodic RDCNs:** To reduce the "bandwidth tax" and to achieve high throughput, existing designs resort to emulating a complete graph where each rack-pair is connected directly once in every matching cycle [2, 3]. Such designs achieve high throughput, in fact maximum across all topologies in our design space, i.e., the optimal throughput. However, as we will show in this paper, the high throughput offered by emulating a complete graph comes at the cost of high delay and is subject to the availability of large buffers.

[1]These works also claimed that static expander-based topologies are similar or better designs (in terms of performance and cost) compared to Clos-based topologies. We henceforth focus on static expanders. A detailed discussion can be found in §6.



Topo.	Throughput	Delay	Buffer
1	Low (Optimal)	Low	Low
2	High (Optimal)	High	High
3	Low (Not optimal)	High	Low
4	**High (Optimal)**	**Moderate**	**Any**

Fig. 1. Periodic reconfigurable datacenter topologies pose fundamental tradeoffs across throughput, delay and buffer requirements. Existing designs are at the extremes of a spectrum of optimal designs with lower delay and buffer requirements.

3 **Existing periodic RDCNs under resource constraints:** Given the large buffer requirements of existing designs (emulating a complete graph), we study their throughput under limited buffer. Interestingly, we find that, existing periodic reconfigurable datacenter topologies may perform equally or worse compared to a static uni-regular topology in terms of throughput, when buffer sizes are bounded.

4 **MARS:** Exploiting the fundamental tradeoffs across the design space, we propose MARS, a periodic reconfigurable topology that provides near optimal, high throughput with the limited amount of available buffer. Specifically, we parametrize our design based on the delay tolerance and available buffer. We systematically determine the optimal degree d which depending on the resource constraints lies between a static topology and a complete graph.

It is interesting to observe from Figure 1 that the throughput-delay relation implies an *infeasible* region for topology design (shown in red shade). The available buffer space further restricts the design space (shown in gray shade), imposing a fundamental tradeoff across the topologies within the feasible region in terms of throughput, delay and buffer.

Our analytical approach in this paper is novel and relies on a reduction of a periodic evolving graph to a specific static graph (Theorem 1). This enables us to study the throughput maximization problem using well-known graph analysis techniques for static graphs and to analytically evaluate the throughput of both existing dynamic topologies (RotorNet, Opera, and Sirius) as well as possible alternatives (Mars). We believe that this reduction technique may be of independent interest and could be potentially used to study other properties of dynamic topologies.

In summary, our key contributions in this paper are:

- We provide a throughput-centric view and formal model of the performance of periodic reconfigurable datacenter topologies. In particular, we analytically derive the relation between throughput, delay, and the required amount of buffer in the network which reveals a non-trivial tradeoff in the design of periodic reconfigurable topologies.
- We present MARS, a novel reconfigurable datacenter design that maximizes throughput under limited buffer and delay requirements.

- We report on an extensive evaluation showing that MARS improves throughput by up to 4x compared to existing approaches when buffer sizes are bounded. Our evaluation also shows that MARS improves the 99-percentile flow completion time for short flows by up to 75% compared to the state-of-the-art approach [4] and by up to 87% compared to Sirius and RotorNet.

For ease of presentation and due to space constraints, we explain the main concepts of the paper intuitively, and defer the formal definitions and detailed proofs to the appendix.

2 PRELIMINARIES & MODEL

We first give a brief background on periodic reconfigurable topologies and introduce our model.

2.1 Periodic Reconfigurable Topologies

The key enablers and building blocks for fast periodic reconfigurations are optical circuit switching technologies such as rotor-switches [3], AWGR gratings [20] and tunable lasers [2, 21]. Such technologies are oblivious to the demand, thus avoiding the overhead of measuring and estimating the demand. For instance, the overhead is in the order of microseconds for RotorNet and nanoseconds for Sirius which periodically reconfigure the network topology.

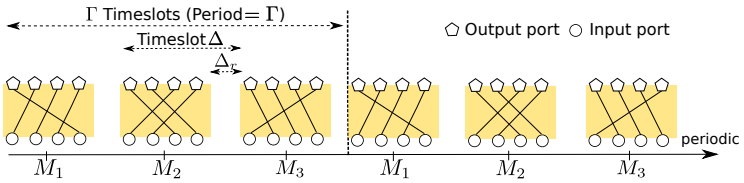
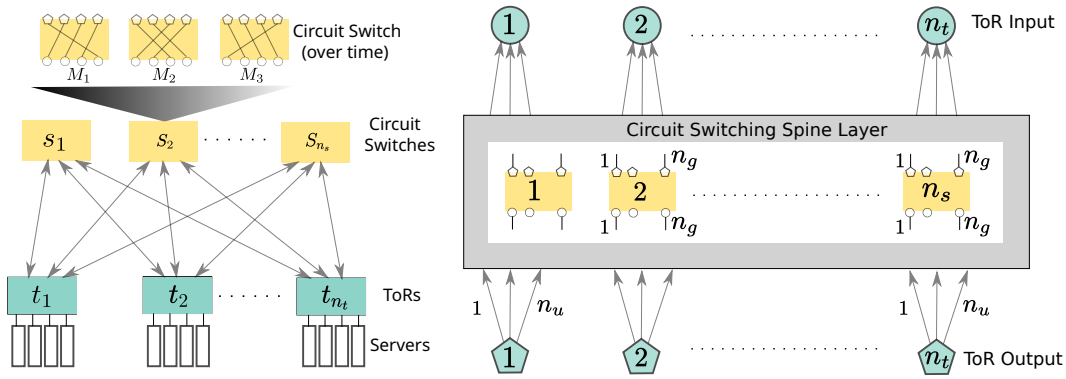


Fig. 2. The periodic sequence of matchings of an optical circuit switch.

Circuit switch model: As a unified model of periodic circuit switching technologies, a *rotor-switch* serves as a building block for us in the rest of the paper. Rotor-switches perform *circuit switching* and implement a sequence of matchings in a periodic schedule with period Γ where a matching is a map from the switch's input ports to its output ports. The packets arriving at an input port of a rotor-switch are forwarded to the matched output port. Specifically, rotor-switches do not process packets and the forwarding is dictated by the matching at any given instance of time. Figure 2 shows an example of a rotor-switch with four input and output ports with a sequence of three matchings. We use the term *timeslot* denoted by Δ to refer to the total time spent by rotor-switch in each matching including the reconfiguration time. We denote the reconfiguration time by Δ_r. The utilization time (when traffic is sent) is then $\Delta - \Delta_r$. In essence, a rotor-switch pays a "latency tax" of $\Delta_u = \frac{\Delta_r}{\Delta}$.

Note that, newer proposals such as Sirius [2] use optical gratings and tunable lasers to achieve fast periodic reconfigurations at nanosecond scale. In contrast to rotor-switches, Sirius uses optical gratings which are the building blocks and the ToR switches (or end-hosts) connect to the gratings via tunable lasers which tune the wavelength of emitted light periodically. However, both RotorNet and Sirius logically capture a rotor-switch that rotates periodically across a fixed number of matchings. To this end, Sirius can be abstracted as a topology with rotor-switches and the difference arises in the system level parameters such as reconfiguration time Δ_r. We do not model system level parameters in this paper and only focus on the topological aspects. In the following, we will sometimes use the more general term *optical switch* when referring to the *rotor-switch*.

Topology: Following the literature on existing proposals [2–4], we consider 2-tier reconfigurable datacenter topologies with a set of ToR switches (leaf layer) and a set of circuit switches (spine



(a) A 2-tier periodic RCDN.

(b) An abstract view of periodic RCDN topologies.

Fig. 3. Illustration of periodic RCDN model we consider in this paper, consisting of n_u ToR switches each with n_u uplinks. The datacenter is interconnected by a circuit switching spine layer consisting of n_s circuit switches, each with n_p input and output ports.

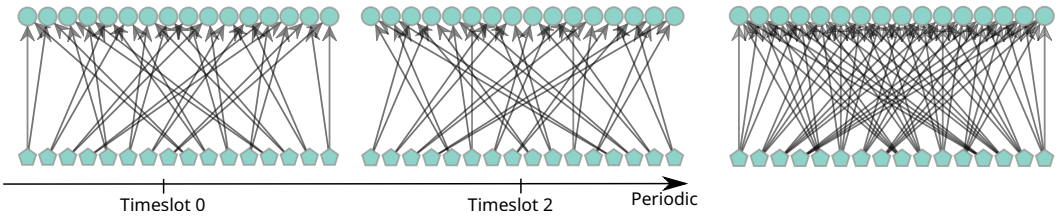
layer) as shown in Figure 3a, where each circuit switch functions as a rotor-switch. The uplinks of the ToR switches connect to the spine layer which interconnects the datacenter[2]. Concretely, as illustrated in Figure 3b, we consider a periodic reconfigurable datacenter with n_t ToR switches each with n_u uplinks; n_s circuit switches each with n_p input and output ports. We assume that a link e in the topology has capacity $c(e)$. We will later show that this generalized view reveals an entire spectrum of topologies where the existing systems (e.g., RotorNet and Sirius) are special instances of the spectrum which emulate a complete graph i.e., each ToR connects to every other ToR in a period.

2.2 Graph Theoretic Model of Periodic ToR-to-ToR Connectivity

In contrast to packet switched networks, circuit switched networks do not buffer packets. To this end, since periodic reconfigurable topologies are circuit switched, we are mainly interested in the connectivity between the end points of the circuit switched network, namely the ToR-to-ToR connectivity. Hence, circuit availability between a pair of ToR switches can be considered as a direct link between the pair. In the following, we model the periodically reconfigurable ToR-to-ToR connectivity over time as a *periodic evolving graph*.

Periodic evolving graph: Consider the periodic reconfigurable topology described in §2.1 (shown in Figure 3b). ToR switches connect to the optical switches via uplinks. The optical spine layer in turn establishes a circuit between ToR pairs *periodically* as shown in Figure 4a since the optical switches reconfigure periodically according to a fixed schedule. To this end, we represent the ToR-to-ToR connectivity as *periodic evolving graph* denoted by $\mathcal{G} = (V, \mathcal{E})$. Specifically, $\mathcal{G}$ is a periodic sequence of directed graphs defined for timeslots $t \in [0, \infty)$ where each timeslot is of Δ duration. We denote the graph at time t as $\mathcal{G}_t = (V, \mathcal{E}_t)$, where V is the set of all ToR switches and the edge set $\mathcal{E}_t$ represents the circuit availability between the ToR switches at time t. The sequence of edge sets $\mathcal{E}_t$ and consequently the evolving graph $\mathcal{G}$ are periodic with period of Γ timeslots. The period in turn relates to the periodic switching schedule of the optical switch. We denote the capacity of circuit between ToR pairs at any time t as $c_t(e)$ for all $e \in \mathcal{E}_t$. We explicitly set $c_t(e) = 0$ if $e \notin \mathcal{E}_t$. Further, we model the "latency tax" Δ_u due to reconfigurations by limiting the access to

[2]Each ToR uplink is a set of two SERDESes [2, 22] which are uni-directional corresponding to ToR output (pentagons) and input (circles) as depicted in Figure 3b.



(a) Example of a periodic evolving graph with period $\Gamma = 2$ depicting the ToR-to-ToR connectivity over time; $n_t = 16$, $n_u = 2$, $n_s = 2$ and $n_p = 16$.

(b) The corresponding *Emulated* graph (static) over time.

Fig. 4. Given the number of ToR switches n_t with uplinks n_u and a periodic sequence of matchings corresponding to each circuit switch, the topology is a periodic evolving graph which *emulates* a static graph over time.

an edge $e \in \mathcal{E}_t$ to only a $(1 - \Delta_u)$ fraction of a timeslot (from the start of every timeslot). This model serves as an entry point for our formal analysis of throughput, delay, and buffer requirements of reconfigurable topologies in the next section.

3 MOTIVATION: FUNDAMENTAL TRADEOFFS OF PERIODIC RDCNS

We now provide a more detailed motivation for our work. Specifically, we analytically derive the throughput of periodic RDCNs (§3.1) and its relation to delay (§3.2). We further study the buffer requirements (§3.3) and highlight that the topologies introduce a fundamental tradeoff between throughput, delay and buffer. We then present remarks and practical implications of our results (§3.4). Finally, we discuss the optimization opportunities in the design of periodic reconfigurable topologies (§3.5).

3.1 Throughput of Periodic RDCNs

We first study the throughput of periodic reconfigurable topologies. Specifically, we prove that the throughput of a periodic evolving graph is equivalent to that of a unique static graph. Following this result, we re-investigate the throughput of static graphs and obtain a more general result which is in line with the definitions and intuitions discussed in the literature. Our result asserts that the throughput of periodic reconfigurable topologies as well as static topologies is inversely proportional to *average route length* as opposed to shortest path lengths used in the most recent work [16].

In the following, we briefly introduce our definitions in the context of periodic evolving graphs. Prior works [16, 23, 24] define throughput of a static topology given a demand matrix $\mathcal{M}$ as the maximum scaling factor θ such that there exists a feasible flow that satisfies the scaled demand $\theta \cdot \mathcal{M}$. A more formal definition appears in Appendix A. While prior works studied the throughput of static topologies, we are not aware of a formal definition and study of the throughput of general dynamic topologies. Initial studies on dynamic topologies informally define throughput of existing systems (e.g., RotorNet) which emulate a complete graph, specifically in terms of demand completion times [13]. We emphasize that our definitions and results in this paper hold for *any* periodic reconfigurable topology. Next, we define the throughput of dynamic topologies using the notion of *temporal flows* which are formally defined in Appendix B.

Demand matrix $\mathcal{M}$: Given a set of vertices V, a demand matrix specifies the demand rate between every pair of vertices in bits per second defined as $\mathcal{M} = \{m_{u,v} \mid u \in V, v \in V\}$ where $m_{u,v}$ is the demand between the pair u, v. A saturated demand matrix is such that the total demand originating at a source s equals its outgoing capacity and the total demand terminating at a destination d equals

its incoming capacity. We consider *saturated* demand matrices since we focus on the maximum achievable throughput. Specifically, saturated demand matrices allow for studying the maximum demand that can be routed in a topology within the capacity constraints.

Legal flow: A legal flow in a static graph obeys capacity constraints and satisfies flow conservation at all times (Appendix A, Definition 4). In the evolving graph, a legal *temporal* flow (Appendix B, Definition 8) is subject to capacity constraints at all times and conserves flow in every period (Γ timeslots).

Throughput $\theta(\mathcal{M}, \mathcal{F})$ of a flow: Following the seminal works on throughput [25–27][3], given a demand matrix $\mathcal{M}$ specified in bits per second, we define the throughput achieved by a *temporal flow* $\mathcal{F}$ in a periodic evolving graph as the scaling factor $\theta(\mathcal{M}, \mathcal{F})$ such that the flow $\mathcal{F}$ satisfies the scaled demand matrix $\theta(\mathcal{M}, \mathcal{F}) \cdot \mathcal{M}$ where $\mathcal{F}$ is subject to capacity constraints at all times and flow conservation in every period (Γ timeslots). We assume that certain amount of buffer is available at each node to hold traffic until it can be forwarded to the next hop or destination.

Throughput $\theta(\mathcal{M})$ of the graph for a demand matrix: Since we are interested in the ideally achievable throughput, following prior work [16, 23, 24], we define the throughput of a periodic evolving graph as follows: given a demand matrix $\mathcal{M}$, throughput is the maximum scaling factor $\theta(\mathcal{M})$ such that the scaled demand matrix $\theta(\mathcal{M}) \times \mathcal{M}$ is feasible in the network i.e., $\theta(\mathcal{M})$ is the maximum throughput across all feasible flows: $\theta(\mathcal{M}) = \max_{\mathcal{F}} \theta(\mathcal{M}, \mathcal{F})$.

Throughput θ^* of the graph: The *throughput* θ^* is the maximum scaling factor $\theta(\mathcal{M})$ for a worst-case demand matrix i.e., θ^* is the minimum throughput across all the *saturated* demand matrices: $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$. In essence, the network can always support $\theta^* \cdot \mathcal{M}$ demand for *any* saturated demand matrix, where a saturated demand matrix is such that the total demand originating at any source, and received by any destination equals the total node capacity.

We study the throughput of periodic RDCNs in relation to the graph emulated by the periodic topology which we call the *emulated graph*, formally defined below.

Emulated graph: Consider a periodic evolving graph (see §2.2) $\mathcal{G} = (V, \mathcal{E})$ with edge capacities $c_t(e)$ for every edge e at time t. Consider a static (directed, multi, edge-labeled) graph which is a function of the evolving graph $\mathcal{G}$ defined as $G(\mathcal{G}) = (V, E)$, where $E = \{(e, \ell) \mid e \in \mathcal{E}_\ell,\ \ell \in [0, \Gamma)\}$ is the union of edges of the evolving graph over one period of time-interval and the labels correspond to the time in the evolving graph. Let the capacity of an edge $(e, \ell) \in E$ in the static graph be $\hat{c}(e, \ell)$ where $\hat{c}(e, \ell) = \frac{(1-\Delta_u)}{\Gamma} \cdot c_\ell(e)$. We refer to this static graph as *emulated graph*. Figure 4b illustrates an example of the emulated graph corresponding to the periodic evolving graph in Figure 4a. Our formal definition appears in Appendix B.3.

Our first main result, which is the basis for our methodology, is that the throughput of a periodic evolving graph is equivalent to the throughput of its corresponding emulated graph.

Theorem 1 (Relation to Emulated Graph)**.** *Given a demand matrix $\mathcal{M}$, any legal temporal flow $\mathcal{F}$ in the periodic evolving graph $\mathcal{G}$ with throughput $\theta(\mathcal{M}, \mathcal{F})$ can be converted to a legal flow F in the emulated graph $G(\mathcal{G})$ with the same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$ and vice versa. Consequently the periodic evolving graph and the emulated graph have the same throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$: the maximum scaling factor given a demand matrix $\mathcal{M}$; they further have the same throughput θ^* for a worst-case demand matrix, where $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$.*

[3]The maximum concurrent flow problem [25] considers that the demand between each pair s, d is equal. We relax this assumption for the throughput maximization problem in our networking context similar to prior work [23].

Proof sketch. We present the formal proof of Theorem 1 in Appendix C and provide a proof sketch here. We first define what we call *extended paths* for the static emulated graph that allows a conversion between paths in the periodic evolving graph and the corresponding static emulated graph. While the standard definition of path in a static graph is a sequence of edges connecting a source and a destination, our extended path definition differs in two aspects. Given a temporal path in the periodic evolving graph, we define the extended path in the static emulated graph as the sequence of edges traversed in the temporal path where each edge is associated with a label corresponding to the time when the edge was accessed in the temporal path. Further, each extended path consists of a unique identifier. Since the evolving graph is periodic, we mainly focus on the foundation set of temporal paths that start in the first period. As a result, there exists a one-to-one mapping between the foundation set of temporal paths in the periodic evolving graph and the extended set of paths in the static emulated graph. Exploiting this relation, we prove the following: *(i)* we prove in Lemma 1 that if a legal flow in the emulated graph has throughput θ over the set of extended paths, then there exists a legal temporal flow in the periodic evolving graph with the same throughput; *(ii)* we prove in Lemma 2 that if a legal temporal flow achieves throughput θ in the evolving graph, then there exists a legal flow in the emulated graph with the same throughput over the set of extended paths; *(iii)* finally, we prove in Lemma 3 that a flow in the emulated graph over the set of *all* paths has the same throughput as that of a flow over the set of extended paths. Using the above three results, we prove our main claim in Theorem 1.

Discussion. Theorem 1 enables us to study the throughput of periodic reconfigurable topologies with the techniques used in static graphs. We emphasize that our result is *general* for periodic reconfigurable topologies and is not specific to the existing systems; for instance the emulated graph could indeed be an expander graph, or even a non-regular graph, but our result still holds. We believe that our proof renders useful not only to study throughput maximization problem but also other variants of flow problems. For example, by using our technique to convert flows from the static to the periodic graph and vice versa one could easily derive a relation between the metric of interest in each graph. We leave this exploration for future work.

We now shift our focus to obtain the throughput upper bound of static topologies for two reasons. First, from Theorem 1, the throughput of a periodic reconfigurable topology is equivalent to its corresponding emulated graph (a *static graph*). Second, the best known throughput upper bound [16] for static topologies is specific to certain datacenter topologies and is not general enough (see Appendix A). To this end, we revisit the problem and analyze it again using the formal definitions typically used in the literature. We find that, the throughput of a static topology is directly proportional to the total capacity and inversely proportional to the *average route length*, which generalizes the recently derived throughput upper bound [16] for shortest paths.

Theorem 2 (Throughput)**.** *Given a demand matrix $\mathcal{M}$ and a flow F, the throughput $\theta(\mathcal{M}, F)$ of a static graph represented as $G = (V, E)$ is given by Equation 1. The graph G has throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$: the maximum scaling factor given a demand matrix $\mathcal{M}$; and has throughput $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$ for a worst-case demand matrix.*

$$\theta(\mathcal{M}, F) \leq \frac{\hat{C}}{M \cdot \mathrm{ARL}(\mathcal{M}, F)} \tag{1}$$

where $\hat{C} = \sum_{e \in E} \hat{c}(e)$ is the total capacity of the network, $M = \sum_{s,d \in V} m_{s,d}$ is the total demand for the network and $\mathrm{ARL}(\mathcal{M}, F) = \sum_{s,d \in V} \sum_{p \in P_{s,d}} \frac{m_{s,d}}{M} \cdot r_p \cdot \mathrm{len}(p)$ is the average route length *for $\mathcal{M}$ and F, where r_p is the fraction of demand transmitted on the path p.*

Proof sketch. Our proof of Theorem 2 appears in Appendix D. We rely on *average route length* formally defined in Definition 12, i.e., the weighted sum of all path lengths where the weight

corresponding to each path is the fraction of total demand transmitted on the path. The rest of the proof follows by capacity constraints and flow conservation rules.

Discussion. Theorem 2 indeed expresses that the throughput maximization problem essentially boils down to minimizing *average route length* as opposed to shortest path lengths [16] or average shortest paths [24]. Both Theorem 1 and Theorem 2 provide a concrete understanding on the achievable throughput of periodic reconfigurable topologies. First, Theorem 1 provides a relation between the throughput of periodic reconfigurable topologies and the throughput of static topologies. Second, Theorem 2 provides the throughput of static topologies more concretely. Finally, for the worst-case demand matrix, we rely on an important result stated in [16] (and even earlier informally in [23, 24]) and claims that the worst-case demand matrix in a static topology is a specific permutation matrix (longest-matching). Consequently the worst-case demand matrix for *any* periodic reconfigurable topology, due to Theorem 1, is a specific permutation matrix in the corresponding emulated graph[4].

However, so far we still assumed that *(i)* a certain amount of buffer is available at each intermediate node and *(ii)* any amount of delay is tolerable. Hence, we study two important questions in the remainder of this section. We first seek to understand whether high throughput provided by periodic reconfigurations comes at the cost of inflated delay.

(Q1) Delay: *What is the relation between throughput and delay in a periodic reconfigurable topology?*

Given the increasing gap between capacity growth and switch buffers, we are further interested in investigating whether periodic reconfigurable topologies address the near-end of Moore's law w.r.t buffer requirements.

(Q2) Buffer: *How much buffer is required at each node to achieve the throughput upper bound stated in Theorem 2?*

3.2 Delay of Periodic RDCNs

The periodic reconfigurations naturally introduce a certain delay. In our context, delay is the time it takes for a packet sent from a source to reach its destination in the periodic evolving graph, *without experiencing congestion from other packets.* The maximum delay for a path δ is bounded by $\text{len}(\delta) \cdot \Gamma$ since every consecutive edge in the path is available within Γ timeslots (period). We are mainly interested in the maximum delay incurred by feasible paths which typically reflects in the tail latencies observed in a datacenter. In the following, we state the relation between throughput θ^* and delay.

Theorem 3 (Delay)**.** *Given a demand matrix $\mathcal{M}$, a n_u-regular periodic evolving graph $\mathcal{G}$ emulating a d-regular graph with a period of Γ timeslots each of duration Δ and a flow $\mathcal{F}$ that achieves throughput $\theta(\mathcal{M}, \mathcal{F})$, the average route delay $\text{ARD}(\mathcal{M}, \mathcal{F})$ and the maximum delay L_{max} are given by*[5]*,*

$$\begin{aligned} L_{max} \geq \text{ARD}(\mathcal{M}, \mathcal{F}) &= \text{ARL}(\mathcal{M}, \mathcal{F}) \cdot \Gamma \cdot \Delta \\ &\geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta(\mathcal{M}, \mathcal{F})}\right) \end{aligned} \tag{2}$$

where $\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \sum_{\delta} \frac{m_{s,d}}{M} \cdot r_\delta \cdot L(\delta)$ is the average route delay *for $\mathcal{M}$ and $\mathcal{F}$, where r_δ is the fraction of demand transmitted on the legal temporal path δ, and $L(\delta)$ is the delay of the*

[4] A similar but less general result was recently claimed for specific RDCN designs [13]. In particular, it holds only for designs which emulate a complete graph, while our result holds for *any* emulated graph due to our new result in Theorem 1.
[5] Ω is the asymptotic lower bound notation.

path δ, $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$ is the total demand. In particular, for a worst-case demand matrix, the maximum latency is bounded by

$$L_{max} \geq \Omega \left(\frac{d \cdot \Delta}{n_u \cdot \theta^*} \right) \tag{3}$$

Proof sketch. We use the argument of flow conservation to prove our claim. Our full proof appears in Appendix E. Specifically, consider a temporal path δ with delay $L(\delta)$ between a source s and a destination d; r_δ be the fraction of demand $m_{s,d}$ transmitted on the path δ. Notice that the destination d does not receive the flow initially until $L(\delta)$ duration due to the path delay. From then on, in order to satisfy the demand, source s transmits $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot r_\delta$ amount of flow in every period and destination d receives $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot r_\delta$ amount of flow is every period. Due to conservation of flow, the initially transmitted data (flow volume) until $L(\delta)$ time i.e., $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot r_\delta \cdot L(\delta)$ circulates (moves between s-d) in the network in every period. As a result, the overall capacity consumed can be written as $\theta(\mathcal{M}, \mathcal{F}) \cdot \frac{ARD(\mathcal{M}, \mathcal{F})}{\Gamma \cdot \Delta}$ where ARD is the average route delay (Definition 14). However, the overall capacity utilized can also be written in terms of path lengths i.e., $\theta(\mathcal{M}, \mathcal{F}) \cdot \text{ARL}(\mathcal{M}, \mathcal{F})$ where ARL is the average route length (Definition 12). By equating the two, we obtain our result in Eq. (2). Eq. (3) follows by considering the throughput θ^* for a worst-case demand matrix.

Discussion. Using Theorem 3, we illustrate the throughput and delay relation for different values of d in Figure 1. Here d is the degree of the graph emulated by a periodic reconfigurable topology. Note that the worst-case demand matrix (a permutation matrix) specifies non-zero demands between ToR pairs at maximum distance in the graph [16] and hence the average route lengths ARL is close to the diameter which is bounded by $\log_d(n_t)$ for d-regular graphs. Hence the throughput θ^* is inversely proportional to $\log_d(n_t)$ whereas the delay based on Eq. (3) is proportional to $d \cdot \log_d(n_t)$. Specifically, we notice that the existing designs which emulate a complete graph achieve the highest throughput but at the cost of high delay.

3.3 Buffer Requirements of Periodic RDCNs

With the understanding of the relation between throughput and delay of periodic reconfigurable topologies, we now study their buffer requirements. We assume that each node is equipped with a memory region shared across the entire device and hence buffer is an aggregate value per node in our analysis. This model is similar to shared memory architectures with complete sharing [28]. Our analysis reveals an interesting inequality for the required buffer, which is conceptually of the well-known "bandwidth-delay product" form.

Theorem 4 (Buffer)**.** *Given a demand matrix $\mathcal{M}$, a periodic evolving graph $\mathcal{G}$ requires at least $\hat{B}$ total amount of buffer in the network in order to achieve throughput $\theta(\mathcal{M}, \mathcal{F})$.*

$$\hat{B} \geq (\theta(\mathcal{M}, \mathcal{F}) \cdot M) \cdot \text{ARD}(\mathcal{M}, \mathcal{F}) \tag{4}$$

where $\hat{B} = \sum_{u \in V} B(u)$ is the total buffer of the network and $B(u)$ is the available buffer at a node u; $M = \sum_{s,d \in V} m_{s,d}$ is the total demand for the network; $\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \sum_{\delta} \frac{m_{s,d}}{M} \cdot r_\delta \cdot L(\delta)$ *is the* average route delay *for $\mathcal{M}$ and $\mathcal{F}$, where r_δ is the fraction of demand transmitted on the legal temporal path δ.*

Proof sketch. While the proof of Theorem 3 accounts for flow conservation with in one period time interval, the proof of Theorem 4 follows by carefully accounting the flow that is in transit within one period time interval before conservation applies. Our full proof appears in Appendix E.

Discussion. Theorem 4 reveals a non-trivial tradeoff across the spectrum of periodic reconfigurable topologies which we discuss in detail in §3.5. Our result for the required buffer in Theorem 4 intuitively resembles the "bandwidth-delay product" commonly used in the TCP literature. A "pipe"

must have at least a bandwidth-delay product amount of bytes in transit (or inflight) in order to achieve full utilization. Similarly, our result shows that the total buffer in a periodic reconfigurable network must be at least the product of total demand (in bits per second) and the average route delay. The required buffer stems from the waiting times at intermediate nodes along a path due to the periodic reconfigurations. We believe this analogy may also render useful for instance to predict the overall graph throughput based on the configuration of transport protocols. For example, the window size[6] of a transport protocol typically relates to the amount of buffering, and the maximum window size may relate to the overall periodic graph throughput. We leave it for future work to study such relations in detail.

3.4 Remarks and Discussion on Theorems 1-4

Our results in the previous sections allow us to answer and shed light on several basic questions also discussed in the literature [2, 3, 29].

Can buffering at the end-hosts instead of ToRs alleviate the problems of excessive buffering? Our result in Theorem 4 expresses the *total* amount of buffering required in the network. We note that this buffering can either be done at the ToR switches or can be moved to the end-hosts. Several works in the past [3, 29] informally discussed the need for in-network buffers but outside the circuit switching layer[7]. Two-hop and in general multi-hop routing further worsens the need for additional in-network buffers in periodic reconfigurable networks [3]. To this end, prior works proposed to move the buffering from ToR switches to the end-hosts with specialized synchronization and packet pause/unpause techniques between the end-hosts and the ToR switches [29]. These techniques are motivated based on the fact that end-host DRAM memory is cheap and abundant while on-chip ToR buffers are costly and limited in size. Nevertheless, excessive buffering at the end-hosts still poses the well-known problems of queuing delay, although it alleviates the problems of scarce buffer resource at the ToRs. The question of buffering at the end-hosts vs ToR switches poses an inherent tradeoff between *a)* simplicity by buffering at the ToR but at the cost of large on-chip costly buffers and *b)* cost-effectiveness by buffering at the end-hosts but at the cost of complex techniques that consume additional CPU resources and potentially added processing delays.

Can nanosecond reconfiguration mask the problems of delay and buffer needs? Notice that delay (from Theorem 3) is directly proportional to the timeslot duration Δ. Naturally, if the reconfiguration delay Δ_r is small, the timeslot duration Δ can also be reduced. As a result, the maximum delay and the average route delay can be significantly reduced. However, notice that the buffer requirements (from Theorem 4) are directly proportional to the timeslot duration Δ as well as the total capacity e.g., the buffer requirements would remain the same if all the links in the network are increased in capacity by 2x even if the reconfiguration time is reduced by 2x.

What is the difference between the degree of the emulated graph and the physical topology? We emphasize that the emulated graph is obtained by the union of edges of the periodic evolving graph (ToR-to-ToR connectivity) over one period of a time-interval. Hence, the node degree of the emulated graph is the number of ToR switches that are directly connected to a given ToR switch (over one period). In contrast, the node degree of the physical topology is the number of circuit switches that are directly connected to a given ToR switch at any time. In particular, the node degree in the physical topology is given by n_u: the number of uplinks. However, for example, both RotorNet and Sirius emulate a complete graph and hence have the same node degree in the emulated graph given by n_t: the total number of ToR switches. Hereafter in this paper, the emulated

[6]The window size of a transport protocol is typically the maximum amount of bytes allowed to be in transit at any point in time.

[7]Recall that circuit switches are bufferless.

graph and its degree play an important role (§4) especially as seen in Theorem 1 and Theorem 3 where d is the degree.

Are the results applicable to Valiant routing?

We stress that our results in Theorem 1, 2, 3, 4 are general for an ideal routing scheme and are *not* based on Valiant load balancing. We only assume the specific case of Valiant load balancing later in §4.

3.5 Tradeoffs & Optimization Opportunity

Interestingly, the relation between throughput and delay is a concave function without buffer constraints (Theorem 2) but is a convex function for a fixed amount of buffer (Theorem 4). Figure 1 illustrates the tradeoffs that arise across the design space of periodic reconfigurable topologies. Specifically, for a certain fixed amount of buffer, existing designs emulating a complete graph are no longer throughput optimal as the available buffer simply cannot hold enough traffic to achieve the ideally achievable maximum. Further, throughput and delay pose a fundamental tradeoff in terms of performance.

Although periodic reconfigurable datacenter designs were developed in view of bandwidth scaling problem posed by the gradual end of Moore's law, the required buffer with linear relation to throughput does not indicate high scalability in future. Several studies in the recent past show an increasing gap between the capacity growth and switch buffers [14, 15]. Specifically, the top-of-the-rack buffers are extremely shallow and the buffer per port per Gbps has been gradually decreasing over the recent years. In the following, we summarize our key motivation to explore the design space of topologies.

- **Throughput:** If buffer is not a concern, then emulating a complete graph (existing designs) results in high throughput but at the cost of high delay.
- **Delay:** If delay is a concern, even without buffer concerns, emulating a d-regular graph where $d < n_t$ results in lower delay but at the cost of throughput.
- **Buffer:** Finally, if buffer is a concern, there exists a d-regular graph that maximizes throughput within the buffer limits.

Our goal in this work is to design a periodic reconfigurable topology that maximizes throughput given the buffer and delay requirements.

4 MARS: NEAR-OPTIMAL THROUGHPUT RDCN WITH SHALLOW BUFFERS

Reflecting on our observations in §3, we study the family of periodic reconfigurable topologies that emulate a d-regular graph. Specifically, our aim is to systematically determine the high throughput topology within the underlying buffer limits.

4.1 Overview

We first give an intuition on our design. Recall that the network consists of n_t ToR switches each with n_u in/out ports interconnected via optical circuit switches.

Mars emulates a d-regular graph with near-optimal throughput: Given the importance of the graph emulated by the topology, Mars emulates a "good" d-regular graph with the degree d optimized for throughput with the limited available buffer. Specifically, the degree d of the emulated graph influences two factors *(i)* average path lengths which relates to throughput and *(ii)* delay which relates to the buffer requirements. Among the set of all d-regular graphs, we are interested in the "good" graphs with diameter close to $\log_d(n_t)$, where n_t is the number of ToR switches. Note that the lower bound for diameter of any d-regular graph is $\log_d(n_t)$. Following prior work [2], we assume Valiant load balancing [30] for the routing scheme for simplicity. This inflates the average route length by a factor of two i.e., $2 \cdot \log_d(n_t)$. Notice that emulating a K_{n_t} complete graph with

large degree would result in shorter average route length. However, as we will see later, emulating a degree $d \leq n_t$ results in better delay and better throughput under limited buffer.

A parametrized approach: We parametrize our design based on two values *(i)* delay requirement L and *(ii)* buffer limit B. The degree d of the emulated graph of MARS depends on the delay and buffer requirements.

Delay and buffer requirements: MARS finds a balance between throughput, delay and buffer. Specifically in Figure 1, MARS is the topology at the intersection of throughput upper bound with and without buffer restrictions. To this end, MARS maximizes the throughput while finding a balance in delay and buffer requirements.

4.2 Properties of MARS

We first express the throughput of MARS without delay and buffer constraints which gives an intuition on the ideally achievable throughput.

Theorem 5 (Unconstrained Throughput Upper Bound)**.** *The throughput of MARS connecting n_t ToR switches each with n_u in/out ports, emulating a d-regular graph without any delay and buffer constraints, under Valiant load balancing, is given by:*

$$\theta^* = \frac{1}{ARL} \approx \frac{1}{2 \cdot \log_d(n_t)}$$

First the diameter of MARS's emulated graph is close to $\log_d(n_t)$. Second, we then argue that the worst-case permutation demand matrix specifies non-zero demands between ToR pairs which are separated by a distance close to diameter. Further, Valiant load balancing inflates the *route lengths* by a factor of 2 e.g., similar to existing designs [2] i.e., ARL $= 2 \cdot \log_d(n_t)$ under worst-case demand matrix and hence $\theta^* = \frac{1}{2 \cdot \log_d(n_t)}$. For instance, in the case of $d = n_t$ (emulating a complete graph), we obtain $\theta^* = \frac{1}{2}$.

With the understanding of the throughput of MARS, we can now relate its throughput to delay. Using our result in Theorem 3, the delay L_{max} of MARS is related to throughput as $\Omega(\frac{d \cdot \Delta}{n_u \cdot \theta^*}) = \Omega(\frac{2 \cdot (\log_d n_t) \cdot d \cdot \Delta}{n_u})$. Given a constraint on delay i.e., the topology must ideally incur a delay less than L, we state the optimal degree for MARS in the following.

Theorem 6 (Optimal degree d with delay constraints)**.** *The optimal d-regular graph emulated by MARS that maximizes throughput (given the delay requirement L and under Valiant load balancing) has a degree d given by,*

$$d = \lfloor e^{-\mathcal{W}(k)} \rfloor$$

where $k = \frac{-2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$; $\mathcal{W}$ is the Lambert W function [31]; $\ln(.)$ is the natural logarithm and e is the Euler's number.

We provide a proof sketch here. We begin by equating the delay of MARS and the desired delay L. We then use the property that if $y^y = k$ where k is some constant, then $y = e^{\mathcal{W}(ln(k))}$. Our full proof appears in Appendix F.

Our observations in §3 reveal that a key tradeoff arises across the buffer requirements and the achievable throughput. We are now interested in determining the optimal degree d of MARS for a limited buffer B at each node. As an intuition, a topology requires more buffer as the product of the scaled demand and the average route delay increases.

Theorem 7 (Optimal degree d with buffer constraints)**.** *The optimal degree d for the emulated graph of MARS independent of the specific flow that maximizes throughput, given a limited buffer $B \leq n_t \cdot \Delta$*

at each node, is given by,

$$d = \lfloor \frac{B}{c \cdot \Delta} \rfloor$$

where c is the capacity of every edge in the topology and Δ is the timeslot value.

Our proof is a straight-forward extension using our results in Theorem 3, 4. The full proof appears in §F. From Theorem 7, emulating a complete graph like the existing designs would require a buffer of $\frac{n_t}{n_u} \cdot (n_u \cdot c) \cdot \Delta = n_t \cdot c \cdot \Delta$ where $n_u \cdot c \cdot \Delta$ is the amount of data that can be sent out or received in a single timeslot and $\frac{n_t}{n_u}$ is the period for a complete graph. In contrast, a d-regular graph would only require $d \cdot c \cdot \Delta$ amount of buffer and achieves better throughput under limited buffer compared to a periodic topology emulating complete graph.

4.3 Interconnect

Switch size: Given n_t number of ToR switches each with n_u uplinks, Mars requires a switch size of at least $n_p = n_t$ similar to prior work [3, 4]. We need n_u such circuit switches to interconnect the datacenter. We leave it for future work to study the feasibility of using circuit switches with lower port count.

Wiring: Each ToR uses its n_u uplinks to connect to each of the n_u circuit switches.

Matchings: We first generate a d-regular directed graph. Specifically, in generating the emulated graph, we choose the d-regular graph for which the diameter approaches the lower bound of $\lceil \log_d(n_t) \rceil$ [32–34] e.g., deBruijn digraphs [35]. A straight-forward argument shows that a d-regular directed graph can be decomposed into d perfect matchings. The idea is to recursively find a 1-factor and delete it from the graph where each 1-factor by definition is a perfect matching [36, 37]. We shuffle the resulting d matchings and randomly assign $\frac{d}{n_u}$ matchings to each of the n_u circuit switches in the topology. Each circuit switch then cycles through the $\frac{d}{n_u}$ matchings periodically. We note that decomposing the d-regular emulated graph into matchings can be computationally expensive. However, this is performed only once at the time of deployment and the circuit switches cycle through the installed matchings periodically thereafter.

Note that the interconnect for Mars can be implemented using the same hardware as in prior work [2–4]; in fact, the only change required in Mars compared to these systems concerns the matchings configurations.

4.4 Example: deBruijn-based Emulated Graph

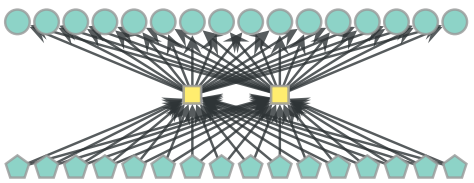
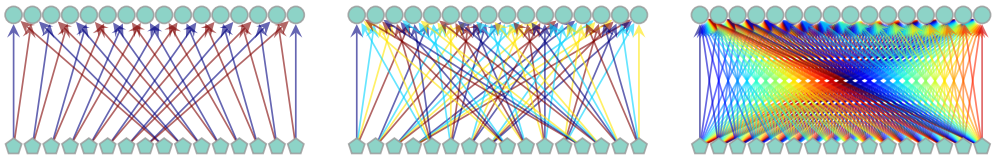


Fig. 5. An example of a periodic RDCN interconnect with 16 ToR switches each with 2 in/out ports; 2 circuit switches each with 16 in/out ports. For better visualization, the ToR switch outputs are depicted as pentagons and inputs are depicted as circles. Circuit switches are depicted as squares.

We walk through an example to better illustrate the topology construction and the inherent tradeoffs. We consider an example with $n_t = 16$ ToR switches each with $n_u = 2$ in/out ports. The interconnect requires $n_s = n_u = 2$ optical circuit switches each of size $n_p = n_t = 16$. Each ToR uses its uplinks to connect to each of the two circuit switches as shown in Figure 5. Circuits are set up for $90\mu s$ duration and it takes $10\mu s$ to reconfigure (timeslot $\Delta = 100\mu s$ for each matching) similar to RotorNet switches [3]. All the links have a capacity of $c = 400$Gbps.

Before constructing the sequence of matchings, we take one of the factors into consideration *(i)* buffer size of ToR switches and *(ii)*



(a) Static 2-regular Emulated Graph (2 Matchings)

(b) 4-regular deBruijn Emulated Graph (4 Matchings)

(c) Complete Emulated Graph (16 Matchings)

Fig. 6. A periodic reconfigurable topology (shown in Figure 5) can emulate a spectrum of graphs depending on the matchings schedule in the circuit switches. **(a)** A static emulated graph has the least delay but at the cost of throughput. **(b)** MARS finds the best degree of the emulated graph in order to achieve near-optimal throughput based on the available buffer and delay tolerance. **(c)** Existing designs emulate a complete graph trading delay for throughput and suffer in terms of throughput with shallow buffers. Each color in the figure corresponds to a matching. Since there are two circuit switches (shown in Figure 5), at any point in time, the union of two matchings corresponds to the ToR-to-ToR connectivity.

latency tolerance. Using Theorem 6 and Theorem 7, we determine the optimal degree d to maximize throughput. It remains to generate a d-regular emulated graph with optimal diameter and decompose it to a sequence of matchings which are then deployed in the circuit switches.

We consider the spectrum of deBruijn digraphs in generating the emulated graph. Specifically, for n_t ToR switches (vertices), $G = (V, E)$ is a deBruijn graph (emulated graph), where $V = \{0, 1, ..., n_t - 1\}$ is the set of vertices and $E = \{(u, v) \mid v \equiv (u \cdot d + a) \textbf{ mod } n_t, a \in \{0, 1, ..., d - 1\}\}$ is the edge set [33]. It is known that deBruijn graphs have a low diameter of $\lceil \log_d(n_t) \rceil$ which is close to Moore's bound for diameter. Our emphasis on the graph diameter is due to Theorem 2 as the average route length gets closer to diameter under a worst-case demand matrix which specifies non-zero demand between pairs at maximum distance in the graph.

Using the optimal degree d based on Theorem 6 and Theorem 7, we generate a d-regular directed deBruijn graph and decompose the edge set into d matchings. We shuffle the d-matchings and assign $\frac{d}{2}$ number of matchings to each of the two circuit switches. In Figure 6, we show three emulated graph instances which differ in the degree d for $n_t = 16$ ToR switches each with $n_u = 2$ in/out ports. We note that the standard complete graph K_{n_t} has a degree of $n_t - 1$ without self-loops. For simplicity, we consider that the optical circuit switches match every ToR to every ToR including one self-loop i.e., degree is n_t instead of $n_t - 1$. Table 1 summarizes our example. We next walkthrough the throughput, delay and buffer requirements of different design choices including MARS.

Topology	Throughput	Delay	Buffer
1	0.125	≈ 0	≈ 0
2	0.5	$1600\mu s$	80MB
3	0.125	$1600\mu s$	20MB
4	0.25	$850\mu s$	20MB

Table 1. Tradeoffs across different design choices based on the degree of the emulated graph for the example topology shown in Figure 6, 5.

1 **Static uni-regular DCNs:** With two circuit switches in the topology, each deployed with only one matching results in a static topology[8] as shown in Figure 6a. While this is an extreme choice given the reconfigurable circuit switches, it is interesting to observe the throughput vs buffer requirements. Specifically, the diameter is 4 resulting in a throughput of $\frac{1}{8}$ (from Theorem 5) for a worst-case permutation demand matrix. However, the delay is nearly zero theoretically since the topology remains static assuming negligible transmission and propagation delays. Since the delay is near-zero, the topology also requires near-zero buffers at each node to achieve the ideal throughput of $\frac{1}{8}$.

2 **Existing designs:** Using 16 matchings in total, with 8 matchings deployed in each of the two circuit switches, the topology emulates a complete graph as shown in Figure 6c which is similar to existing designs [2, 3]. Emulating a complete graph results in optimal throughput of $\frac{1}{2}$ but at the expense of high delay as much as $16 \cdot \Delta = 1600\mu s$ (from Theorem 3). In order to achieve the optimal throughput of $\frac{1}{2}$, each ToR switch would require significantly large buffers as much as 80MB in our small scale example topology.

3 **Existing designs under resource constraints:** While emulating a complete graph as shown in Figure 6c may result in optimal throughput, the achievable throughput critically depends on the available buffer at each ToR switch. For instance, if each ToR switch is equipped with only 20MB buffer, then the throughput drops to $\frac{1}{8}$ (from Theorem 4). Notice that even with 10MB buffer, a simple static topology achieves similar throughput with significantly lower delay (see above).

4 **Mars:** Our approach leverages the insights from §3. Based on the available buffer (say $B = 20$MB) and the delay requirements (say $L = 850\mu s$), we systematically determine the degree d of the emulated graph to be emulated by the reconfigurable network. In this case, the optimal degree turns out to be $d = 4$ both in terms of delay L (from Theorem 6) and available buffer B (from Theorem 7). The topology has a diameter of $\log_4(16) = 2$ and achieves much higher throughput: $\frac{1}{4}$ in our example.

5 EVALUATION

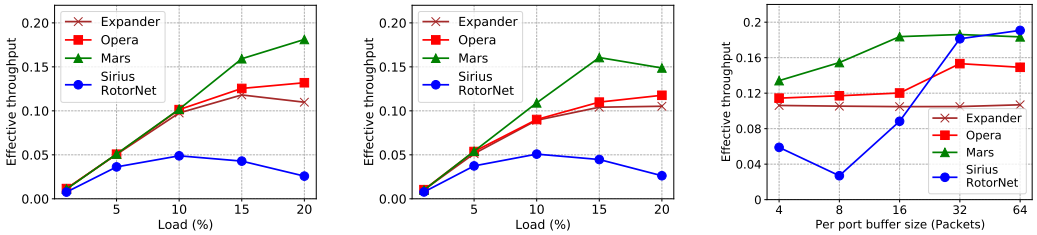
We evaluate Mars and compare against the existing approaches in the design space. Our evaluation aims at answering four main questions.

***(Q1)* Can Mars improve the throughput in datacenters?** Our evaluation shows that, Mars improves the throughput of existing datacenter designs by up to 64% compared to an expander DCN, by up to 37% compared to Opera [4] and by more that 4x compared to Sirius [2] and RotorNet [3] under Valiant load balancing.

***(Q2)* How does Mars perform under shallow buffers?** Even under extremely shallow buffers, Mars significantly outperforms existing approaches by improving the throughput by up to 6x at moderate load conditions. At low loads, Mars performs similar to existing approaches.

***(Q3)* Can Mars improve the FCTs of short flows?**
We find that Mars does not trade latency for throughput. Indeed Mars's low buffer requirements to achieve high throughput also contribute to better latency even under permutation demand matrices. Our evaluation shows that Mars can improve the FCTs of short flows by up to 96% compared to expander DCN, by up to 75% compared to Opera and by up to 87% compared to RotorNet and Sirius under Valiant load balancing.

[8]In this case, since there are no reconfigurations, we set the timeslot value as $\Delta \approx 0$, the latency tax due to reconfigurations $\Delta_u = 0$ and the period $\Gamma = 1$.



(a) Effective throughput under all-to-all demand matrix and websearch workload with 8 packets per port buffer size.

(b) Effective throughput under random permutation demand matrix and websearch workload with 8 packets per port buffer size.

(c) Effective throughput across buffer sizes under random permutation demand matrix and websearch workload.

Fig. 7. MARS achieves the best throughput under shallow buffers. Existing approaches which emulate a complete graph (Sirius/RotorNet) suffer significantly under limited buffer (a,b) but can gain throughput if buffers are large (c).

***(Q4)* Do long flows benefit from MARS?** Our results show that MARS reduces the FCTs of long flows for various workloads by up to 4% on average compared to existing approaches under various loads and demand matrices.

5.1 Setup

Our evaluation is based on packet-level simulations using HTSIM used in prior work [4, 38].

Interconnect: We consider a datacenter network with 256 servers organized into 64 ToR switches. Each ToR switch has 4 uplinks that connect to 4 rotor-switches [3] with a reconfiguration delay of $1\mu s$. All the links have a capacity of 10Gbps and 500 nanoseconds delay.

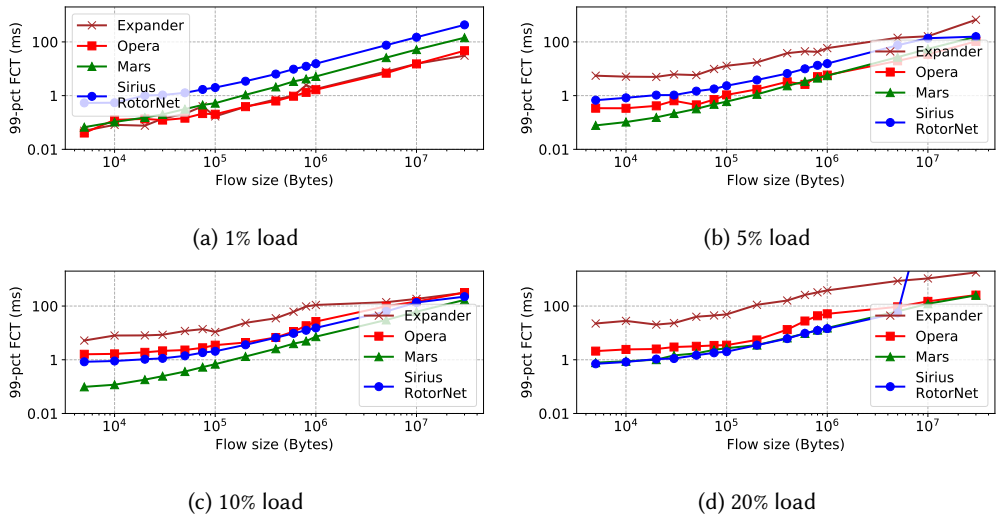
Workload: We generate traffic using websearch [39] and datamining [40] workloads. We evaluate across various loads on the server-ToR links in the range $1-20\%$ for all-to-all and random permutation demand matrices. Note that 20% load is already close to the maximum load that an expander topology can sustain[9]. Flows arrive according to a Poisson process and we control the mean inter-arrival time to achieve the desired load. In the interest of space, we report our results for the datamining workload in the Appendix and only discuss our results for websearch workload in this section.

Comparisons: We compare MARS to Opera [4], Sirius [2], RotorNet [3] and static expander networks. For expander, we generate $d = 4$ (number of uplinks) random regular graphs G with 64 ToRs as vertices such that $\lambda(G) \leq 2\sqrt{u-1}$ where λ is the second eigen value and $2\sqrt{u-1}$ is the Ramanujan constant. This gives us a Ramanujan graph which is known to be excellent expander.

Metrics: We report server downlink utilization indicating the effective throughput. We also report 99-percentile flow completion times (FCTs) and buffer occupancies.

Configuration: We construct MARS which emulates a deBruijn directed graph of degree 8 (see §4.3) optimized for a shallow buffer size of 8 packets per port in the interconnect described above. Switches are configured with routing information statically at initialization time and packets are sprayed across all equal cost paths. MARS, Sirius [2] and RotorNet [3] use Valiant load balanced

[9] Atleast $\frac{1}{\text{ARL}}$ capacity is sacrificed due to multi-hop routing; where ARL is the average route length. An alternative in order to further increase throughput is to simply"undersubscribe" i.e., with oversubscription < 1. We leave it for future work to analyze such designs which requires a comprehensive cost-analysis for a fair comparison.



(a) 1% load

(b) 5% load

(c) 10% load

(d) 20% load

Fig. 8. For the websearch workload under a random permutation demand matrix, MARS improves the 99-percentile flow completion times for short flows without sacrificing the flow completion times of long flows. Note the log scale on both axes.

paths; expander uses all (edge-disjoint) shortest paths; and Opera uses all (edge-disjoint) shortest paths in each topology slice [4] for short flows and single hop paths for long flows. The long flow cutoff is set to 15MB based on [4]. We use NDP [38] as the transport protocol and set the trimming threshold per port to 8 packets by default based on [38] unless explicitly stated. We vary the packet trimming threshold to vary the maximum buffer size values in our evaluation. Finally, we set minRTO to 1ms.

5.2 Results

MARS significantly improves the throughput: In Figure 7a and Figure 7b, we show the effective throughput achieved by MARS and existing approaches across various loads of websearch workload. Specifically, in Figure 7a for the All-to-All demand matrix, we see that MARS improves the effective throughput at 20% load by 1.37x compared to Opera, by 1.64x compared to expander and by 7.02x compared to Sirius and RotorNet under Valiant load balancing. In Figure 7b, for random permutation demand matrix, we see that MARS improves the effective throughput by up to 1.33x on average compared to Opera and expander; and by 5.66x compared to Sirius and RotorNet. At low loads, MARS achieves similar throughput compared to existing approaches.

MARS outperforms under shallow buffers: Given the high throughput of MARS, we evaluate its performance under various buffer sizes at the ToR switches and compare against existing approaches. In Figure 7c, we see that even at extremely shallow buffers such as 4 packets per port, MARS achieves 1.57x better throughput on average compared to Opera, expander, Sirius and RotorNet. However, as we show in Figure 1, existing systems require significantly more buffer to achieve high throughput. As expected, in Figure 7c, we see that Opera, Sirius and RotorNet increase in throughput with large buffers. Indeed, Sirius and RotorNet can sustain even higher loads up to 50% since they emulate a complete graph with shorter path lengths but this requires extremely large buffers. We omit these results for brevity, given that Sirius and RotorNet require buffer sizes as large as 64 packets per port to achieve 20% throughput in Figure 7c.

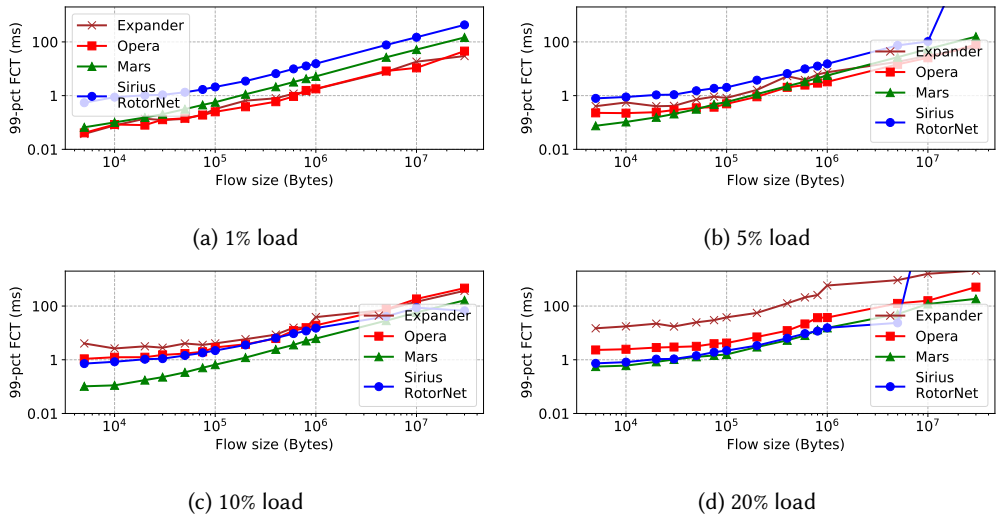


Fig. 9. For the websearch workload under all-to-all demand matrix, as the load increases, MARS outperforms in flow completion times for both short and long flows. Note the log scale on both the axes.

MARS does not require large buffers: In Figure 10, we show the CDF of buffer occupancies at the ToR switches for 20% websearch load and 64 packet per port buffers. We see that MARS and Opera require significantly lower buffer compared to expander, Sirius and RotorNet (under Valiant load balancing). Opera selectively buffers packets at the end-hosts based on their flow size and only routes short flows across multi-hops which contributes its very low buffer requirements as seen in Figure 10. However, Opera achieves 1.22x *lower* throughput compared to MARS as seen in Figure 7c. We also observe that Sirius and RotorNet emulating a complete graph, consume significantly more buffer compared to MARS: 2.09x higher at the tail.

MARS significantly improves FCTs of short flows: The high throughput as well as low buffer requirements of MARS significantly improve the FCTs of short flows. Figure 8 shows the 99-percentile FCTs across various loads for random permutation demand matrix under websearch workload. At 1% load (Figure 8a), we see that MARS achieves similar FCTs for short flows compared to Opera and expander networks and improves upon Sirius and RotorNet by 87.37%. As the load increases, MARS outperforms existing approaches. Specifically, at 5% load (Figure 8b), MARS reduces the FCTs for short flows by 77% compared to Opera, by 98.11% compared to expander and by 88.43% compared to Sirius and RotorNet. Further, at 10% load (Figure 8c), MARS improves the FCTs

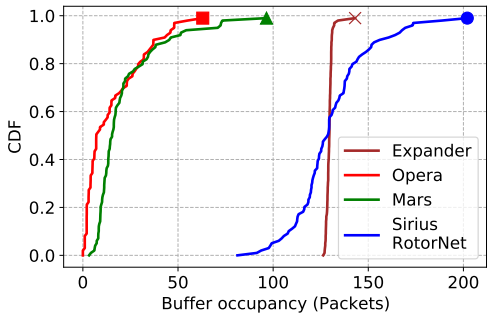


Fig. 10. MARS achieves high throughput but requires significantly less buffer compared to Expander, Sirius and RotorNet; and similar buffers as Opera.

for short flows by 93.88% compared to Opera, by 98.11% compared to expander and by 83.47% compared to Sirius and RotorNet.

We observe similar improvements under an all-to-all demand matrix as shown in Figure 9. At 5% (10%) load, MARS reduces the FCTs for short flows by 66.96% (90.15%) compared to Opera, by 81.35% (97.46%) compared to expander and by 87.6% (83.90%) compared to Sirius and RotorNet.

It is interesting to observe that the 99%-percentile flow completion times for short flows at 1% load follows the delay trend shown in Figure 1: a static expander has the least path delay and similarly Opera since it routes all the short flows over an expander; Sirius and RotorNet which emulate a complete graph experience the highest delays. However, as the load increases, multi-hop routing and queueing delays also impact the FCTs of short flows.

MARS does not trade long flow FCTs for short flows: MARS not only improves the FCTs for short flows but also achieves on-par 99-percentile FCTs for long flows compared to existing approaches. Under random permutation demand matrix (Figure 8), at 20% load, MARS reduces the 99-percentile FCTs for long flows by 2.37% compared to Opera, by 85.98% compared to expander and by 99.9% compared to Sirius and RotorNet. Further, under an all-to-all demand matrix (Figure 9), at 20% load, MARS reduces the FCTs for long flows by 63.42% compared to Opera, by 81.96% compared to expander and by 99.2% compared to Sirius and RotorNet. Interestingly, at 20% load, under random permutation (Figure 8d) and all-to-all (Figure 9d) demand matrices, Sirius and RotorNet cannot sustain long flow FCTs given the shallow buffers used in our setup. This result further strengthens our motivation on the performance of existing approaches under resource constraints.

Overall, our evaluation confirms that MARS outperforms existing approaches by improving throughput and reducing the flow completion times for short flows as well as long flows.

6 DISCUSSION

While our model and choice of topologies are in line with the assumptions and conclusions in the literature, they have certain practical implications which we discuss below.

Demand Matrix: Our model assumes that a demand matrix is fixed and does not change over time. In contrast, real-world demand matrices evolve. This would indeed complicate our analysis framework by introducing time variables to the demand. However, our analysis still provides insights within the duration in which the demand matrix remains constant. As demand matrices do not change rapidly over time [41], one could use our analysis to find the throughput between two time instances when the demand matrix changes significantly.

Congestion control and load balancing: Both congestion control and load balancing significantly impact the achievable throughput. Our main results in this paper rely on the theoretical definition of flow and throughput maximization problem. As a result, we make a simplifying assumption on the underlying system: congestion control, load balancing and routing have a central view of the entire network and perform ideally. Our theoretical results can be useful in understanding the ideal scenario and provides insights into the performance gap of a deployed system.

Worst-case analysis: Literature defines throughput of topologies based on worst-case demand matrices and corresponding maximum flow. On one hand, worst-case demand matrices help in understanding the performance bound of a topology under *any* demand matrix (within the scope of our definition). In other words, a topology optimized for the worst-case demand matrix achieves strictly greater throughput under any other demand matrix. On the other hand, the theoretical definition of "flow" relates to fluid transmissions with ideal congestion control, load balancing and routing (see above). In essence, our analysis captures the performance bounds of an ideal transport under worst-case demand. Our key insights from theory (summarized in §3.5) drove

the design of Mars which is optimized for the worst-case. However, our evaluation of Mars incorporates stochastic flow arrival process (not the worst-case) based on real-world flow size distributions [39, 40] at a given load. The significantly better performance of Mars (optimized for the worst-case) in our evaluation setup (not the worst-case) shows that our analysis indeed finds itself useful even under realistic settings.

Cost-equivalent Clos topologies: We emphasize that our results in Theorem 1, 2, 3, 4 and our evaluation concerns uni-regular topologies i.e., every switch is connected to servers and generates traffic. In contrast, traditional datacenters built on Clos topologies add additional layers of switches (additional capacity) that do not generate traffic in order to maximize throughput. This paper *does not* argue that uni-regular topologies are better than Clos topologies. However, most recent works [2–4, 8, 42] demonstrate the significant benefits of reconfigurable (uni-regular) topologies over Clos. Furthermore, comparing any other topologies outside the spectrum shown in Figure 1 requires a cost analysis which is volatile in nature abd . We hence focus strictly on the spectrum of topologies with periodic reconfigurations and their tradeoffs. It would be interesting to determine the buffer size at which the performance of a throughput-optimal periodic reconfigurable topology drops below a cost-equivalent Clos topology.

We leave it for future work to theoretically study the throughput of periodic reconfigurable topologies under dynamic demand matrices and more practical design considerations.

7 RELATED WORK

Indeed, only little is known today about how existing reconfigurable topologies fare against each other, and whether alternative designs could improve the throughput further. In the following, we discuss the related work in network throughput both in the context of traditional static datacenter topologies [1, 17–19, 43–45] and emerging reconfigurable datacenter topologies [2–11, 46–54].

Throughput as a metric: Bisection bandwidth has been used extensively as a metric in the networking community. A classic result in graph theory suggests that the max-flow can be an $O(\log(n))$ factor lower than the sparsest cut [26, 27]. The *throughput* maximization problem has been studied in the context of max-flow multicommodity flow and maximum concurrent flow problems [25, 55, 56]. A fully polynomial-time approximation scheme exists for arbitrary demands and uniform capacity [25]. Recently, Jyothi et al. [23] revisited the relation between cut-based metrics and max-flow in our networking context and proposed throughput under a worst-case demand matrix as metric. While it still remains an active area of research to efficiently compute throughput of a static topology [16, 23, 24], initial studies on dynamic and demand-aware topologies attempt to characterize throughput in terms of demand-completion times [13]. We are not aware of a formal definition and analysis of throughput in the context of periodic reconfigurable topologies.

Static topologies: Clos-based topologies [43, 44] have been shown to provide optimal throughput [16]. Bcube [57] proposes a server-centric architecture and provides high capacity for all-to-all traffic patterns but may not be optimal in terms of the throughput metric. JellyFish [18] argues that random graphs are highly flexible for datacenters in-terms of heterogeneous expansion and fault-tolerance while achieving high throughput. SlimFly [19] optimizes for diameter of the topology (consequently throughput) but imposes strict conditions on the size of switches. Xpander [17] focuses on incremental deployability while achieving high throughput. F10 [58] on fault-tolerance, and FatClique [45] on the cost, incremental expansion and management in a datacenter.

Reconfigurable topologies: In contrast to static topologies based on costly, power-intensive electrical packet switches, reconfigurable topologies rely on cost-effective technologies such as optical circuit switches and tunable lasers. Reconfigurable topologies can be broadly classified

into two types *(i)* demand-aware and *(ii)* demand-oblivious. Demand-aware topologies such as Duo [46], ReNets [59] and others [8–10, 60] adjust the topology based on the traffic patterns. However, such networks incur high "latency tax" due to the added complexity of measuring and calculating the demand via control-plane. Demand-oblivious topologies such as RotorNet [2–4] rely on a pre-defined schedule for circuit setup and have been shown to provide high-throughput with low "latency tax" due to reconfigurations. More recently in a parallel work, the relation between throughput and delay of periodic reconfigurable networks has been studied in detail [61]. Our main focus in this work has been on demand-oblivious designs under resource constraints. In particular, we reveal the fundamental tradeoffs across throughput, delay *and* buffer requirements for such designs. We further propose Mars, a throughput-optimal periodic (demand-oblivious) reconfigurable topology.

Reducing the buffer requirements: A vast literature in buffer management [28, 62, 63] and active queue management [64–67], scheduling [40, 68, 69] and end-host congestion control [38, 39, 70–73] focuses on reducing the queueing at a bottleneck link specifically in static datacenter topologies. However, as we rigorously discussed in this paper (§3, §4), periodic reconfigurable topologies fundamentally require certain amount of buffering and incur significant throughput loss under limited amount of buffer. Our approach exploits this tradeoff to systematically find the throughput-optimal design for any given delay and buffer constraints.

8 CONCLUSION

This paper was motivated by the observation that while dynamically reconfigurable datacenter networks can greatly improve throughput, existing RCDN designs which emulate complete graphs can entail high delays and buffer requirements. Based on our analysis of the underlying performance tradeoffs, we presented a more scalable network design, Mars, which achieves near-optimal throughput by emulating a d-regular graph and ensuring shallow buffers.

We understand our work as a first step and believe that it opens several interesting avenues for future research. In particular, naturally, the optimal RDCN topology will also depend on the price, and it will be interesting to conduct an economic study of the viability of different architectures. Furthermore, we have so far focused on flat networks; while such networks are common in the literature and have several advantages, it will be interesting to extend our study of buffer-aware RDCN designs to multi-tier networks as well.

ACKNOWLEDGEMENTS

We would like to thank our shepherd, Mohammad Hajiesmaili, as well as the anonymous reviewers and the technical program committee of SIGMETRICS 2023 for their useful feedback in shaping the final version of this paper. This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, consolidator project Self-Adjusting Networks (AdjustNet), grant agreement No. 864228, Horizon 2020, 2020-2025.

REFERENCES

[1] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the ACM SIGCOMM 2015 Conference*, page 183–197, 2015.

[2] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the ACM SIGCOMM 2020 Conference*, page 782–797, 2020.

[3] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the ACM SIGCOMM 2017 Conference*, page 267–280, 2017.

[4] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1–18, Santa Clara, CA, February 2020. USENIX Association.

[5] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, page 339–350, 2010.

[6] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the ACM SIGCOMM 2014 Conference*, page 319–330, 2014.

[7] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking*, 22(2):498–511, 2014.

[8] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the ACM SIGCOMM 2016 Conference*, page 216–229, 2016.

[9] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. C-through: Part-time optics in data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):327–338, aug 2010.

[10] Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking*, 24(3):1421–1433, 2016.

[11] Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, and Yueping Zhang. Proteus: A topology malleable data center network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, 2010.

[12] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. A survey of reconfigurable optical networks. *Optical Switching and Networking*, 41:100621, 2021.

[13] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. Cerberus: The power of choices in datacenter topology design - a throughput perspective. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(3), dec 2021.

[14] Wei Bai, Shuihai Hu, Kai Chen, Kun Tan, and Yongqiang Xiong. One more config is enough: Saving (dc)tcp for high-speed extremely shallow-buffered datacenters. *IEEE/ACM Transactions on Networking*, 29(2):489–502, 2021.

[15] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E Anderson. Backpressure flow control. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Renton, WA, April 2022. USENIX Association.

[16] Pooria Namyar, Sucha Supittayapornpong, Mingyang Zhang, Minlan Yu, and Ramesh Govindan. A throughput-centric view of the performance of datacenter topologies. In *Proceedings of the ACM SIGCOMM 2021 Conference*, page 349–369, 2021.

[17] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 205–219, 2016.

[18] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, San Jose, CA, April 2012. USENIX Association.

[19] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 348–359. IEEE, 2014.

[20] Stanley Cheung, Tiehui Su, Katsunari Okamoto, and SJB Yoo. Ultra-compact silicon photonic 512× 512 25 ghz arrayed waveguide grating router. *IEEE Journal of Selected Topics in Quantum Electronics*, 20(4):310–316, 2013.

[21] Larry A Coldren, Gregory Alan Fish, Y Akulova, JS Barton, L Johansson, and CW Coldren. Tunable semiconductor lasers: A tutorial. *Journal of Lightwave Technology*, 22(1):193, 2004.

[22] Broadcom. 2020. 25.6 tb/s strataxgs tomahawk 4 ethernet switch series.
[23] Sangeetha Abdu Jyothi, Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 761–772. IEEE, 2016.
[24] Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. High throughput data center topology design. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 29–41, Seattle, WA, April 2014. USENIX Association.
[25] Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.
[26] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. Technical report, Massachusetts Inst of Tech Cambridge Microsystems Research Center, 1989.
[27] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, nov 1999.
[28] Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. Abm: Active buffer management in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022.
[29] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit switching under the radar with REACToR. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 1–15, Seattle, WA, April 2014. USENIX Association.
[30] Leslie G Valiant and Gordon J Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, 1981.
[31] Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the lambertw function. *Advances in Computational mathematics*, 5(1):329–359, 1996.
[32] Béla Bollobás and W Fernandez de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2):125–134, 1982.
[33] Makoto Imase, Terunao Soneoka, and Keiji Okada. Connectivity of regular directed graphs with small diameters. *IEEE Transactions on Computers*, 34(03):267–273, 1985.
[34] Makoto Imase and Masaki Itoh. A design for directed graphs with minimum diameter. *IEEE Transactions on Computers*, 32(08):782–784, 1983.
[35] D. Z. Du and F. K. Hwang. Generalized de bruijn digraphs. *Networks*, 18(1):27–38, 1988.
[36] D. Naddef and W.R. Pulleyblank. Matchings in regular graphs. *Discrete Mathematics*, 34(3):283–291, 1981.
[37] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15(1):193–220, 1891.
[38] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the ACM SIGCOMM 2017 Conference*, page 29–42, 2017.
[39] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, page 63–74, 2010.
[40] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference*, page 435–446, 2013.
[41] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 66–85, 2022.
[42] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 281–294. ACM, 2017.
[43] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference*, page 63–74, 2008.
[44] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference*, page 51–62, 2009.
[45] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. Understanding lifecycle management complexity of datacenter topologies. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 235–254, Boston, MA, February 2019. USENIX Association.
[46] Johannes Zerwas, Csaba Gyorgyi, Andreas Blenk, Stefan Schmid, and Chen Avin. Duo: A high-throughput reconfigurable datacenter network using local routing and control. In *ACM SIGMETRICS*, 2023.

[47] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao, and Haitao Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454, aug 2012.

[48] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways to de-congest data center networks. In *HotNets*. ACM SIGCOMM, 2009.

[49] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling Wide-Spread communications on optical fabric with MegaSwitch. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 577–593, Boston, MA, March 2017. USENIX Association.

[50] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. Quartz: A new design element for low-latency dcns. *ACM SIGCOMM Computer Communication Review*, 44(4):283–294, aug 2014.

[51] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. *Queueing Systems*, 88(3):311–347, 2018.

[52] Roy Schwartz, Mohit Singh, and Sina Yazdanbod. Online and offline greedy algorithms for routing with switching costs. *CoRR*, abs/1905.02800, 2019.

[53] Michelle Hampson. Reconfigurable optical networks will move supercomputerdata 100x faster,". *IEEE Spectrum*, 2021.

[54] Fred Douglis, Seth Robertson, Eric van den Berg, Josephine Micallef, Marc Pucci, Alex Aiken, Keren Bergman, Maarten Hattink, and Mingoo Seok. Fleet—fast lanes for expedited execution at 10 terabits: Program overview. *IEEE Internet Computing*, 25(3):79–87, 2021.

[55] Yonatan Aumann and Yuval Rabani. An o(log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[56] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

[57] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference*, page 63–74, 2009.

[58] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A Fault-Tolerant engineered network. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 399–412, Lombard, IL, April 2013. USENIX Association.

[59] Chen Avin and Stefan Schmid. Renets: Statically-optimal demand-aware networks. In *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.

[60] Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2021.

[61] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. Optimal oblivious reconfigurable networks. *arXiv*, abs/2111.08780, 2021.

[62] A.K. Choudhury and E.L. Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions on Networking*, 6(2):130–140, 1998.

[63] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. In *ACM Workshop on Buffer Sizing*, 2019.

[64] Kathleen Nichols and Van Jacobson. Controlling queue delay: A modern aqm is just one piece of the solution to bufferbloat. *Queue*, 10(5):20–34, may 2012.

[65] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148–155, 2013.

[66] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[67] Cedric Westphal, Kiran Makhijani, and Richard Li. Packet trimming to reduce buffer sizes and improve round-trip times. In *ACM Workshop on Buffer Sizing*, 2019.

[68] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, aug 2012.

[69] Naveen Kr. Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable calendar queues for high-speed packet scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 685–699, Santa Clara, CA, February 2020. USENIX Association.

[70] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the ACM SIGCOMM 2015 Conference*, page 537–550, 2015.

[71] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the ACM SIGCOMM 2018 Conference*, page 221–235, 2018.

[72] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM SIGCOMM 2019 Conference*, page 44–58, 2019.

[73] Vamsi Addanki, Oliver Michel, and Stefan Schmid. PowerTCP: Pushing the performance limits of datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Renton, WA, April 2022. USENIX Association.

[74] Mark EJ Newman. Analysis of weighted networks. *Physical review E*, 70(5):056131, 2004.

APPENDICES

A PRELIMINARIES: THROUGHPUT OF STATIC TOPOLOGIES

Our goal in this section is to formally introduce throughput for static topologies and the relevant definitions, including the limitations of existing bounds. This section builds the intuition and motivation for our throughput analysis in the context of periodic reconfigurable topologies (§B). Much of the prior work [16, 23, 24] focused on static topologies and it remains unclear how the existing methodologies can be used to study the throughput problem in the context of reconfigurable topologies which is the focus of this paper. Initial studies on reconfigurable topologies [13] informally define throughput of specific existing systems (e.g., RotorNet) which emulate a complete graph. We emphasize that our definitions hold for *any* periodic reconfigurable topology.

The predominant throughput metrics for static topologies are defined via demand matrices (Definition 1, below), and in particular, the worst-case demand matrix. Following the definition of Jyothi et al. [23], we formally introduce throughput given a demand matrix for static topologies. In a nutshell, given a demand matrix $\mathcal{M}$, throughput is the highest scaling factor $\theta(\mathcal{M})$ such that the scaled demand matrix $\theta(\mathcal{M}) \cdot \mathcal{M}$ is feasible in the topology i.e., there exists a feasible flow that satisfies the demand. The throughput θ^* under a worst case demand matrix is the minimum $\theta(\mathcal{M})$ over the set of all saturated demand matrices i.e., $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$. Before formally defining throughput, we first define certain preliminaries which build the intuition for our definition of throughput in dynamic topologies (§B).

Consider any static topology (at switch level) represented as a static directed multigraph $G = (V, E)$ where V is the set of vertices (switches) and E is the set of **labelled** edges (links). Labelled edges allow for distinguishing parallel links. Let $c(e, \ell)$ denote the capacity of an edge $(e, \ell) \in E$ where ℓ is the corresponding label and e is directed edge of the form (u, v) connecting two vertex u to vertex v. We denote by $c(u)$, the sum of capacities of all outgoing (correspondingly incoming) edges of a node u. We assume that each node has equal incoming and outgoing total capacity. A demand matrix specifies the demand in bps (bits/second) between every pair of vertices.

Definition 1 (Demand matrix)**.** *Given a set of vertices V, a demand matrix specifies the demand rate between every pair of vertices in bps defined as $\mathcal{M} = \{m_{u,v} \mid u \in V, v \in V\}$ where $m_{u,v}$ is the demand between the pair u, v. A saturated demand matrix is such that the total demand originating at a source s equals its outgoing capacity and the total demand terminating at a destination d equals its incoming capacity i.e., $\sum_{u \in V} m_{s,u} = c(s)$ and $\sum_{u \in V} m_{u,d} = c(d)$.*

We consider *saturated* demand matrices since we focus on the maximum achievable throughput. Specifically, saturated demand matrices allow for studying the maximum demand that can be routed in a topology within the capacity constraints.

A.1 Paths and Flow in Static Graphs

Given a demand matrix $\mathcal{M}$, the graph G has a set of s-d paths for transmitting the demand $m_{s,d}$ from source s to destination d for all s-d pairs. We consider simple paths i.e., without cycles.

Definition 2 (Simple paths in static graphs (Standard))**.** *Given a static directed graph $G = (V, E)$ with set of vertices V and the set of all labelled edges E, a path p^* of length n connecting a source*

s and a destination d is a sequence of n labelled edges $\langle (e_1, \ell_1), (e_2, \ell_2) \ldots (e_n, \ell_n) \rangle$ *where* $(e_i, \ell_i) \in E$*; the corresponding sequence of vertices is* $\langle v_1, v_2 \ldots v_n, v_{n+1} \rangle$ *where* $v_i \in V$*,* v_1 *is the source s,* v_{n+1} *is the destination d and* e_i *is an edge between* v_i *and* v_{i+1} *i.e.,* $e_i = (v_i, v_{i+1})$ *for all* $i \in [1, n]$*. We consider* simple *paths where each vertex appears only once in the path, so there are no cycles.*

Similar to edge labels in a multigraph, we generalize the definition of paths to facilitate distinguishing between two paths with the same sequence of labelled edges (crucial for our analysis later). From here on, we will use an extended definition of paths in static graphs, formally defined below. We associate each path p with a unique identifier. As a result, notice that by associating each path with a unique identifier, the set of all extended paths P may contain more than one path with the same sequence of labelled edges and vertices. For example, (p^*, i) and (p^*, j) are treated as two different paths unless $i = j$ even though p^* from Definition 2 has the same sequence of labelled edges and vertices. We define paths with unique identifiers deliberately for ease of analysis later. Given a set of (extended) paths P and a path $p \in P$, we denote by $I(p)$, the set of all (extended) paths which have the identical sequence of labelled edges and vertices as that of p.

Definition 3 (Extended paths in static graphs)**.** *Given a static directed graph* $G = (V, E)$*, an extended path p is a pair* $p = (p^*, i)$*, where* p^* *is a standard path (Definition 2) and i is a unique identifier. An extended set of paths P is a set of extended paths. Given such P, the set of all s-d (extended) paths in P is denoted by* $P_{s,d}$*.*

Intuitively, the transmission of *s-d* demand from s to d in a graph is called an *s-d flow*. The set of all *s-d* flows is called a *flow*. We consider that *s-d* flow is splittable over multiple edges along all *s-d* paths. A legal (or feasible) flow in a static graph must be conserved along the path and must obey capacity constraints on every edge. Given a static graph and the set of *all* paths (Definition 2), there are infinite possibilities for an extended set of paths from Definition 3. Note that, flow is a map from a *finite* set of extended paths. We explicitly assume that for each path $p \in P$, and any $p' \in I(p)$, we have $F(p) = F(p')$.

Definition 4 (Flow in static graphs)**.** *Given static graph* $G = (V, E)$ *and a finite set of (extended) paths P, a Flow F is a map* $F : P \to \mathbb{R}^+$ *where* $\mathbb{R}^+$ *denotes non-negative real numbers. A* legal *flow further has the following constraints:* (i) *flow is conserved along the path and* (ii) *it obeys the capacity constraints. The amount of flow on a path* $p \in P$ *is denoted by* $F(p)$*, namely* $F(p)$ *is the amount of flow in each edge* $e \in p$*, denoted by* $F(p, e)$*. Formally,* $F(p, e) = F(p)$ *for all* $e \in p$*. Additionally, w.l.o.g, we assume that for each path* $p \in P$*, and any* $p' \in I(p)$*, we have* $F(p) = F(p')$*. The capacity constraint for a legal flow is given by the following inequality, where* $\mathbb{I}(\cdot)$ *is the indicator function.*

$$\sum_{p \in P} F(p) \cdot \mathbb{I}((e, \ell) \in p) \leq c(e, \ell) \qquad \forall e,\ \forall \ell \tag{5}$$

A.2 Throughput of Static Graphs

Following the definition of Jyothi et al. [23], throughput given a demand matrix $\mathcal{M}$ is the highest scaling factor θ such that the scaled demand matrix $\theta \cdot \mathcal{M}$ is feasible in the topology i.e., there exists a legal flow which can serve the scaled demand matrix $\theta \cdot \mathcal{M}$. The Throughput θ^* of a static topology is defined as the throughput under a worst-case demand matrix. Importantly, the following formal definition builds intuition for our context of periodic reconfigurable topologies.

Definition 5 (Throughput of static graphs)**.** *Given a demand matrix* $\mathcal{M}$ *and a graph G, a legal flow F has a throughput* $\theta(\mathcal{M}, F)$ *if it satisfies the scaled demand matrix* $\theta(\mathcal{M}, F) \cdot \mathcal{M}$*. Formally, for each s, d pair, the s-d flow sent from the source s to destination d is greater than the scaled demand*

$\theta(\mathcal{M}, F) \cdot m_{s,d}$.

$$\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V \tag{6}$$

The graph G has throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$ and the throughput θ^* under a worst-case demand matrix is the minimum $\theta(\mathcal{M})$ i.e., $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$.

Given a demand matrix $\mathcal{M}$, the throughput maximization problem has the objective to maximize θ, subject to Equation 5 (capacity constraint) and Equation 6 (demand constraint). Unfortunately, a linear program approach does not scale well to large topologies.

A.3 TUB and its Limitations

Recently, Namyar et al., proposed TUB [16], a scalable throughput upper bound for static topologies. The focus of this paper is not on the throughput of static topologies but rather on periodic reconfigurable topologies. However, even the most recently proposed throughput upper bound for static topologies has key limitations. Specifically, Namyar et al.propose the following throughput upper bound (Theorem 2.2 in [16]),

$$\theta^* \leq \min_{\mathcal{M}} \frac{\sum_e c(e)}{\sum_{s \in V} \sum_{d \in V} m_{s,d} \cdot L_{s,d}}$$

where $c(e)$ is the capacity of an edge $e \in E$ and $L_{s,d}$ is the *shortest path length* from s to d.

Key limitations of TUB: Consider a topology represented as a complete graph K_n with edge capacity c. Every node connects to every other node and hence the shortest path length for all pairs is one. In this case, TUB suggests that the topology has full throughput under a saturated permutation demand matrix, i.e., $\theta^* \leq 1$. However, the actual throughput is only $\frac{n}{2 \cdot n-1} \approx \frac{1}{2}$. This is since the saturated permutation demand matrix specifies a demand of $(n-1) \times c$ but the shortest path can only accommodate c demand. The remaining demand must be routed through multi hop paths resulting in lower throughput. Unfortunately, TUB does not capture this effect. In general we observe that TUB does not converge even for large scale topologies when the shortest path length significantly differs from the average route length. We show in our analysis that the throughput upper bound for any static graph is indeed related to the *average route length* (Theorem 2) which gives a much tighter bound $\theta^* \leq \frac{1}{2}$ in the above example.

B PRELIMINARIES: THROUGHPUT OF PERIODIC RDCNS

We now study the throughput of periodic reconfigurable topologies. Similar to static topologies, we first need to formally define paths and flow in the context of periodic reconfigurable topologies. In this section, we follow the periodic evolving graph model introduced in §2.2. Recall that the period of the periodic evolving graph is denoted by Γ, timeslot is denoted by Δ and the reconfiguration time is denoted by Δ_r. Since circuit switches do not route traffic during reconfiguration, links remain idle for Δ_r amount of time in every timeslot Δ. We denote this fraction of time spent in reconfiguration by Δ_u. We next formally define periodic evolving graph.

Definition 6 (Periodic Evolving Graph)**.** *A periodic evolving graph (henceforth for simplicity referred to as evolving graph) denoted by $\mathcal{G} = (V, \mathcal{E})$ is a periodic sequence of directed graphs with a period of Γ timeslots, defined for each timeslot $t \in \mathbb{W}$. The directed graph at time $t \in \mathbb{W}$ is defined as $\mathcal{G}_t = (V, \mathcal{E}_t)$, where V is the set of vertices and $\mathcal{E}_t \subseteq V \times V$ is the set of directed edges at time t and $\mathbb{W}$ is the set of whole numbers. Note that an edge $e \in V \times V$ may appear in multiple edge sets $\mathcal{E}_t$ at different times t. The evolving graph $\mathcal{G}$ has the following properties:*

- *The sequence of graphs starts at $t = 0$ and is defined for each timeslot in the interval $t \in [0, \infty)$.*
- *The edge set $\mathcal{E}_t$ at time t is periodic with period Γ i.e., $\mathcal{E}_{t+\Gamma} = \mathcal{E}_t$ and consequently the graph $\mathcal{G}_t$ is periodic with period Γ.*
- *An edge $e \in V \times V$ has a capacity $c_t(e)$ at time t. Since the graph is periodic, the capacity of an edge is also periodic: $c_{t+\Gamma}(e) = c_t(e)$. For notational convenience we set $c_t(e) = 0$ whenever $e \notin \mathcal{E}_t$.*

In this section, we first define the notions of path and flow in the periodic evolving graph. Using these definitions, we then formally define throughput.

B.1 Temporal Paths and Temporal Flow

Recall from Definition 6 that a periodic evolving graph $\mathcal{G}$ is a periodic sequence of directed graphs with period Γ. The graph at time t is denoted by $\mathcal{G}_t = (V, \mathcal{E}_t)$ where V is the set of vertices (switches) and $\mathcal{E}_t$ is the set of edges (links) at time t. Note that the edge set evolves over time. As a result, paths are formed over time as opposed to static paths. To this end, we define a temporal path in the evolving graph as a sequence of (e, t) edge-time pairs, where an edge e is accessed at time t. Consecutive edges along the path are accessed in non-decreasing order of time. We assume that the next edge along a path is accessed within Γ timeslots (the period). Further, temporal paths are periodic. We define the set of all temporal paths which start within the first period as the *foundation set* of temporal paths denoted by $\mathcal{P}^0$. The set of *all* temporal paths is denoted by $\mathcal{P}^*$. The foundation set is a crucial part of our analysis in §C, especially since temporal paths are periodic. We define temporal paths formally in the following.

Definition 7 (Temporal paths in evolving graphs)**.** *Given a periodic evolving graph $\mathcal{G}$ with set of vertices V, a temporal path δ of length n connecting a source s and a destination d is a sequence of n edges and corresponding time values $\langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$ where t_i is the time when the edge e_i is accessed along the path and e_i is an edge in $\mathcal{E}_{t_i}$; the corresponding sequence of vertices is $\langle v_1, v_2 \ldots v_n, v_{n+1} \rangle$ where v_1 is the source s, v_{n+1} is the destination d and e_i is an edge between v_i and v_{i+1} i.e., $e_i = (v_i, v_{i+1})$ for all $i \in [1, n]$. A* legal *temporal path additionally has the following properties:*

- *An edge e_{i+1} is accessed at time t_{i+1} within a period time (Γ) after accessing e_i at time t_i i.e., $t_i < t_{i+1} \leq t_i + \Gamma$*
- *$v_i \neq v_j$ if $i \neq j$ i.e., the temporal path is simple and has no cycles*

The foundation set *of temporal paths, denoted as $\mathcal{P}^0$ includes* all *legal temporal paths in $\mathcal{G}$ that start within the first period (i.e., $t_1 \in [0, \Gamma)$). For the* foundation set *$\mathcal{P}^0$, we define the set $\mathcal{P}^*$ of all* legal *temporal paths in $\mathcal{G}$ as follows:*

- *$\mathcal{P}^0 \subset \mathcal{P}^*$ and temporal paths are periodic with period Γ.*
- *A function $periodic(\delta)$ is defined as follows for every temporal path δ: the set of all temporal paths which have the same sequence of edges with the time sequence shifted by an integer multiple of Γ. Precisely, for any temporal path $\delta \in \mathcal{P}$, $periodic(\delta) = \{\langle (e_1, t_1 + k \cdot \Gamma), (e_2, t_2 + k \cdot \Gamma) \ldots (e_\ell, t_\ell + k \cdot \Gamma) \rangle \mid k \in \mathbb{W}\}$, where $\delta = \langle (e_1, t_1), (e_2, t_2) \ldots (e_\ell, t_\ell) \rangle$ and $\mathbb{W}$ is the set of whole numbers.*
- *$\mathcal{P}^* = \bigcup_{\delta \in \mathcal{P}^0} periodic(\delta)$ i.e., the set of all temporal paths $\mathcal{P}^*$ is the union of periodic temporal paths for each temporal path in the foundation set $\mathcal{P}^0$. For simplicity of notation, we say $\mathcal{P}^* = periodic(\mathcal{P}^0)$.*

The sets $\mathcal{P}^0$ and $\mathcal{P}^*$ are unique given $\mathcal{G}$. Let the set of all s-d temporal paths in $\mathcal{P}^0$ and $\mathcal{P}^*$ be denoted by $\mathcal{P}^0_{s,d}$ and $\mathcal{P}^*_{s,d}$, respectively.

Similar to paths and flow in static graphs (§A), in our context, the evolving graph and the set of s-d paths (Definition 7) in the evolving graph allow for transmitting the demand of all s-d pairs.

Intuitively, the transmission of *s-d* demand along *s-d* paths is called flow. In contrast to the flow in static graphs, the evolving graph $\mathcal{G}$ forces for the flow (Definition 4) to be additionally split in time since temporal paths are spread over time. Similar to the flow in static graphs, in the following we define a *Temporal-Flow* and its constraints in periodic evolving graphs. Specifically, a legal temporal flow must obey capacity constraints i.e., the sum of all flows on an edge at time t is upper bounded by the edge capacity $c_t(e)$.

Definition 8 (Temporal flow in evolving graphs)**.** *Given a periodic evolving graph $\mathcal{G}$ a legal set of temporal paths $\mathcal{P}$, a temporal flow $\mathcal{F}$ is a map $\mathcal{F} : \mathcal{P} \rightarrow \mathbb{R}^+$. The amount of flow on a path $\delta \in \mathcal{P}$ is denoted by $\mathcal{F}(\delta)$, namely $\mathcal{F}(\delta)$ is the amount of flow in each edge $(e,t) \in \delta$, denoted by $\mathcal{F}(\delta, e, t)$. Formally, $\mathcal{F}(\delta, e, t) = \mathcal{F}(\delta)$ for all $(e,t) \in \delta$ and otherwise $\mathcal{F}(\delta, e, t) = 0$. Temporal-Flow is periodic on periodic paths i.e., $\mathcal{F}(\delta) = \mathcal{F}(\delta')$ if $\delta' \in periodic(\delta)$. A temporal flow is said to be* legal *if it obeys the capacity constraints at any time t, formally*

$$\sum_{\delta \in \mathcal{P}} \mathcal{F}(\delta, e, t) \leq c_t(e) \qquad \forall e,\ \forall t \tag{7}$$

where $c_t(e)$ is the capacity of an edge e at time t.

B.2 Throughput of Evolving Graphs

Based on the definition of temporal-paths and temporal-flow, we are now ready to define throughput in evolving graphs. Intuitively, given a demand matrix $\mathcal{M}$, we define throughput as the highest scaling factor $\theta(\mathcal{M})$ such that $\theta(\mathcal{M}) \cdot \mathcal{M}$ amount of demand can be routed in the evolving graph *over one period* on average. Specifically, a temporal path δ sends $\mathcal{F}(\delta) \cdot (\Delta - \Delta_r)$ amount of demands (in bits) over $\Gamma \cdot \Delta$ time before its next periodic flow begins. Hence, $\frac{(\Delta - \Delta_r)}{\Gamma \cdot \Delta} \cdot \mathcal{F}(\delta) = (\frac{1 - \Delta_u}{\Gamma}) \cdot \mathcal{F}(\delta)$ is the average bits per second transmitted on a temporal path δ. Following this intuition, we define throughput in the evolving graph.

Definition 9 (Throughput in the evolving graph)**.** *Given a demand matrix $\mathcal{M}$ and an evolving graph $\mathcal{G}$, a legal temporal flow $\mathcal{F}$ has throughput $\theta(\mathcal{M}, \mathcal{F})$ if it satisfies the scaled demand matrix over Γ timeslots. For each s, d pair, the s-d temporal flow sent from the source s to destination d over Γ timeslots is greater than the scaled demand $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d}$. Formally, let $\mathcal{P}^0$ be the foundation set of $\mathcal{G}$ and $\mathcal{P}^0_{s,d}$ is the set of all s-d temporal flows starting in the first period.*

$$\left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(\delta) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

The graph $\mathcal{G}$ has throughput $\theta(\mathcal{M}) = \max_{\mathcal{F}} \theta(\mathcal{M}, \mathcal{F})$ and the throughput θ^* under a worst-case demand matrix is the minimum $\theta(\mathcal{M})$ i.e., $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$.

Throughput in the evolving graph can be calculated over *any* one period time-interval. For simplicity, we consider the first period and the foundation set of temporal paths $\mathcal{P}^0$ especially since the summation of *s-d* flows can be converted to the first period using the periodic property of temporal paths and flow i.e., for every $\delta \in \mathcal{P}^0_{s,d}$, $\mathcal{F}(\delta) = \mathcal{F}(\delta')$ for $\delta' \in periodic(\delta)$. However, note that it takes at most the duration of the longest temporal path, in order for Definition 9 to hold at the *destination*. In essence, Definition 9 for periodic evolving graphs, follows the reasoning behind throughput of static topologies i.e., the maximum scaling factor θ such that the scaled demand matrix is feasible to route continuously in the periodic topology.

We now begin to analyze the throughput maximization problem in periodic evolving graphs. Notice that the problem is much more complicated compared to static graphs. First, the set of all temporal paths is infinite. Second, even the foundation set of temporal paths is exponential in size and grows with larger Γ (period).

To this end, our technique involves a static graph which we call the *Emulated graph* corresponding to a periodic evolving graph. We then show that a periodic evolving graph and its corresponding *static* emulated graph have the same throughput.

B.3 Emulated Graph: An Equivalent Static Topology

Given a periodic evolving graph $\mathcal{G}$, we define the corresponding emulated graph $G(\mathcal{G})$ as a function of $\mathcal{G}$. Specifically, edge set of the emulated graph is obtained from the union of edge sets of the periodic evolving graph taken over any *one period* time interval and the edges are **labelled** with the corresponding time of the edge in the evolving graph. The edge capacities are such that the emulated graph has the same amount of average capacity between all pairs compared to the evolving graph, including the overhead of reconfiguration (Δ_r). In the following, we formally define the emulated graph.

Definition 10 (Emulated graph)**.** *The emulated graph $G(\mathcal{G})$ of periodic graph $\mathcal{G}$ is defined as a static directed multigraph $G = (V, E)$, where V is the set of vertices of $\mathcal{G}$ and $E = \{(e, \ell) \mid e \in \mathcal{E}_\ell,\ \ell \in [0, \Gamma)\}$ is the set of directed edges obtained from the union of edges of the evolving graph over the $[0, \Gamma)$ time interval and every edge (e, ℓ) is labelled with ℓ if $e \in \mathcal{E}_\ell$. An edge $(e, \ell) \in E$ has capacity $\hat{c}(e, \ell)$ where the relation between $\hat{c}(e, \ell)$ and the original capacity $c_\ell(e)$ in the evolving graph is given by,*

$$\hat{c}(e, \ell) = \left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot c_\ell(e) \qquad \forall e,\ \forall \ell \in [0, \Gamma) \tag{8}$$

We now precisely specify the extended set of paths (see Definition 3) in the emulated graph. Specifically, the extended set of paths with unique identifiers allow for a relation between a path in the emulated graph to a temporal path in the evolving graph. We define a function *static* which converts a temporal path to static extended path and a function temporal which converts a static extended path to a temporal path. In essence, the set of all extended paths in the emulated graph is $P^* = \text{static}(\mathcal{P}^0)$. The temporal function allows for backward conversion i.e., $\mathcal{P}^0 = \text{temporal}(P^*)$.

Definition 11 (Extended paths in Emulated graph)**.** *Given a periodic evolving graph $\mathcal{G}$ with the foundation set of paths $\mathcal{P}^0$, the corresponding emulated graph $G(\mathcal{G})$ has a set of extended paths P^* given by* static$(\mathcal{P}^0)$*. The* static *function is defined as follows: for every temporal path δ, where $\delta = \langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$, the extended static path $p =$ static(δ) is given by $p = (\langle (e_1, \ell_1), (e_2, \ell_2) \ldots (e_n, \ell_n) \rangle, \delta)$, where $\ell_i = t_i \bmod \Gamma$ and δ is the unique identifier of the extended static path; recall that δ is unique. The inverse of* static *function is defined as:* temporal$(p) =$ static$^{-1}(p) = \delta$*, for all $p \in P^*$, where δ is the unique identifier associated with the extended path p.*

$$P^* = \text{static}(\mathcal{P}^0) = \{\text{static}(\delta) \mid \delta \in \mathcal{P}^0\}$$
$$\mathcal{P}^0 = \text{temporal}(P^*) = \{\text{temporal}(p) \mid p \in P^*\} \tag{9}$$

Note that, given a periodic evolving graph $\mathcal{G}$, its foundation set of temporal paths $\mathcal{P}^0$ is unique and includes *all* temporal paths which start in the first period. Since the emulated graph is a function of the periodic evolving graph, the set of extended static paths (Definition 11) is also unique. The extended set of paths and the static, temporal functions are a crucial part of our throughput analysis in the next section.

C ANALYSIS: THROUGHPUT OF PERIODIC RDCNS

Before presenting our analysis, we first state our main result in this section. Consider a periodic periodic evolving graph (see Definition 6) $\mathcal{G} = (V, \mathcal{E})$ with edge capacities $c_t(e)$ for every edge e at time t. Consider a static graph which is a function of the evolving graph $\mathcal{G}$ obtained from Definition 10 represented as $G(\mathcal{G}) = (V, E)$. We prove that the throughput of a periodic evolving graph is equivalent to the corresponding emulated graph.

Theorem 1 (Relation to Emulated Graph)**.** *Given a demand matrix $\mathcal{M}$, any legal temporal flow $\mathcal{F}$ in the periodic evolving graph $\mathcal{G}$ with throughput $\theta(\mathcal{M}, \mathcal{F})$ can be converted to a legal flow F in the emulated graph $G(\mathcal{G})$ with the same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$ and vice versa. Consequently the periodic evolving graph and the emulated graph have the same throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$: the maximum scaling factor given a demand matrix $\mathcal{M}$; they further have the same throughput θ^* for a worst-case demand matrix, where $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$.*

A sketch of our approach is as follows: *(i)* we prove in Lemma 1 that if a legal flow in the emulated graph has throughput θ over set of extended paths $P^* = \text{static}(\mathcal{P}^0)$, then there exists a legal temporal flow in the periodic evolving graph with the same throughput; *(ii)* we prove in Lemma 2 that if a legal temporal flow achieves throughput θ in the evolving graph, then there exists a legal flow in the emulated graph with the same throughput over the set of extended paths $P^* = \text{static}(\mathcal{P}^0)$; *(iii)* finally, we prove in Lemma 3 that a flow in emulated graph over the set of *all paths* P has the same throughput upper bound as that of a flow over the set of extended paths $P^* = \text{static}(\mathcal{P}^0)$. Using the above three results, we prove our main claim.

Lemma 1. *Let $\mathcal{M}$ be a demand matrix, $\mathcal{G}$ be a periodic evolving graph, $G(\mathcal{G})$ be its emulated graph and $P^* = \text{static}(\mathcal{P}^0)$. If $F : P^* \to \mathbb{R}^+$ is a legal flow in $G(\mathcal{G})$ with throughput $\theta(\mathcal{M}, F)$ then there exists a legal temporal flow $\mathcal{F}$ in the evolving graph $\mathcal{G}$ with the same throughput $\theta(\mathcal{M}, \mathcal{F}) = \theta(\mathcal{M}, F)$. The temporal flow $\mathcal{F}$ in the evolving graph can be constructed as follows:*

$$\mathcal{F}(\delta) = \left(\frac{\Gamma}{1-\Delta_u}\right) \cdot F(p) \qquad \forall p \in P^*;\ \delta \in \textit{periodic}(\text{temporal}(p)) \tag{10}$$

where $\mathcal{F}$ is periodic; for any temporal path $\delta \in \text{temporal}(P^*)$, $\mathcal{F}(\delta, e, t) = \mathcal{F}(\delta)$ for all $(e, t) \in \delta$ and $\mathcal{F}(\delta, e, t) = 0$ if $(e, t) \notin \delta$. $\mathcal{F}$ is then a map $\mathcal{F} : \mathcal{P}^* \to \mathbb{R}^+$, where $\mathcal{P}^* = \textit{periodic}(\text{temporal}(P^*))$.

Proof. From Definition 4, since F is a legal flow in the emulated graph, F obeys capacity constraints.

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e, \ell) \in p) \le \hat{c}(e, \ell) \qquad \forall e,\ \forall \ell \in [0, \Gamma)$$

We first substitute $\hat{c}(e, \ell) = \frac{(1-\Delta_u)}{\Gamma} \cdot c_\ell(e)$ and change ℓ to ℓ **mod** Γ without changing the value of both sides of the above inequality.

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e, \ell \bmod \Gamma) \in p) \le \left(\frac{1-\Delta_u}{\Gamma}\right) \cdot c_{(\ell \bmod \Gamma)}(e) \qquad \forall e,\ \forall \ell \in [0, \Gamma)$$

We now expand the above inequality as follows for all $\ell \in [0, \infty)$, without changing the value on both sides of the inequality. Recall that $c_\ell(e) = c_{\ell + k \cdot \Gamma}$ for any integer $k \ge 0$ ($c_\ell(e)$ is periodic).

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e, \ell \bmod \Gamma) \in p) \le \left(\frac{1-\Delta_u}{\Gamma}\right) \cdot c_\ell(e) \qquad \forall e,\ \forall \ell \in [0, \infty)$$

The above inequality holds since $c_\ell(e)$ is periodic and ℓ **mod** Γ always ranges between $[0, \Gamma)$. Substituting the temporal flow $\mathcal{F}$ for the static flow F, using Equation 10,

$$\sum_{p \in P^*} \mathcal{F}(\text{temporal}(p)) \cdot \mathbb{I}((e, \ell \bmod \Gamma) \in p) \le c_\ell(e) \qquad \forall e,\ \forall \ell \in [0, \infty)$$

From Definition 11 and from the periodic property of temporal paths (Definition 7), for every path $p \in P^*$ and for every $(e, \ell \bmod \Gamma) \in p$, there exists exactly one path $\delta \in periodic(\text{temporal}(p))$ such that $(e, \ell) \in \delta$. Using this relation, we convert the above inequality as follows,

$$\sum_{\delta \in periodic(\text{temporal}(P^*))} \mathcal{F}(\delta) \cdot \mathbb{I}((e,t) \in \delta) \leq c_t(e) \qquad \forall e,\ \forall t \in [0, \infty)$$

Since $periodic(\text{temporal}(P^*)) = \mathcal{P}^*$, and since $c_t(e) = 0$ if $e \notin \mathcal{E}_t$, we obtain the following relation.

$$\sum_{\delta \in \mathcal{P}^*} \mathcal{F}(\delta, e, t) \leq \begin{cases} c_t(e) & e \in \mathcal{E}_t \\ 0 & e \notin \mathcal{E}_t \end{cases}$$

From Definition 8, we conclude that $\mathcal{F}$ (from Equation 10) obeys capacity constraints. Further since temporal flow is constant and equal for all $(e,t) \in \delta$ for all temporal paths, it obeys flow conservation rules. It remains to prove that $\mathcal{F}$ also achieves throughput θ. Since F has a throughput $\theta(\mathcal{M})$, from Definition 5 we have that,

$$\sum_{p \in P^*_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

using the relation between F and $\mathcal{F}$ from Equation 10, we obtain the following inequality. From Definition 9, we conclude that $\mathcal{F}$ also achieves throughput $\theta(\mathcal{M}, \mathcal{F}) = \theta(\mathcal{M}, F)$.

$$\left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

□

From Lemma 1, we have that, given a demand matrix $\mathcal{M}$, if the static emulated graph has throughput $\theta(\mathcal{M}, F)$ for a specific flow F, then the scaled demand matrix $\theta(\mathcal{M}, F) \cdot \mathcal{M}$ is feasible in the periodic evolving graph. In the following, we state the reverse i.e., given a demand matrix $\mathcal{M}$, if the periodic evolving graph has throughput $\theta(\mathcal{M}, \mathcal{F})$ for a specific temporal flow $\mathcal{F}$, then the scaled demand $\theta(\mathcal{M}, \mathcal{F}) \cdot \mathcal{M}$ is feasible in the static emulated graph.

Lemma 2. *Given a demand matrix $\mathcal{M}$, if $\mathcal{F} : \mathcal{P}^* \to \mathbb{R}^+$ is a legal temporal flow in the evolving graph $\mathcal{G}$ with throughput $\theta(\mathcal{M}, \mathcal{F})$ then there exists a legal flow F in the emulated graph $G(\mathcal{G})$ with same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$. F is obtained as follows:*

$$F(\text{static}(\delta)) = \left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot \mathcal{F}(\delta) \quad \forall \delta \in \mathcal{P}^0 \tag{11}$$

F is then a map $F : P^* \to \mathbb{R}^+$, where $P^* = \text{static}(\mathcal{P}^0)$.

Proof. From Definition 4, for F to be a legal flow in the emulated graph G, F must obey capacity and flow conservation constraints. Further in order to achieve throughput $\theta(\mathcal{M}, F)$, we require that $F \geq \theta(\mathcal{M}, F) \cdot \mathcal{M}$. Since $\mathcal{F}$ is a legal flow in the evolving graph, $\mathcal{F}$ obeys capacity constraints. From Definition 8, for all $e \in \mathcal{E}_t$ at any time t we have that,

$$\sum_{\delta \in \mathcal{P}^*} \mathcal{F}(\delta) \cdot \mathbb{I}((e,t) \in \delta) \leq c_t(e) \qquad \forall e,\ \forall t \in [0, \infty)$$

The remainder of the proof follows similar logic as that of the proof of Lemma 1. Since for any path $\delta \in \mathcal{P}^*$, $\mathcal{P}^* = periodic(\mathcal{P}^0) = periodic(\text{temporal}(P^*))$, if $(e,t) \in \delta$, then there exists no other path $\delta' \in periodic(\delta)$ where $(e,t) \in \delta'$, except for the path δ itself. For every such path δ, there exists exactly one path $p \in P^*$ where $(e, \ell) \in p$ and $\ell = t \bmod \Gamma$. Using the relation between $\mathcal{F}$ and

F from Equation 11 and substituting the capacity relation $\hat{c}(e,\ell) = \frac{(1-\Delta_u)}{\Gamma} \cdot c_\ell(e)$, we obtain the following:

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e,\ell) \in p) \leq \hat{c}(e,\ell) \qquad \forall e,\ \forall \ell \in [0,\Gamma)$$

From Definition 4, the above inequality implies that the flow F obeys capacity constraints. Further, flow is conserved since $F(p)$ is constant and equal for all $e \in p$ for all paths. It remains to prove that F achieves throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$.
Since $\mathcal{F}$ has a throughput $\theta(\mathcal{M}, \mathcal{F})$ for a demand matrix $\mathcal{M}$, from Definition 9 we have that,

$$\left(\frac{1-\Delta_u}{\Gamma}\right) \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(\delta) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

We now substitute static variables using Equation 11 and obtain the throughput relation,

$$\sum_{p \in \text{static}(\mathcal{P}^0_{s,d})} F(p) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

Using the above inequality and since $static(\mathcal{P}^0_{s,d}) = P^*_{s,d}$, from Definition 5, we conclude that F also achieves throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$. □

From Lemma 1 and Lemma 2, given a demand matrix $\mathcal{M}$, the periodic evolving graph and the emulated graph with extended set of paths $P^* = \text{static}(\mathcal{P}^0)$, have the same throughput. It remains to show that the emulated graph with set of extended paths $P^* = \text{static}(\mathcal{P}^0)$ has the same throughput compared to the emulated graph with set of all possible paths (not extended).

Lemma 3. *Let $\mathcal{M}$ be a demand matrix, $\mathcal{G}$ be a periodic evolving graph, $G(\mathcal{G})$ be its emulated graph and $P^* = \text{static}(\mathcal{P}^0)$. If $F : P \to \mathbb{R}^+$ is a legal flow in $G(\mathcal{G})$ with throughput $\theta(\mathcal{M}, F)$ where P is the set of all simple paths then there exists a legal flow $F' : P^* \to \mathbb{R}^+$ in the emulated graph $\mathcal{G}$ with the same throughput $\theta(\mathcal{M}, F') = \theta(\mathcal{M}, F)$ where F' is as follows:*

$$F'(p') = \frac{F(p)}{|I(p)|} \quad \forall p' \in I(p) \tag{12}$$

where $I(p)$ is the set of all extended paths which have the same sequence of labelled edges as that of a path $p \in P$.

Proof. We have that $F : P \to \mathbb{R}^+$ is a legal flow in the emulated graph and has a throughput $\theta(\mathcal{M}, F)$. The proof follows by showing that $F' : P^* \to \mathbb{R}^+$ also obeys capacity constraints and has throughput $\theta(\mathcal{M}, F') = \theta(\mathcal{M}, F)$. We will first prove that the following capacity constraint (from Definition 4) for F' holds. We prove this using the relation between F and F' from Equation 12 and the fact that F obeys capacity constraints. Since F obeys capacity constraints, the following inequality holds.

$$\sum_{p \in P} F(p) \cdot \mathbb{I}((e,\ell) \in p) \leq c(e) \qquad \forall e,\ \forall \ell$$

For every path $p \in P$, the set of extended paths P^* consists of a set $I(p)$ of extended paths. Note that $I(p)$ is strictly greater than zero i.e., $I(p) > 0$. The argument is that *(i)* P is the set of all paths; *(ii)* P^* is obtained from the foundation set of all temporal paths $\mathcal{P}^0$ as $P^* = static(\mathcal{P}^0)$; *(iii)* for any path $p \in P$ with the sequence of edges $\langle e_1, e_2, ...e_n \rangle$, there exists at least one legal temporal path $\delta \in \mathcal{P}^0$ where $\delta = \langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$ such that t_1 is the time when the edge e_1 appears for the first time in the first period, t_2 is the time when the edge e_2 appears for the first time after t_1 and so on; each edge appears within Γ timeslots due to the periodicity of the edge set and such a temporal path δ belongs to $\mathcal{P}^0$ by definition. Further, for any path $p \in P$ and a labelled edge $(e,\ell) \in p$, by

definition, the labelled edge (e, ℓ) also belongs to any path $p' \in I(p)$. Using this relation, we expand the summation in the above inequality as follows:

$$\sum_{p \in P} \sum_{p' \in I(p)} \frac{F(p)}{|I(p)|} \cdot \mathbb{I}((e,\ell) \in p') \leq c(e) \qquad \forall e,\ \forall \ell$$

Substituting F' using Equation 12, we obtain the following relation.

$$\sum_{p \in P^*} F'(p') \cdot \mathbb{I}((e,\ell) \in p') \leq c(e) \qquad \forall e,\ \forall \ell$$

From Definition 4, the above inequality suggests that F' obeys capacity constraints. Similarly, it is easy to show that $\sum_{p' \in P^*_{s,d}} F'(p') \geq \theta(\mathcal{M}, F) \cdot m_{s,d}$ using the fact that F has a throughput $\theta(\mathcal{M})$ i.e., $\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d}$, for all s, d source-destination pairs. This concludes that the flow F' has throughput $\theta(\mathcal{M}, F') = \theta(\mathcal{M}, F)$. □

Using Lemma 1, Lemma 2 and Lemma 3, it is now straight-forward that our claim in Theorem 1 holds.

Proof of Theorem 1. From Lemma 1 and Lemma 2, for any feasible temporal flow $\mathcal{F}$ in the evolving graph with throughput $\theta(\mathcal{M}, \mathcal{F})$, there exists a feasible flow F in the emulated graph with the same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$ and vice versa. From Lemma 3, we have that, our definition of extended paths and its use in our analysis does not impact the throughput of emulated graph i.e., throughput with our definition of extended set of paths is same as the throughput with the standard definition of paths. This concludes that the evolving graph and the corresponding emulated graph have the same throughput $\theta(\mathcal{M})$: the maximum scaling factor given a demand matrix and the same throughput θ^* under a worst-case demand matrix. □

Corollary 1 (Reduction to simple graph)**.** *Throughput of a periodic evolving graph $\mathcal{G} = (V, \mathcal{E})$ is equivalent to the simple static graph $G(\mathcal{G}) = (V, E)$, where V is the set of vertices (same as the evolving graph) and $E = \bigcup_{t \in [0,\Gamma)} \mathcal{E}_t$ is the union of edges over one period of time (without any labels). The capacity of an edge $e \in E$ is given by,*

$$\hat{c}(e) = \left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot \sum_{t \in [0,\Gamma)} c_t(e)$$

where $c_t(e)$ is the capacity of an edge $e \in \mathcal{E}_t$ at time t in the evolving graph.

Notice that the simple static graph in Corollary 1 is a weighted simple graph corresponding to the emulated graph (Definition 10) with the same amount of capacity between any $u, v \in V$ for every $(u, v) \in E$. It is standard in the literature that a multigraph and the corresponding weighted graph have the same max-flow [74]. This allows us to use the simple static graph in order to obtain the throughput of a periodic evolving graph.

D REVISITING THROUGHPUT OF STATIC TOPOLOGIES

Definition 12 (Average route length)**.** *Given a demand matrix $\mathcal{M}$, a static graph $G = (V, E)$ and a flow F that satisfies the demand, the average route length denoted by $\mathrm{ARL}(\mathcal{M}, F)$ is defined by,*

$$\mathrm{ARL}(\mathcal{M}, F) = \sum_{s,d \in V} \sum_{p \in P_{s,d}} \frac{m_{s,d}}{M} \cdot r_p \cdot \mathrm{len}(p)$$

where $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$; r_p is the fraction of s-d demand transmitted on the path p and $\mathrm{len}(p)$ is the length of the path p.

Theorem 2 (Throughput). *Given a demand matrix $\mathcal{M}$ and a flow F, the throughput $\theta(\mathcal{M}, F)$ of a static graph represented as $G = (V, E)$ is given by Equation 1. The graph G has throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$: the maximum scaling factor given a demand matrix $\mathcal{M}$; and has throughput $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$ for a worst-case demand matrix.*

$$\theta(\mathcal{M}, F) \leq \frac{\hat{C}}{M \cdot \text{ARL}(\mathcal{M}, F)} \tag{1}$$

where $\hat{C} = \sum_{e \in E} \hat{c}(e)$ is the total capacity of the network, $M = \sum_{s,d \in V} m_{s,d}$ is the total demand for the network and $\text{ARL}(\mathcal{M}, F) = \sum_{s,d \in V} \sum_{p \in P_{s,d}} \frac{m_{s,d}}{M} \cdot r_p \cdot \text{len}(p)$ *is the* average route length *for $\mathcal{M}$ and F, where r_p is the fraction of demand transmitted on the path p.*

Proof. We restrict our analysis to saturated demand matrices. Let the static topology under consideration be a graph $G = (V', E)$, where V' is the set of all vertices and E is the set of all edges. A set $V \subseteq V'$ denotes the set of all vertices generating traffic (demand). Specifically, given any set of vertices V, a demand matrix $\mathcal{M}$ from Definition 1 has elements $m_{u,v}$ such that $\sum_{v \in V} m_{u,v} = c(u)$ for all $u \in V$ where $c(u)$ is the total *physical* bandwidth of the vertex u. We denote the capacity of an edge e by $\hat{c}(e)$.

Given a graph $G = (V', E)$, the set of vertices generating traffic (demand) $V \subseteq V'$, a demand matrix $\mathcal{M}$ for V, and a flow F, the relation between the throughput as a function of demand matrix $\mathcal{M}$ is given by Definition 5. From Definition 4, for F to be legal we have that,

$$\sum_{p \in P} F(p) \cdot \mathbb{I}(e \in p) \leq \hat{c}(e) \quad \forall e$$

summing over all the edges $e \in E$, we obtain the following:

$$\sum_{e \in E} \sum_{p \in P} F(p) \cdot \mathbb{I}(e \in p) \leq \sum_{e \in E} \hat{c}(e) \tag{13}$$

From Definition 5, a flow F achieves throughput $\theta(\mathcal{M}, F)$ if it obeys the following constraint:

$$\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \quad \forall s \in V,\ d \in V$$

Let $r_p = \frac{F(p)}{\sum_{p \in P_{s,d}} F(p)}$ for all $p \in P_{s,d}$ for each s, d pair. We now expand the inequality in Equation 13 as follows:

$$\sum_{s \in V} \sum_{d \in V} \sum_{e \in E} \sum_{p \in P_{s,d}} F(p) \cdot \mathbb{I}(e \in p) \leq \sum_{e \in E} \hat{c}(e)$$

we convert the summation over all edges $e \in E$ to a summation over only edges of each path $e \in p$ and drop the identifier $\mathbb{I}$ without changing the value of the LHS in the above inequality:

$$\sum_{s \in V} \sum_{d \in V} \sum_{p \in P_{s,d}} \sum_{e \in p} F(p) \leq \sum_{e \in E} \hat{c}(e)$$

substituting $F(p) = r_p \cdot \sum_{p \in P_{s,d}} F(p)$ and since $\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d}$:

$$\sum_{s \in V} \sum_{d \in V} \sum_{p \in P_{s,d}} \sum_{e \in p} \theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p \leq \sum_{e \in E} \hat{c}(e)$$

Since $\theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p$ is constant for all edges of a given path, the summation over all edges $e \in p$ gives $\text{len}(p) \cdot \theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p$ where $\text{len}(p)$ denotes the number of edges in the path p or

simply the length of the path.

$$\sum_{s\in V}\sum_{d\in V}\sum_{p\in P_{s,d}} \text{len}(p)\cdot\theta(\mathcal{M},F)\cdot m_{s,d}\cdot r_p \le \sum_{e\in E}\hat{c}(e)$$

Finally from the inequality, we obtain the throughput upper bound given a demand matrix as follows,

$$\theta(\mathcal{M},F) \le \frac{\sum_{e\in E}\hat{c}(e)}{\sum_{s\in V}\sum_{d\in V} m_{s,d}\cdot\left(\sum_{p\in P_{s,d}}\text{len}(p)\cdot r_p\right)}$$

where $\sum_{p\in P_{s,d}}\text{len}(p)\cdot r_p$ is conceptually the average route length from s to d. Let $M = \sum_{s\in V}\sum_{d\in V} m_{s,d}$ and $\hat{C} = \sum_{e\in E}\hat{c}(e)$,

$$\theta(\mathcal{M},F) \le \frac{\hat{C}}{M\cdot \text{ARL}(\mathcal{M},F)} \tag{14}$$

where $\text{ARL}(\mathcal{M},F) = \sum_{s,d\in V}\sum_{p\in P_{s,d}}\frac{m_{s,d}}{M}\cdot r_p\cdot\text{len}(p)$ is the *average route length* for $\mathcal{M}$ and F. Given a demand matrix, the above inequality varies based on the flow F. In order to maximize throughput, we rewrite the above inequality as a maximum over possible flow F.

$$\theta(\mathcal{M}) = \max_F \frac{\hat{C}}{M\cdot\text{ARL}(\mathcal{M},F)} \tag{15}$$

The throughput under a worst-case demand matrix is then the minimum $\theta(\mathcal{M})$ over the set of all saturated demand matrices.

$$\theta^* = \min_{\mathcal{M}}\max_F \frac{\hat{C}}{M\cdot\text{ARL}(\mathcal{M},F)} \tag{16}$$

□

We note that providing a *lower bound* for the $\text{ARL}(\mathcal{M},F)$, even when M and/or F are not known would provide an upper bound for θ^*. One such example is to take the shortest paths in the topology, as was done in [16]. But this will not always lead to a tight bound (see Appendix A.3), for example in the complete graph K_n. While efficiently computing the throughput upper bound is an on-going research [16, 23, 24], Theorem 2 shows that computing throughput essentially boils down to computing average route lengths.

E ANALYSIS: DELAY & BUFFER REQUIREMENTS

In this section, we determine the amount of buffer required at each node in order to achieve a throughput $\theta(\mathcal{M})$ which reveals an interesting relation between throughput and buffer in periodic reconfigurable topologies. We first formally define the delay of a temporal path for a better understanding later on the flow that needs to be "stored" at intermediate nodes.

Definition 13 (Temporal path delay)**.** *Given a periodic evolving graph $\mathcal{G}$ and the set of all temporal paths $\mathcal{P}$, the delay of a temporal path $\delta\in\mathcal{P}$ denoted by $L(\delta)$ is time it takes when the first edge $e_1\in\delta$ is accessed until the time the flow reaches the destination on the last edge $e_n\in\delta$. Formally, for a path $\delta\in\mathcal{P}$, if $\langle(e_1,t_1),(e_2,t_2)\ldots(e_n,t_n)\rangle$ is the sequence of edge-time values, then $L(\delta) = (t_n - t_1 + 1)\cdot\Delta + (\Gamma-1)\cdot\Delta$ where Δ is the absolute time of each timeslot and Γ is the period of the evolving graph.*

Clearly, the delay is lower bounded by $\text{len}(\delta)\cdot\Delta + (\Gamma-1)\cdot\Delta$ since it takes Δ (timeslot) amount of time to send over each edge $e\in\delta$ and there is always an inherent delay of $(\Gamma-1)$ timeslots due to the periodic nature. Similarly, the delay is upper bounded by $\text{len}(\delta)\cdot\Gamma\cdot\Delta$ since the delay between accessing each edge along a path can be at most Γ timeslots.

We first, define *average route delay* similar to average route length (Definition 12, which plays a key role in the analysis of the maximum latency and the required buffer at each node.

Definition 14 (Average route delay)**.** *Given a demand matrix $\mathcal{M}$, a periodic evolving graph $\mathcal{G}$ and a flow $\mathcal{F}$ that satisfies the demand, the average route delay denoted by* $\text{ARD}(\mathcal{M}, \mathcal{F})$ *is defined by,*

$$\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \frac{m_{s,d}}{M} \cdot L_{s,d}$$

where $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$; $L_{s,d} = \sum_{\delta \in \mathcal{P}^0_{s,d}} r_\delta \cdot L(\delta)$ is the s-d delay; r_δ is the fraction of s-d demand transmitted on the temporal path δ and $L(\delta)$ is the delay of the temporal path δ.

We now state the lower bound for the maximum delay in a periodic evolving graph.

Theorem 3 (Delay)**.** *Given a demand matrix $\mathcal{M}$, a n_u-regular periodic evolving graph $\mathcal{G}$ emulating a d-regular graph with a period of Γ timeslots each of duration Δ and a flow $\mathcal{F}$ that achieves throughput $\theta(\mathcal{M}, \mathcal{F})$, the average route delay* $\text{ARD}(\mathcal{M}, \mathcal{F})$ *and the maximum delay L_{max} are given by*[10]*,*

$$\begin{aligned} L_{max} \geq \text{ARD}(\mathcal{M}, \mathcal{F}) &= \text{ARL}(\mathcal{M}, \mathcal{F}) \cdot \Gamma \cdot \Delta \\ &\geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta(\mathcal{M}, \mathcal{F})}\right) \end{aligned} \tag{2}$$

where $\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \sum_{\delta} \frac{m_{s,d}}{M} \cdot r_\delta \cdot L(\delta)$ *is the* average route delay *for $\mathcal{M}$ and $\mathcal{F}$, where r_δ is the fraction of demand transmitted on the legal temporal path δ, and $L(\delta)$ is the delay of the path δ, $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$ is the total demand. In particular, for a worst-case demand matrix, the maximum latency is bounded by*

$$L_{max} \geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta^*}\right) \tag{3}$$

Proof. Based on Definition 9, for every *s-d* source-destination pair, the source generates $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d}$ flow on average in bits per second in every period. From Definition 14, $L_{s,d}$ denote the average delay between a source s and destination d. Then the destination only receives its first data at $L_{s,d}$ time later, implying that the source has already generated $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot (L_{s,d})$ amount of data. From then on, the source transmits $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d}$ average flow in bits per second and the destination receives $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d}$ average flow in bits per second in every period. Due to conservation of flow, for every source-destination pair, the data generated until $L_{s,d}$ delay i.e., $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot (L_{s,d})$ amount of data, circulates (moves between source-destination) in the network in every period. As a result, the capacity consumed by each *s-d* flow is $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot \frac{L_{s,d}}{\Gamma \cdot \Delta}$. The total capacity utilized is then $\sum_{s \in V} \sum_{d \in V} \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot \frac{L_{s,d}}{\Gamma \cdot \Delta}$. However, the total utilized capacity can also be written as $\theta(\mathcal{M}, \mathcal{F}) \cdot M \cdot \text{ARL}(\mathcal{M}, \mathcal{F})$, where $\theta(\mathcal{M}, \mathcal{F}) \cdot M$ is the total bits per second generated by the sources and $\text{ARL}(\mathcal{M}, \mathcal{F})$ is the average route length. Equating the above two, we have that,

$$\sum_{s \in V} \sum_{d \in V} \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \cdot \frac{L_{s,d}}{\Gamma \cdot \Delta} = \theta(\mathcal{M}, \mathcal{F}) \cdot M \cdot \text{ARL}(\mathcal{M}, \mathcal{F})$$

$$\frac{1}{M} \sum_{s \in V} \sum_{d \in V} m_{s,d} \cdot L_{s,d} = \text{ARD}(\mathcal{M}, \mathcal{F}) = \text{ARL}(\mathcal{M}, \mathcal{F}) \cdot \Gamma \cdot \Delta$$

Since the maximum delay L_{max} is atleast the average delay,

$$L_{max} \geq \text{ARD}(\mathcal{M}, \mathcal{F}) = \text{ARL}(\mathcal{M}, \mathcal{F}) \cdot \Gamma \cdot \Delta$$

[10] Ω is the asymptotic lower bound notation.

The period Γ is at least $\frac{d}{n_u}$ timeslots since the node degree is limited to the number of uplinks n_u in each timeslot. Further, since the total capacity $\hat{C}$ equals the total demand M for saturated demand matrices; and using the relation between ARL and θ^* from Theorem 2, for a worst-case demand matrix we obtain the following:

$$L_{max} \geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta^*}\right)$$

□

Notice from Definition 7 and Definition 8 that it requires to "store" flow for a certain time and "forward" the flow when the next edge along a path is available. As a result, every node in the evolving graph requires certain amount of buffer space to store and forward flow on temporal paths in the evolving graph.

In the following, we extend our model to capture the required buffer space at each node.

Theorem 4 (Buffer)**.** *Given a demand matrix $\mathcal{M}$, a periodic evolving graph $\mathcal{G}$ requires at least $\hat{B}$ total amount of buffer in the network in order to achieve throughput $\theta(\mathcal{M}, \mathcal{F})$.*

$$\hat{B} \geq (\theta(\mathcal{M}, \mathcal{F}) \cdot M) \cdot \text{ARD}(\mathcal{M}, \mathcal{F}) \tag{4}$$

where $\hat{B} = \sum_{u \in V} B(u)$ is the total buffer of the network and $B(u)$ is the available buffer at a node u; $M = \sum_{s,d \in V} m_{s,d}$ is the total demand for the network; $\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \sum_{\delta} \frac{m_{s,d}}{M} \cdot r_\delta \cdot L(\delta)$ *is the* average route delay *for $\mathcal{M}$ and $\mathcal{F}$, where r_δ is the fraction of demand transmitted on the legal temporal path δ.*

PROOF. Let $B(u)$ be the amount of available (and used) buffer at a node $u \in V$. Then at any time $T \geq 0$, the difference between the total flow arrived and departed from node u (except the flow originating and terminating at u) is the amount of flow stored at u, formally expressed below. For simplicity, we set $\Delta_t = \Delta - \Delta_r$ and $\frac{\Delta_t}{\Delta} = 1 - \frac{\Delta_r}{\Delta} = 1 - \Delta_u$. We denote the set of incoming (outgoing) edges of a node u at any time t by $\mathcal{E}_t^-(u)$ ($\mathcal{E}_t^+(u)$).

$$B(u) \geq \sum_{t=0}^{T} \sum_{s,d \in V \setminus \{u\}} \sum_{\delta \in \mathcal{P}_{s,d}} \left(\sum_{e \in \mathcal{E}_t^-(u)} \mathcal{F}(\delta, e, t) - \sum_{e \in \mathcal{E}_t^+(u)} \mathcal{F}(\delta, e, t) \right) \cdot \Delta \tag{17}$$

For simplicity, we define the sum of all flows arriving and departing from a node u at any time t as follows:

$$R_t^-(u) = \sum_{s,d \in V} \sum_{\delta \in \mathcal{P}_{s,d}} \sum_{e \in \mathcal{E}_t^-(u)} \mathcal{F}(\delta, e, t)$$

$$R_t^+(u) = \sum_{s,d \in V} \sum_{\delta \in \mathcal{P}_{s,d}} \sum_{e \in \mathcal{E}_t^+(u)} \mathcal{F}(\delta, e, t)$$

Notice that at any time t, $\sum_{u \in V} R_t^-(u) = \sum_{u \in V} R_t^+(u)$. Since we take the summation over all nodes, for every outgoing flow on an edge, there is a corresponding equal flow incoming at the other end of the edge.

We now simplify Equation 17 using the above notation. Note that $\sum_{s \in V \setminus \{u\}} \sum_{\delta \in \mathcal{P}_{s,u}} \sum_{e \in E^-(u)} \mathcal{F}(\delta, e, t)$ is same as $\sum_{s \in V} \sum_{\delta \in \mathcal{P}_{s,u}} \sum_{e \in E^-(u)} \mathcal{F}(\delta, e, t)$ since $\mathcal{P}_{u,u}$ is

a null set.

$$B(u) \geq \sum_{t\in[0,T]} \left(R_t^-(u) - \sum_{s\in V} \sum_{\delta\in\mathcal{P}_{s,u}} \sum_{e\in E^-(u)} \mathcal{F}(\delta,e,t) \right) \cdot \Delta$$
$$- \sum_{t\in[0,T]} \left(R_t^+(u) - \sum_{d\in V} \sum_{\delta\in\mathcal{P}_{u,d}} \sum_{e\in E^+(u)} \mathcal{F}(\delta,e,t) \right) \cdot \Delta$$

Summing over all the nodes $u \in V$, we obtain the following,

$$\sum_{u\in V} B(u) \geq \overbrace{\sum_{u\in V} \left(\sum_{t\in[0,T]} \sum_{d\in V} \sum_{\delta\in\mathcal{P}_{u,d}} \sum_{e\in E^+(u)} \mathcal{F}(\delta,e,t) \right) \cdot \Delta}^{\text{Data sent out from each source } u\in V}$$
$$\underbrace{- \sum_{u\in V} \left(\sum_{t\in[0,T]} \sum_{s\in V} \sum_{\delta\in\mathcal{P}_{s,u}} \sum_{e\in E^-(u)} \mathcal{F}(\delta,e,t) \right) \cdot \Delta}_{\text{Data received at each destination } u\in V}$$

Let $L(\delta)$ denote the temporal path delay. $\mathcal{F}(\delta) \cdot \frac{L(\delta)\cdot(1-\Delta_u)}{\Gamma}$ is the amount of flow sent on a periodic path before the first data arrives at the destination i.e., $\mathcal{F}(\delta) \cdot \Delta \cdot (1-\Delta_u)$ is the data sent over Γ timeslots ($\Gamma \cdot \Delta$ time) and $\mathcal{F}(\delta) \cdot \frac{L(\delta)\cdot(1-\Delta_u)}{\Gamma}$ is the data sent over $L(\delta)$ time. For each s-d source-destination, the total data received at the destination d until time T is the total data sent from source s until time T minus the total data sent from source s before the destination d receives its first data.

$$\sum_{u\in V} B(u) \geq \sum_{u\in V} \left(T\cdot\Delta \cdot \sum_{d\in V} \theta(\mathcal{M},\mathcal{F}) \cdot m_{u,d} \right)$$
$$- \sum_{u\in V} \left(T\cdot\Delta \cdot \sum_{s\in V} \theta(\mathcal{M},\mathcal{F}) \cdot m_{s,u} \right)$$
$$+ \sum_{u\in V} \sum_{s\in V} \sum_{\delta\in\mathcal{P}^0_{s,u}} \frac{\mathcal{F}(\delta)\cdot L(\delta)\cdot(1-\Delta_u)}{\Gamma}$$

Finally, the above inequality reduces to,

$$\sum_{u\in V} B(u) \geq \sum_{s\in V} \sum_{d\in V} \sum_{\delta\in\mathcal{P}^0_{s,d}} \mathcal{F}(\delta) \cdot \frac{L(\delta)\cdot(1-\Delta_u)}{\Gamma}$$

We expand the above summation $\sum_{\delta\in\mathcal{P}^0_{s,d}} \mathcal{F}(\delta)$ by multiplying and dividing by $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d}$ where $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} = \sum_{\delta\in\mathcal{P}^0_{s,d}} \frac{\mathcal{F}(\delta)\cdot(1-\Delta_u)}{\Gamma}$ from Definition 9.

$$\sum_{u\in V} B(u) \geq$$
$$\sum_{s\in V} \sum_{d\in V} \left(\frac{\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} \cdot \Gamma \cdot \sum_{\delta\in\mathcal{P}^0_{s,d}} \mathcal{F}(\delta) \cdot \frac{L(\delta)\cdot(1-\Delta_u)}{\Gamma}}{\sum_{\delta\in\mathcal{P}^0_{s,d}} \mathcal{F}(\delta)\cdot(1-\Delta_u)} \right)$$

Let r_δ denote the fraction of temporal flow on a path $\delta \in P^0_{s,d}$ i.e., $r_\delta = \frac{\mathcal{F}(\delta)}{\sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(\delta)}$.

$$\sum_{u \in V} B(u) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot \sum_{s \in V} \sum_{d \in V} \left(m_{s,d} \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} r_\delta \cdot L(\delta) \right)$$

Let $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$ and the *average route delay* for $\mathcal{M}$ and $\mathcal{F}$ is denoted by $\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \sum_{\delta \in \mathcal{P}_{s,d}} \frac{m_{s,d}}{M} \cdot r_\delta \cdot L(\delta)$. The total buffer in the network is $\hat{B} = \sum_{u \in V} B(u)$. We obtain the following,

$$\hat{B} \geq (\theta(\mathcal{M}, \mathcal{F}) \cdot M) \cdot \text{ARD}(\mathcal{M}, \mathcal{F})$$

□

F ANALYSIS: MARS

We now restate and prove the throughput optimal degree d of MARS, given the delay requirement L.

Theorem 6 (Optimal degree d with delay constraints)**.** *The optimal d-regular graph emulated by MARS that maximizes throughput (given the delay requirement L and under Valiant load balancing) has a degree d given by,*

$$d = \lfloor e^{-\mathcal{W}(k)} \rfloor$$

where $k = \frac{-2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$; $\mathcal{W}$ is the Lambert W function [31]; $\ln(.)$ is the natural logarithm and e is the Euler's number.

PROOF. We begin by equating the delay of MARS and the desired delay L.

$$L = \frac{2 \cdot \log_d(n_t) \cdot d \cdot \Delta}{n_u}$$

Rearranging the terms in the above equation, we obtain the following,

$$\frac{\ln(d)}{d} = \frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$$

$$\frac{1}{d} \cdot \ln\left(\frac{1}{d}\right) = -\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$$

For simplicity, let $-\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$ be some constant k.

$$\frac{1}{d} \cdot \ln\left(\frac{1}{d}\right) = k$$

The above equation is of the form $y \cdot \ln(y) = k$ whose solution is $y = e^{\mathcal{W}(k)}$ where $\mathcal{W}$ is the lambert W function [31].

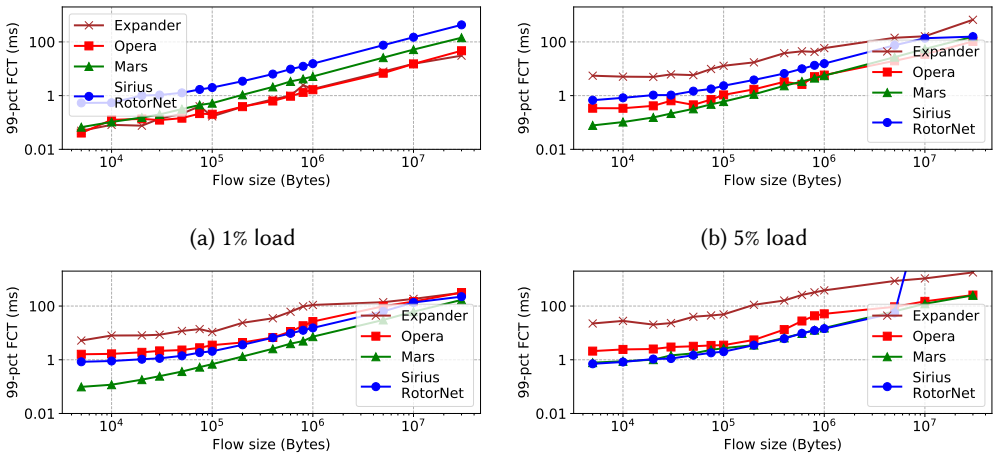
$$\frac{1}{d} = e^{\mathcal{W}(k)}$$

Substituting k in the above, we obtain the following.

$$d = e^{-\mathcal{W}(k)} = e^{-\mathcal{W}\left(-\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}\right)}$$

Here, if $k < -\frac{1}{e}$, there exists no real solution. However, we show that k is always greater than $-\frac{1}{e}$ i.e., $k > -\frac{1}{e}$

$$k = -\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L} \geq -\frac{2 \cdot \ln(n_t)}{n_u} \geq -2 \cdot \ln(n_t) > -\frac{1}{e}$$



(a) 1% load (b) 5% load

(c) 10% load (d) 20% load

Fig. 11. 99-percentile flow completion times for Datamining workload [40] under **random permutation** demand matrix.

The above inequalities hold since the latency L is at least one timeslot i.e., $L \geq \Delta$; the topology consists of at least two ToRs i.e., $n_t \geq 2$ and $\ln(2) > \frac{1}{e}$.

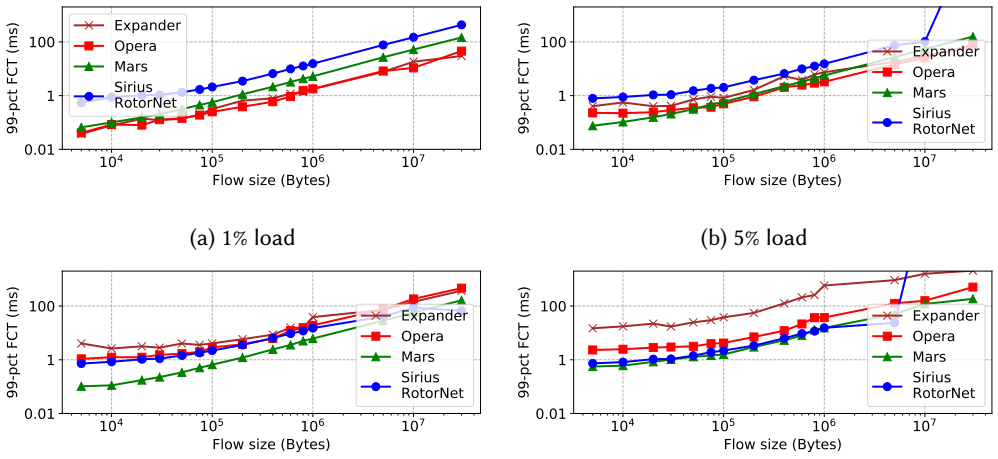
If $-\frac{1}{e} \leq k < 0$, there exists two real solutions for the degree d. In this case, we take the the highest value and round it to the nearest integer. Here, choosing the highest value d lowers the average route length $2 \cdot \log_d(n_t)$ and consequently maximizes throughput within the latency requirement. □

Theorem 7 (Optimal degree d with buffer constraints)**.** *The optimal degree d for the emulated graph of* Mars *independent of the specific flow that maximizes throughput, given a limited buffer $B \leq n_t \cdot \Delta$ at each node, is given by,*

$$d = \lfloor \frac{B}{c \cdot \Delta} \rfloor$$

where c is the capacity of every edge in the topology and Δ is the timeslot value.

Proof. From Theorem 4, the required buffer is $(\theta(\mathcal{M}, \mathcal{F}) \cdot M \cdot ARD(\mathcal{M}, \mathcal{F}))$. Further, from Theorem 3 we have that the delay is bounded by $\frac{d \cdot \Delta}{n_u \cdot \theta(\mathcal{M}, \mathcal{F})}$. Since $M = n_t \cdot n_u \cdot c$ (total demand and capacity), this gives us the buffer requirement of $n_t \cdot c \cdot d \cdot \Delta$ in the whole network. Given the regularity of the Mars, each node would then require $c \cdot d \cdot \Delta$ amount of buffer to hold the demand. Now, given the constraint that only B amount of buffer is available at each node, the optimal degree d is then $\frac{B}{c \cdot \Delta}$ rounded to the nearest integer. □



(a) 1% load (b) 5% load

(c) 10% load (d) 20% load

Fig. 12. 99-percentile flow completion times for Datamining workload [40] under **all-to-all** demand matrix.

Received August 2022; revised October 2022; accepted January 2023