

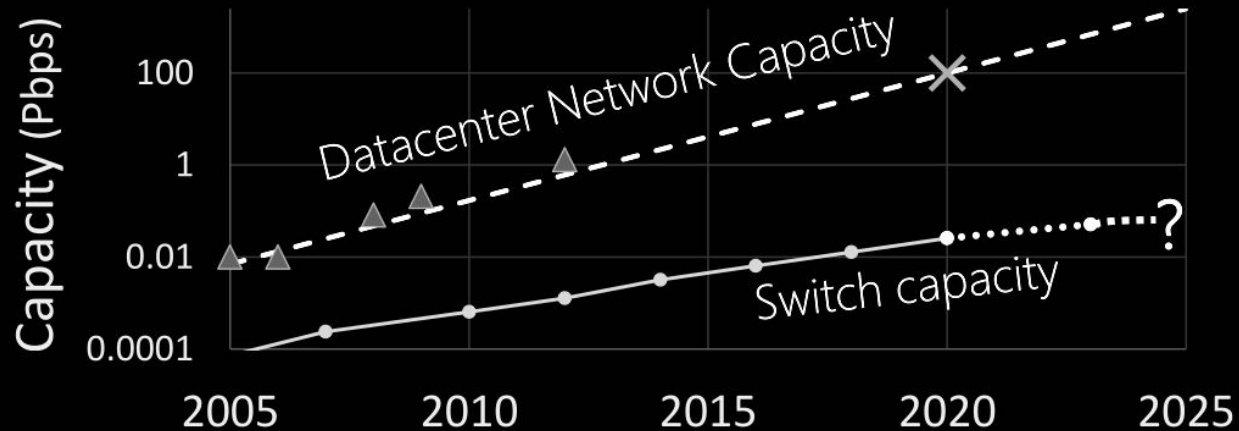
Throughput Bounds of Reconfigurable Networks

Insights & Reflections on Metrics for Collective Communication

Vamsi Addanki



Network Demand vs Capacity **Mismatch**



[1] A flat datacenter network with nanosecond optical switching (SIGCOMM 2020)

Traditional Approach to Data Center Topologies

A Scalable, Commodity Data Center Network Architecture

Mohammad Al-Fares
malfares@cs.ucsd.edu

Alexander Loukissas
aloukiss@cs.ucsd.edu

Amin Vahdat
vahdat@cs.ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

ABSTRACT

Today's data centers may contain tens of thousands of computers with significant aggregate bandwidth requirements. The network architecture typically consists of a tree of routing and switching elements with progressively more specialized and expensive equipment moving up the network hierarchy. Unfortunately, even when deploying the highest-end IP switch/router, resulting topologies may only support 50% of the aggregate bandwidth available at the edge of the network, while still incurring tremendous cost. Non-uniform bandwidth among data center nodes complicates application design and limits overall system performance.

In this paper, we show how to leverage largely commodity Ethernet switches to support the full aggregate bandwidth of clusters consisting of tens of thousands of elements. Similar to how clusters of commodity computers may largely replace more specialized SMPs and MPPs, we argue that appropriately architected and inter-connected commodity switches may deliver more performance at less cost than available from today's higher-end solutions. Our approach requires no modifications to the end host network interface, operating system, or applications; critically, it is fully backward compatible with Ethernet, IP, and TCP.

Categories and Subject Descriptors

C.2.1 [Network Architecture and Design]: Network topology; C.2.2 [Network Protocols]: Routing protocols

General Terms

Design, Performance, Management, Reliability

Keywords

Data center topology, equal-cost routing

1. INTRODUCTION

Growing expertise with clusters of commodity PCs has enabled a number of institutions to harness potentials of computation power and petabytes of storage in a cost-efficient manner. Clusters consisting of tens of thousands of PCs are not unheard of in the largest

institutions and thousand-node clusters are increasingly common in universities, research labs, and companies. Important applications classes include scientific computing, financial analysis, data analysis and warehousing, and large-scale network services.

Today, the principle bottleneck in large-scale clusters is often inter-node communication bandwidth. Many applications must exchange information with remote nodes to proceed with their local computation. For example, MapReduce [12] must perform significant data shuffling to transport the output of its map phase before proceeding with its reduce phase. Applications running on cluster-based systems [18, 28, 30] often require remote-node access before proceeding with their IO operations. A query to a web search engine often requires parallel communication with every node in the cluster hosting the inverted index to return the most relevant result [7]. Even between logically distinct clusters, there are often significant communication requirements, e.g., when updating the inverted index for individual clusters performing search from the site responsible for building the index. Internet services increasingly employ server oriented architectures [13], where the retrieval of a single web page can require coordination and communication with literally hundreds of individual sub-services running on remote nodes. Finally, the significant communication requirements of parallel scientific applications are well known [27, 6].

There are two high-level choices for building the communication fabric for large-scale clusters. One option leverages specialized hardware and communication protocols, such as InfiniBand [2] or Myrinet [6]. While these solutions can scale to clusters of thousands of nodes with high bandwidth, they do not leverage commodity parts (and are hence more expensive) and are not naturally compatible with TCP/IP applications. The second choice leverages commodity Ethernet switches and routers to interconnect cluster machines. This approach supports a familiar management paradigm using unmodified applications, operating systems, and hardware. Unfortunately, aggregate cluster bandwidths poorly with cluster size, and achieving the highest levels of bandwidth incur non-linear cost increases with cluster size.

For compatibility and cost reasons, most cluster communication systems follow the second approach. However, communication bandwidth in large clusters may become oversubscribed by a significant factor depending on the communication patterns. That is, two nodes connected to the same physical switch may be able to communicate at full bandwidth (e.g., 10Gps) but moving between switches, potentially across multiple levels in a hierarchy, may limit available bandwidth severely. Addressing these bottlenecks requires non-commodity solutions, e.g., large 10Gbps switches and routers. Further, typically single path routing along lines of interconnected switches means that overall cluster bandwidth is limited by the bandwidth available at the root of the communication hierarchy.

VL2: A Scalable and Flexible Data Center Network

Albert Greenberg
Srikanth Kandula
David A. Maltz

James R. Hamilton
Changhoon Kim
Parveen Patel

Naveendu Jain
Parantho Lahoti
Sudipta Sengupta

Microsoft Research

Abstract

To be agile and cost effective, data centers should allow dynamic resource allocation across large server pools. In particular, the data center network should enable any server to be assigned to any service. To meet these goals, we present VL2, a practical network architecture that scales to support large data centers with uniform high capacity between servers, performance isolation between services, and Ethernet layer semantics. VL2 uses (1) flat addressing to allow service instances to be placed anywhere in the network, (2) Valiant Load Balancing to spread traffic uniformly across network paths, and (3) end-system based address resolution to scale to large server pools without introducing complexity to the network control plane. VL2's design is driven by detailed measurements of traffic and fault data from a large operational cloud service provider. VL2's implementation leverages proven network technologies, already available at low cost, to high-speed hardware implementations, to build a scalable and reliable network architecture. As a result, VL2 networks can be deployed today, and we have built a working prototype. We evaluate the merits of the VL2 design using measurement, analysis, and experiments. Our VL2 prototype shuffles 1.7 TB of data among 75 servers in 395 seconds—sustaining a rate that is 94% of the maximum possible.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design

General Terms: Design, Performance, Reliability

Keywords: Data center network, communication

1. INTRODUCTION

Cloud services are driving the creation of data centers that hold tens to hundreds of thousands of servers that and concurrently support a large number of distinct services (e.g., search, email, map, video, computations, and utility computing). The motivations for building such shared data centers are both economic and technical: to leverage the economies of scale available in bulk deployment and to benefit from the ability to dynamically reallocate servers among services as workload changes (e.g., [1, 16]).

Also large—upwards of a million reallocate for a 1,000-server data center—with the servers themselves comprising the largest cost component. To be profitable, these data centers must achieve high utilization, and key to this is the property of agility—the capacity to assign any server to any service.

Agility promises improved risk management and cost savings. Without agility, each service must pre-allocate enough servers to meet difficult to predict demand spikes, or risk failure at the brink of success. With agility, the data center operator can meet the fluctuating demands of individual services from a large shared server pool, resulting in higher server utilization and lower costs.

Unfortunately, the designs for today's data center network prevent agility in several ways. First, existing architectures do not provide enough capacity between the servers they interconnect. Conventional architectures rely on the tree-like network configurations built from high-end hardware. Due to the cost of the equipment, the capacity between different branches of the tree is typically oversubscribed by factors of 1 or more, with paths through the highest level of the tree oversubscribed by factors of 16 to 1,024. This limits communication between servers that fragments the server pool—congestion and computation hot spots are prevalent even when spare capacity is available elsewhere. Second, while data centers host multiple services, the network does little to prevent a traffic flood in one service from affecting the other services around it—when one service experiences a traffic flood, it is common for all those sharing the same network sub-tree to suffer collateral damage. Third, the routing design in conventional networks achieves scale by assigning servers topologically significant IP addresses and dividing servers among VLANs. Such fragmentation of the address space limits the utility of virtual machines, which cannot migrate out of their original VLAN while keeping the same IP address. Further, the fragmentation of address space creates an enormous configuration burden when servers must be reorganized among services, and the human involvement typically required in these reconfigurations limits the speed of deployment.

To overcome these limitations in today's design and achieve agility, we arrange for the network to implement a familiar and simple design—give each server a unique IP address, assign it to it, and only those servers, are connected by a single non-interfering Ethernet switch—a Virtual Layer 2—and maintain the illusion even as the size of each service varies from a few servers to thousands. Realizing this vision concretely translates into building a network that meets the following three objectives:

- **Uniform high capacity:** The maximum rate of a server-to-server traffic flow should be limited only by the available capacity on the network interface cards of the sending and receiving servers, and the mapping servers to a service should be independent of network topology.
- **Performance isolation:** Traffic of one service should not be affected by the traffic of any other service, just as each service was controlled by a separate physical switch.
- **Layer 2 semantics:** Just as if all the servers were on a LAN—where any IP address can be connected to any port of an Ethernet switch by flat addressing—data center management software should be able to easily assign any server to any service and configure

Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network

Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armstrong, Roy Bannan, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Gorman, Anand Kanagala, Jeff Provoost, Jason Simmons, Elichi Tandra, Jim Wanderer, Urs Holzle, Stephen Stuart, and Amin Vahdat
Google Inc.
jupiter.sigcomm@google.com

ABSTRACT

We present our approach for overcoming the cost, operational complexity, and limited scale endemic to datacenter networks a decade ago. Three themes unify the five generations of datacenter networks detailed in this paper. First, multi-stage Clos topologies built from commodity switch silicon can support cost-effective deployment of building-scale networks. Second, much of the general, but complex, datacenter network routing and management protocols supporting arbitrary deployment scenarios were decentralized for single-operator, pre-planned datacenter networks. We built a centralized control mechanism based on a global configuration pushed to all datacenter switches. Third, modular hardware design coupled with simple, robust software allowed our design to also support inter-cluster and wide-area networks. Our datacenter networks run at dozens of sites across the planet, scaling in capacity by 100x over ten years to more than 17Tbps of inaction bandwidth.

CSCS Concepts

- **Networks** – Data center networks;

Keywords

Datacenter Networks; Clos topology; Merchant Silicon; Centralized control and management

1. INTRODUCTION

Datacenter networks are critical to delivering web services, modern storage infrastructure, and are a key en-

abler for cloud computing. Bandwidth demands in the datacenter are doubling every 12-15 months (Figure 1), even faster than the wide area Internet. A number of recent trends drive this growth. Dataset sizes are continuing to explode with more photo/video content, logs, and the proliferation of Internet-connected sensors. As a result, network-intensive data processing pipelines must operate over ever-larger datasets. Next, Web services can deliver higher quality results by accessing more data on the critical path of individual requests. Finally, constellations of co-located applications often share substantial data with one another in the same cluster; consider index generation, web search, and serving ads.

Ten years ago, we found the cost and operational complexity associated with traditional datacenter network architectures to be prohibitive. Maximum network scale was limited by the cost and capacity of the highest end switches available at any point in time [28]. These switches were engineering marvels, typically reconfigured from products targeting wide area deployments. WAN switches were differentiated with hardware support/download for a range of protocols (e.g., IP, multi-unicast) or by pushing the envelope of chip memory (e.g., Internet-scale routing tables, off chip DRAM for deep buffers, etc.). Network control and management protocols targeted autonomous individual switches rather than pre-configured and largely static datacenter fabrics. Most of these features were not useful for datacenters, increased cost, complexity, delayed time to market, and made network management more difficult.

Watercenter switches were also built as complex beasts targeting the highest levels of availability. In a WAN Internet deployment, using a single switch/router with applications running on multiple servers, because WAN links are so expensive, it makes sense to invest in high availability. However, more plentiful and cheaper datacenter bandwidth makes it prudent to trade cost for somewhat reduced interconnect capacity. Finally, switches operating in a multi-tenant WAN environment with arbitrary and hosts require support for many protocols to ensure interoperability. In single-operator data-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the ACM publisher.
SIGCOMM '09, August 17-21, 2009, London, United Kingdom
© 2009 Copyright held by the owner(s).
ACM ISBN 978-1-60558-343-5/09
DOI: <http://dx.doi.org/10.1145/278599.2787906>

Traditional Approach vs. Datacenter Topologies

fattreeSigcomm.pdf 61.5%

A Scalable, Commodity Data Center Network Architecture

Mohammad Al-Fares
maffares@cs.ucsd.edu

Alexander Loukissas
aloukiss@cs.ucsd.edu

Amin Vahdat
vahdat@cs.ucsd.edu

jellyfish.pdf 57.0%

Jellyfish: Networking Data Centers, Randomly

Ankit Singla*, Chi-Yao Hong[†], Lucian Popa[‡], P. Brighten Godfrey[§]
[†] University of Illinois at Urbana-Champaign
[‡] University of California, Berkeley

1 Introduction

Data centers today form the backbone of cloud operations. A well provisioned data center network is important to ensure that servers do not face bandwidth bottlenecks to utilization; to isolate services from each other; and to gain more freedom in placement and scaling of servers than having to tailor placement of workloads to where bandwidth is available [16]. As a result, a significant body of work has tackled the problem of building high network capacity interconnects [1, 10–13, 26, 28, 29].

One crucial problem that has been ignored in these designs is that of incremental expansion of the network, i.e., adding servers and network capacity incrementally to the data center. This may be motivated by growth of the user base, which requires more servers, or by the deployment of more bandwidth-intensive applications. Such expansion can be made feasible by either planned overprovisioning of space and power, or by an upgrade of the server base. The latter enables some of the old servers to be replaced by a larger number of more powerful and, at the same time, less power-consuming new servers.

Industry experience indicates that incremental expansion is an important problem. Consider the growth of Facebook's data center server population from ~30,000 in November 2009 to more than 60,000 by June 2010 [24]. While Facebook has added new data center facilities too, much of this growth is more incremental in existing facilities. Adding capacity on a day basis [23]. For instance, Facebook announced that it will double the size of its facility at Prineville, Oregon by early 2012 [9]. Industry experts have also identified incremental build-out as a useful strategy to reduce capex up-front [20].

Do current high-bandwidth data center network proposals allow incremental growth? Consider the fat-tree proposal [1] as an illustrative example. The entire structure is completely determined by the port-count of the switches available. This is limiting in at least two ways. First, it makes the design space very coarse: full-bisection bandwidth fat-trees can only be built as sizes 4, 16, 64, 256, and 1024 corresponding to the commonly structured topologies are not applicable, physical construction, and cabling layout. We discuss these chal-

lenges for counts of 32, 48, and 64. Second, even if (for example) 50-port switches were available, the smallest incremental upgrade from the 16-port switch fat-tree would be 4,000 servers. Moreover, this “incremental” growth would require replacing all the 48-port switches by 50-port switches. There is, of course, the possibility of making the fat-tree scalable by replacing a switch with one with larger port count – this necessarily requires either the servers in the vicinity to share capacity unevenly compared to the rest of the server pool. Thus, without compromise on its structure, bandwidth or cost, a fat tree is not amenable to incremental growth. Other topologies have similar problems: a hypercube [2] allows power-of-2 sizes, a de Bruijn-like construction [27] allows power-of-3 sizes, etc.

Since it seems that *structure* hinders incremental expansion, we propose the opposite: a random network interconnect. The approach, which we call **Jellyfish**, constructs a random graph topology at the switch layer. The inherent slowness nature of this design has the potential of being significantly more flexible than past designs. Additional components – racks of servers or switches to improve capacity – can be incorporated with a few random edge swaps. The design naturally supports *latency*, allowing the addition of *newer* network elements with higher port-counts as they become available, unlike past proposals which depend on certain regular port-counts [1, 11–13, 26, 28]. Jellyfish also allows construction of arbitrary-size networks, unlike past proposals discussed above which limit the network to very coarse design points dictated by their structure.

Are we sacrificing high bandwidth for incremental growth? No. To the contrary, we show that Jellyfish supports more servers at full bisection bandwidth than an equal-cost fat-tree [1] and with lower mean path length. Jellyfish also provides high connectivity and path diversity. In addition, as we discuss later, Jellyfish is resilient to link failures and misroutings during construction.

On the other hand, a random data center network presents several interesting and important challenges that must be addressed for it to be viable. Among these are a suitable routing mechanism (schemes depending on a structured topology are not applicable), physical construction, and cabling layout. We discuss these chal-

shortTermVariations_scatter.pdf 57.4%

VL2: A Scalable and Flexible Data Center Network

Albert Greenberg
Srikanth Kandula
David A. Maltz

James R. Hamilton
Changhoon Kim
Parveen Patel

Navendu Jain
Parantap Lahiri
Sudipta Sengupta

Microsoft Research

slimfly.pdf 57.4%

Slim Fly: A Cost Effective Low-Diameter Network Topology

Maciej Besta
FTW Zurich
maciej.besta@inf.ethz.ch

Torsten Hofer
FTW Zurich
hoer@inf.ethz.ch

Abstract—We introduce a high-performance cost-effective network topology called **Slim Fly** that approaches the theoretically optimal network diameter. **Slim Fly** is based on graphs that approximate the would-be 4,000 servers. Moreover, this “incremental” growth would require replacing all the 48-port switches by 50-port switches. There is, of course, the possibility of making the fat-tree scalable by replacing a switch with one with larger port count – this necessarily requires either the servers in the vicinity to share capacity unevenly compared to the rest of the server pool. Thus, without compromise on its structure, bandwidth or cost, a fat tree is not amenable to incremental growth. Other topologies have similar problems: a hypercube [2] allows power-of-2 sizes, a de Bruijn-like construction [27] allows power-of-3 sizes, etc.

1. INTRODUCTION
Interconnection networks play an important role in today's large-scale computing systems. The importance of the network grows with ever increasing per-node (multi-core) performance and memory bandwidth. Large networks with tens of thousands of nodes are deployed in warehouse-sized HPC and data centers [8]. Key properties of such networks are determined by their *topologies*: the arrangement of nodes and cables.

Several metrics have to be taken into account while designing an efficient topology. First, high bandwidth is indispensable as many applications perform all-to-all communication [38]. Second, networks can account for as much as 33% of the total system cost [27] and 60% of the overall system energy consumption [2] and thus they should be cost and power efficient. Third, low end-to-end latency is important for many applications, e.g., in high-frequency trading. Finally, topologies should be resilient to link failures.

In this paper we show that *lowering network diameter* not only reduces the latency but also the cost of a network and the amount of energy it consumes while maintaining high bisection bandwidth. Lowering the diameter of a network has two effects. First, it reduces energy consumption as each packet traverses a smaller number of SerDes. Another consequence is that packets visit fewer sinks and router buffers and will thus be less likely to contend with other packets flowing through the network. This enables us to reduce the number of costly routers and connections while maintaining high bisection bandwidth.

The well-known fat tree topology [30] is an example of a network that provides high bisection bandwidth. Still, every packet has to traverse many connections as it first has to move up the tree to reach a core router and only then goes down to its destination.

diameter. Other topologies, such as Dragonfly [28], reduce the diameter to three, but their structure also limits bandwidth and, as we will show, has a negative effect on resiliency.

In this work, we propose a new topology, called **Slim Fly**, which further reduces the diameter and thus cost, energy consumption, and the latency of the network while maintaining high bandwidth and resiliency. **Slim Fly** is based on graphs with lower diameter for a given router radix and, in this sense, approaching the optimal diameter for a given router technology. Figure 1 motivates **Slim Fly** by comparing the average number of network hops for random uniform traffic using minimal path routing on different network topologies.

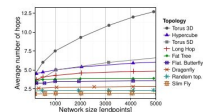


Fig. 1. Comparison of the average number of hops (uniform traffic) in **Slim Fly** and other networks. Topologies are in balanced or close to balanced configuration (explained in Section III), allowing for higher input radixes. **Slim Fly** enables us to construct cost-efficient full-bandwidth networks with over 100K endpoints with diameter two using readily available high radix routers (e.g., 64-port Black Widow [25] or Mellanox H8-port Director [5]). Larger networks with up to tens of millions of endpoints can be constructed with diameter three as discussed in Section II-A.

The main contributions of this work are:

- We design and analyze a new class of cost effective low-diameter network topologies called **Slim Fly**.
- We discuss and evaluate different deadlock-free minimal and adaptive routing strategies and we compare them to existing topologies and approaches.
- We show that, in contrast to the first intuition, **Slim Fly**, using fewer cables and routers, is more tolerant toward links failures than comparable Dragonflies.
- We show a physical layout for a datacenter or an HPC center network and a detailed cost and energy model.

*Numbers for random topologies are updated from values obtained using the simulator to the lower ones calculated with analytical formulas.

jupiter_rising.pdf 57.4%

Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network

Xpander: Towards Optimal-Performance Datacenters

Asaf Valadarsky*
asaf.valadarsky@mail.huji.ac.il

Gal Shafat*
gal.shafat@mail.huji.ac.il

Michael Dinitz*
mdinitz@cs.huji.ac.il

Michael Schpirar*
schspirar@huji.ac.il

Abstract

Despite extensive efforts to meet ever-growing demands, today's datacenters often exhibit far-from-optimal performance in terms of network utilization, resiliency to failures, cost efficiency, incremental expandability, and more. Consequently, many novel architectures for high-performance datacenters have been proposed. We show that the benefits of state-of-the-art proposals are, in fact, derived from the fact that they are (implicitly) utilizing “expander graphs” (aka expanders) in their network topologies, thus unifying a unifying theme of these proposals. We observe, however, that these proposals are not optimal with respect to performance, do not scale, or suffer from seemingly insurmountable deployment challenges. We leverage these insights to present **Xpander**, a novel datacenter architecture that achieves near-optimal performance and provides a tangible alternative to existing datacenter designs. **Xpander**'s design turns ideas from the rich graph-theoretic literature on constructing optimal expanders into an operational reality. We evaluate **Xpander** via theoretical analyses, extensive simulations, experiments with a network emulator, and an implementation on an SNN-capable network switch. Our results demonstrate that **Xpander** significantly outperforms both traditional and proposed datacenter designs. We discuss challenges to real-world deployment and explain how these can be resolved.

1 Introduction

The rapid growth of Internet services is placing tremendous demands on datacenters. Yet, as evidenced by the extensive research on improving datacenter performance [28, 29, 30, 31, 32, 33, 34, 35], today's datacenters often exhibit far-from-optimal performance in terms of network utilization, resiliency to failures, cost efficiency, amenability to incremental growth, and beyond.

1.1 The Secret to High Performance

We show that state-of-the-art proposals for next-generation datacenters, e.g., low-diameter networks such as **Slim Fly** [8], or random networks like **Jellyfish** [6], have an in-

the network topology and exploiting the diversity of short paths afforded by expanders for efficient delivery of data traffic. Thus, our first contribution is shedding light on the underlying reason for the empirically good performance of previously proposed datacenter architectures, by showing that these proposals are specific points in a much larger design space of “expander datacenters”. We observe, however, that these points are either not sufficiently close to optimal performance-wise, are inherently not scalable, or face significant deployment and maintenance challenges (e.g., in terms of unpredictability and wiring complexity), thus unifying a unifying theme of these proposals. We observe, however, that these proposals are not optimal with respect to performance, do not scale, or suffer from seemingly insurmountable deployment challenges. We leverage these insights to present **Xpander**, a novel datacenter architecture that achieves near-optimal performance and provides a tangible alternative to existing datacenter designs. **Xpander**'s design turns ideas from the rich graph-theoretic literature on constructing optimal expanders into an operational reality. We evaluate **Xpander** via theoretical analyses, extensive simulations, experiments with a network emulator, and an implementation on an SNN-capable network switch. Our results demonstrate that **Xpander** significantly outperforms both traditional and proposed datacenter designs. We discuss challenges to real-world deployment and explain how these can be resolved.

We argue that the quest for high-performance datacenter designs is inextricably intertwined with the rich body of research in mathematics and computer science on building good expanders. We seek a point in this design space that offers near-optimal performance guarantees while providing a practical alternative for today's datacenters. In terms of cabling, physical layout, backwards compatibility with today's protocols, and more. We present **Xpander**, a novel expander datacenter architecture carefully engineered to achieve both these desiderata.

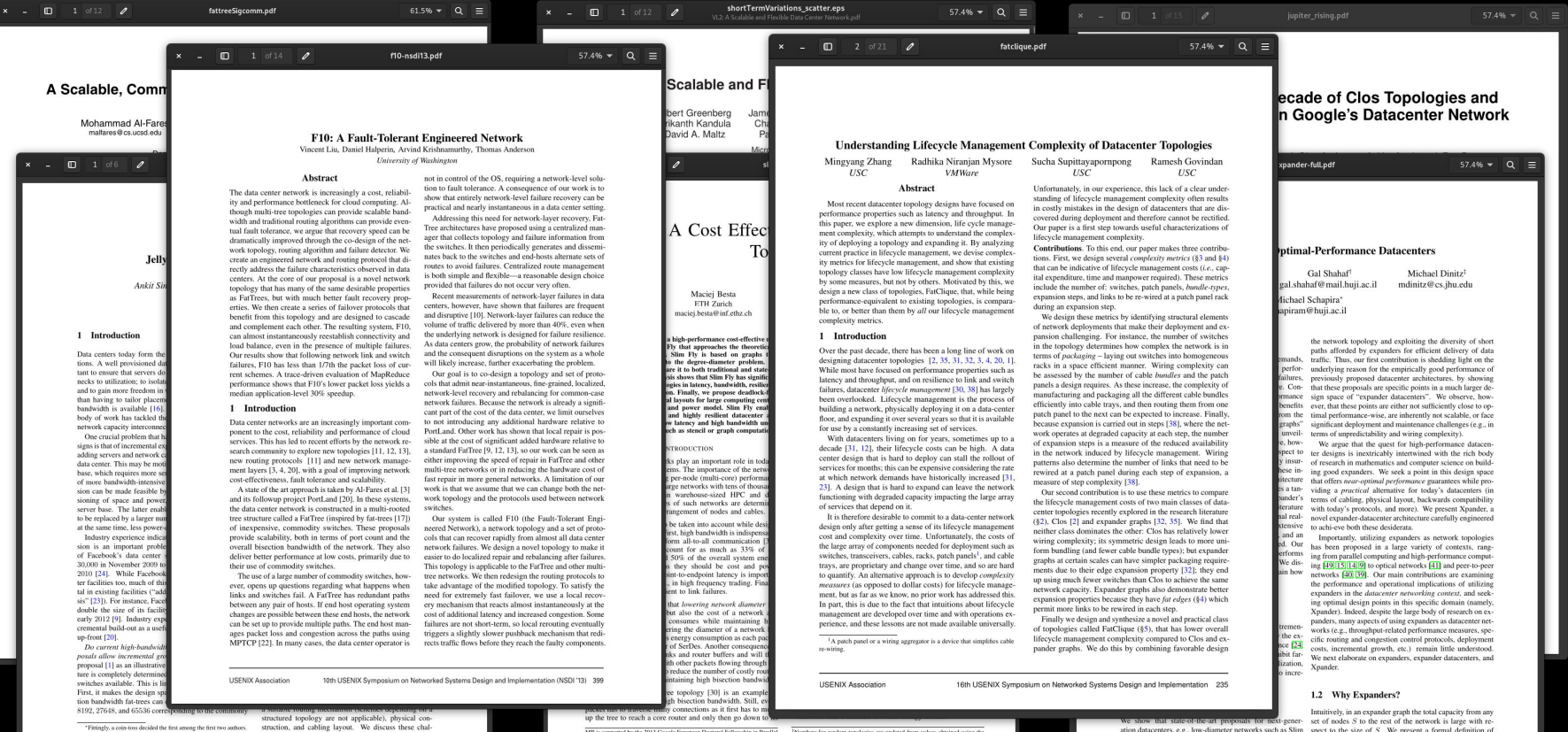
Importantly, utilizing expanders as network topologies has been proposed in a large variety of contexts, ranging from parallel computing and high-performance computing [6, 9, 10, 11, 12, 13] to optical networks [4] and peer-to-peer networks [40, 39]. Our main contributions are examining the performance and operational implications of utilizing expanders in the datacenter networking context, and seeking optimal design points in this specific domain (namely, **Xpander**). Indeed, despite the large body of research on expanders, many aspects of using expanders as datacenter networks (Section II) since this implies that it is an expensive routing and congestion control problems, deployment costs, incremental growth, etc.) remain little understood. We next elaborate on expanders, expander datacenters, and **Xpander**.

1.2 Expander?

Intuitively, in an expander graph the total capacity from any set of nodes S to the rest of the network is large with respect to the size of S . We present a formal definition of

*Formerly, a co-author declined the first among the first two authors.

Traditional Approach: Static Datacenter Topologies



A Scalable, Commodity-Friendly, and Fault-Tolerant Network Topology

Mohammad Al-Fares
malfares@cs.ucsd.edu

F10: A Fault-Tolerant Engineered Network
Virent Liu, Daniel Halperin, Arvind Krishnamurthy, Thomas Anderson
University of Washington

Abstract

The data center network is increasingly a cost, reliability and performance bottleneck for cloud computing. Although multi-tree topologies can provide scalable bandwidth and traditional routing algorithms can provide eventual fault tolerance, we argue that recovery speed can be dramatically improved through the co-design of the network topology, routing algorithm and failure detector. We create an engineered network and routing protocol that directly address the failure characteristics observed in data centers. At the core of our proposal is a novel network topology that has many of the same desirable properties as FatTrees, but with much better fault recovery properties. We then create a series of failover protocols that benefit from this topology and are designed to cascade and complement each other. The resulting system, F10, can almost instantaneously reestablish connectivity and load balance, even in the presence of multiple failures. Our results show that following the network link and switch failures, F10 has less than 170ns the packet loss of current schemes. A trace-driven evaluation of MapReduce performance shows that F10's lower packet loss yields a median application-level 30% speedup.

1 Introduction

Data center networks are an increasingly important component to the cost, reliability and performance of services. This has led to recent efforts by the network research community to explore new topologies [1, 12, 13], new routing protocols [11] and new network management layers [3, 4, 20], with a goal of improving network cost-effectiveness, fault tolerance and scalability.

At the core of the architecture is a novel network topology and its following project PortLand [20]. In these systems, the data center network is constructed in a multi-rooted tree structure called a *FatTree* (inspired by fat-trees [17]) of inexpensive, commodity switches. These proposals provide scalability, both in terms of port count and the overall bioecological bandwidth of the network. They also deliver better performance at low costs, primarily due to their use of commodity switches.

The use of a large number of commodity switches, however, opens up questions regarding what happens when links and switches fail. A *FatTree* has redundant paths between any pair of hosts. In end host operating system changes are possible between these end hosts, the network can be set up to provide multiple paths. The end host manages packet loss and congestion across the paths using NPTCP [22]. In many cases, the data center operator is

not in control of the OS, requiring a network-level solution to fault tolerance. A consequence of our work is to show that entirely network-level failure recovery can be practical and nearly instantaneous in a data center setting. Addressing this need for network-layer recovery, *FatTree* architectures have proposed using a centralized manager that collects topology and failure information from the switches. It then periodically generates and disseminates back to the switches and end-hosts alternate sets of routes to avoid failures. Centralized route management is both simple and flexible—a reasonable design choice provided that failures do not occur very often.

Recent measurements of network-layer failures in data centers, however, have shown that failures are frequent and disruptive [10]. Network-layer failures can reduce the volume of traffic delivered by more than 40%, even when the underlying network is designed for failure resilience. As data centers grow, the probability of network failures and the consequent disruptions on the system as a whole will likely increase, further exacerbating the problem.

Our goal is to co-design a topology and a set of protocols that admit near-instantaneous, fine-grained, localized, network-level recovery and rebalancing for common case network failures. Because the network is already a significant part of the cost of the data center, we limit ourselves to not introducing any additional hardware relative to PortLand. Other work has shown that local repair is a part of the cost of significant added hardware relative to a standard *FatTree* [9, 12, 13], so our work can be seen as an improvement in the speed of repair. A *FatTree* and other multi-tree networks or in reducing the hardware cost of repair in more general networks. A limitation of our current design is that we assume that we can change both the network topology and the protocols used between network switches.

Our system is called F10 (the *FatTree* Engineered Network), a network topology and a set of protocols that can recover rapidly from almost all data center network failures. We design a novel topology to make it easier to do localized repair and rebalancing after failures. This topology is applicable to the *FatTree* and other multi-tree networks. We then redesign the routing protocols to take advantage of the modified topology. To satisfy the need for extremely fast failover, we use a local recovery mechanism that reacts almost instantaneously at the cost of additional latency and increased congestion. Some failures are not short-term, so local reconfiguration eventually triggers a slightly slower pushback mechanism that redirects traffic flows before they reach the faulty components.

Scalable and Fault-Tolerant Network Topologies

Robert Greenberg, Jami
Kantharaj, David A. Maltz

Understanding Lifecycle Management Complexity of Datacenter Topologies
Mingyang Zhang, Radhika Nirwan Mysore, Sucha Supitayapornpong, Ramesh Govindan
USC USC VMWare USC USC

Abstract

Most recent datacenter topology designs have focused on performance properties such as latency and throughput. In this paper, we explore a new dimension, life cycle management complexity, which attempts to understand the complexity of deploying a topology and expanding it. By analyzing current practice in lifecycle management, we derive complexity metrics for lifecycle management, and show that existing topology classes have low lifecycle management complexity by some measures, but not by others. Motivated by this, we design a new class of topologies, *FatTree*, that, while being performance-equivalent to existing topologies, is comparable to, or better than them by all our lifecycle management complexity metrics.

1 Introduction

Over the past decade, there has been a long line of work on designing datacenter topologies [2, 35, 31, 32, 5, 4, 20, 11]. While most have focused on performance properties such as latency and throughput, and on resilience to link and switch failures, datacenter *lifecycle management* [30, 38] has largely been overlooked. Lifecycle management is the process of building a network, physically deploying it on a data center, and expanding it over successive generations of expansion by use of a constantly increasing set of services.

With datacenters living on for years, sometimes up to a decade [13, 12], their lifecycle costs can be high. A data center design that is hard to deploy can stall the rollout of services for months; this can be expensive considering the rate at which network demands have historically increased [31, 23]. A design that is hard to expand can leave the network, a functioning with degraded capacity impacting the large array of services that depend to it.

It is therefore desirable to commit to a data-center network design only after getting a sense of its lifecycle management cost and complexity over time. Unfortunately, the costs of large arrays of components needed for deployment such as switches, transceivers, cables, racks, patch panels¹, and cable trays, are proprietary and change over time, and so are hard to quantify. An alternative approach is to develop *complexity measures* (as opposed to dollar costs) for lifecycle management, but as far as we know, no prior work has addressed this. In part, this is due to the fact that intuitions about lifecycle management are developed over time and with operations experience, and these lessons are not made available universally.

¹A patch panel or a wiring aggregator is a device that simplifies cable re-wiring.

Unfortunately, in our experience, this lack of a clear understanding of lifecycle management complexity often results in costly mistakes in the design of datacenters that are discovered during deployment and therefore cannot be rectified. Our paper is a first step towards useful characterizations of lifecycle management complexity.

Contributions. To this end, our paper makes three contributions. First, we design several *complexity metrics* (§3 and §4) that can be indicative of lifecycle management costs (i.e., capital expenditures, time and manpower required). These metrics include the number of: switches, patch panels, *bundle-trays*, expansion steps, and links to be re-wired at a patch panel during an expansion step.

We design these metrics by identifying structural elements of network deployments that make their deployment and expansion challenging. For instance, the number of switches in the topology determines how complex the network is in terms of *packetaging*—laying out switches into homogeneous racks in a space efficient manner. Wiring complexity can be assessed by the number of cable *bundles* and the patch panels a design requires. As these increase, the complexity of manufacturing and packaging all the different cable bundles efficiently into cable trays, and then routing them from one patch panel to the next can be expected to increase. Finally, because expansion is carried out in steps [38], where the network operates at degraded capacity at each step, the number of expansion steps is a measure of the reduced availability in the network induced by lifecycle management. Wiring patterns also determine the number of links that need to be rerouted at a patch panel during each step of expansion, a measure of step complexity [38].

Our second contribution is to use these metrics to compare the lifecycle management costs of two main classes of data-center topologies recently explored in the research literature (§2). Clos [2] and expander graphs [2, 35]. We find that neither class dominates the other. Clos has relatively lower wiring complexity; its symmetric design leads to more uniform bundling (and fewer cable bundle types); but expander graphs at certain scales can have simpler packaging requirements due to their edge expansion property [23]; they end up using much fewer switches than Clos to achieve the same network capacity. Expander graphs also demonstrate better expansion properties because they have *fewer edges* (§4) which permit more links to be rewired in each step.

Finally we design and synthesize a novel and practical class of topologies called *FatTree* (§5), that has lower overall lifecycle management complexity compared to Clos and expander graphs. We do this by combining favorable design

Evolution of Clos Topologies and in Google's Datacenter Network

Xpander, full.pdf

Gal Shabat¹, Michael Dimitz²
gal.shabat@mail.huji.ac.il mdimitz@cs.huji.ac.il

Optimal-Performance Datacenters

Michael Schapira¹
mschapira@huji.ac.il

the network topology and exploiting the diversity of short paths underlying by expanders for efficient delivery of data traffic. Thus, our first contribution is shedding light on the underlying reason for the empirically good performance of previously proposed datacenter architectures, by showing that these proposals are specific points in a much larger design space of “expander datacenters”. We observe, however, that these points are either not sufficiently close to optimal performance-wise, are inherently not scalable, or face significant deployment and maintenance challenges (e.g., in terms of unpredictability and wiring complexity).

We argue that the quest for high-performance datacenter designs is inextricably intertwined with the rich body of research in mathematics and computer science on building good expanders. We seek a point in this design space that offers *near-optimal performance* guarantees while providing a *practical* alternative for today’s datacenters. (In terms of cabling, physical layout, backwards compatibility with today’s protocols, and more). We present Xpander, a novel expander datacenter architecture carefully engineered to achieve both these desiderata.

Importantly, utilizing expanders as carefully engineered has been proposed in a large variety of contexts, ranging from parallel computing and high-performance computing [6, 9, 11, 13, 19] to optical networks [4] and peer-to-peer networks [8, 29]. Our main contributions are, examining the performance and operational implications of utilizing expanders in data center networking context, and seeking optimal design points in this specific domain (namely, Xpander). Indeed, despite the large body of research on expanders, many aspects of using expanders as datacenter network topologies (e.g., how to design expanders, specific routing and congestion control protocols, deployment costs, incremental growth, etc.) remain little understood. We next elaborate on expanders, expander datacenters, and Xpander.

1.2 Why expanders?

Intuitively, in an expander graph the total capacity from any set of nodes S to the rest of the network is large with respect to the size of S . We present a formal definition of

¹Formerly, a co-lead on the design of the first among the first two authors.

NSDI is supported by the 2013 Google European Digital Center in Paris

Numbers for random topologies are updated from values obtained using the

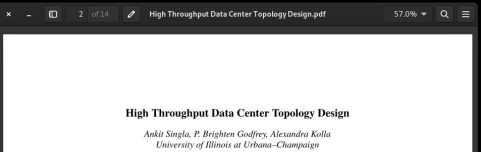
We show that state-of-the-art proposals for next-generation datacenters, e.g., low-dimensional networks such as Slim

Traditional Approach: Static Datacenter Topologies

Which topology has better throughput?

Traditional Approach: Static Datacenter Topologies

Which topology is better though?



Abstract

With high throughput networks acquiring a crucial role in supporting data-intensive applications, a variety of data center network topologies have been proposed to achieve high capacity at low cost. While this work explores a large number of design points, even in the limited case of a network of identical switches, no proposal has been able to claim any notion of optimality. The case of heterogeneous networks, incorporating multiple line-speeds and port counts at data centers goes over time, introduces even greater complexity.

In this paper, we present the first non-trivial upper-bound on network throughput under uniform traffic patterns for any topology with identical switches. We then show that random graphs achieve throughput surprisingly close to this bound, within a few percent at the scale of a few thousand servers. Apart from demonstrating that homogeneous topology designs may be reaching its limits, this result also motivates one use of random graphs as building blocks for designs of heterogeneous networks. Given a heterogeneous pool of network switches, we explore through experiments and analysis, how the distribution of server access switches and the interconnection of switches affect network throughput. We apply these insights to a real-world heterogeneous data center topology, VLL2, demonstrating as much as 43% higher throughput with the same equipment.

1 Introduction

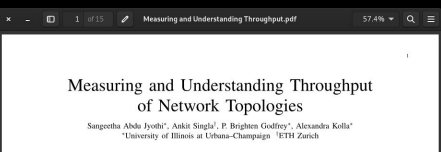
Data centers are playing a crucial role in the rise of Internet services and big data. In turn, efficient data center operations depend on high capacity networks to ensure that computations are not bottlenecked on communication. As a result, the problem of designing massive high-capacity network interconnects has become more important than ever. Numerous data center network architectures have been proposed in response to this need [2, 10–15, 20, 23, 25, 36], exhibiting a variety of network topologies to achieve high throughput, ranging from fat-trees and other Clos networks [2, 13] to modified generalized hypercubes [14] to small world networks [21] and uniform random graphs [23].

However, while this extensive literature exposes several points in the topology design space, even in the limited case of a network of identical switches, it does not answer a fundamental question: *How far are we from throughput-optimal topology design?* The case of heterogeneous networks, i.e., networks composed of switches or servers with disparate capabilities, introduces even greater complexity. Heterogeneous network equipment is, in fact, the common case in the typical data center: servers connect to top-of-rack (ToR) switches, which connect to aggregation switches, which connect to core switches, with each type of switch possibly having a different number of ports and with some variations in line-speed. For instance, the ToR may have both 10 Gbps and 100 Gbps connections, while the rest of the network may have only 10 Gbps links. Further, as the network expands over the years and new, more powerful equipment is added to the data center, one can expect more heterogeneity — each year the number of ports supported by non-blocking commodity Ethernet switches increases. While line-speed changes are slow, the move to 10 Gbps and even 40 Gbps is happening now, and higher line-speeds are expected in the near future.

In spite of heterogeneity being commonplace in data center networks, very little is known about heterogeneous network designs. For instance, there is no clarity on whether the traditional fat-tree/aggregation core organization is superior to a “flatter” network without such a switch hierarchy, or on whether powerful core switches should be connected directly together, or spread more evenly throughout the network.

The goal of this paper is to develop an understanding of how to design high throughput network topologies or limited sets, even when heterogeneous components are involved, and to apply this understanding to improve real-world datacenter designs. This is a non-trivial, combinatorial topology design is hard in general, because of the network expansion of the number of possible networks with size. Consider, for example, the related *degree-diameter* problem [9], a well-known graph theory problem where the quest is to pack the largest possible number of nodes into a graph while adhering to constraints on both degree and the diameter. Numerous initial optimal solutions are known for a fixed d and only a few for d and n both growing.

Designing for low network diameter is related to designing for high throughput, because the number of hops in the network limits line capacity to deliver each packet, see discussion in [23].



Abstract—High throughput is of particular interest in data center and HPC networks. Although myriad network topologies have been proposed, a broad head-to-head comparison across topologies and across traffic patterns is absent, and the right way to compare worst-case throughput performance is a subtle problem.

In this paper, we develop a framework to benchmark the throughput of network topologies using a two-pronged approach. First, we study performance on a variety of studies, and experimentally measure traffic patterns (ToR, fat-tree, and other) to show how to measure worst-case throughput by generating a worst-case TM for any given topology. We apply the framework to study the performance of these TMs in a wide range of network topologies, revealing insights into the performance of topologies with scaling, robustness of performance across TMs, and the effect of network workload placement. Our evaluation code is freely available.

Our key contributions, then, are to (1) develop a heuristic to measure worst-case throughput, and (2) provide an extensive benchmarking of a variety of topologies using a variety of TMs — TMs generated from real-world measurements, synthetic TMs, and finally our new worst-case TMs. We discuss each of these in more detail.

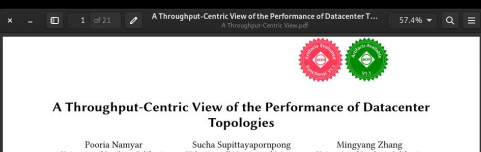
(a) **We evaluate whether our head-to-head metrics, such as bisection bandwidth and space cut, solve the problem of estimating worst-case throughput.** A number of studies (e.g., [6], [8], [9], [13], [25], [36]) employ cut metrics. It has been noted [2] that bisection bandwidth does not always predict throughput in a limited setting. [2] But do our metrics measure worst-case throughput? We show that it does not, in proving the existence of two families of networks, $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ has a higher cut metric even though $\mathcal{B}$ supports a exponentially higher worst-case throughput. The absence of a well-specified benchmark complicates research on network design, making it difficult to evaluate and compare networks. Our goal is to build a framework for accurate and consistent measurement of the throughput of network topologies, and this framework is benchmark, proposed data center and HPC topologies.

To accomplish this, we need metrics for comparison of throughput, and this turns out to be a subtle problem. When put as a question by using particular workloads, or traffic matrices (TMs), the immediate question is: What TM can one measure to be a variety of workloads? We show that one could possibly insight into the effect of topological structure on particular use cases reflected by these specific TMs. However, we argue it is useful to go beyond this. In HPC and data center networks, TMs may vary widely depending on the use-case and across time as applications spin up and down or migrate [19, [23], [25]]. Although some applications may well utilize certain topologies and known worst-case traffic patterns [3], [3] can be avoided in such cases, a mix of multiple applications could still produce an unintended difficult TM. In fact, [2] observes that network contention among applications sharing an HPC system will worsen in the near future (even after accounting for the expected increase in network speeds) if HPC applications grow in size and variety. Hence, it is useful to understand the worst-case performance of a topology for any TM. However, currently, there does not exist a systematic way to evaluate the worst-case throughput achievable in a network and to identify the traffic pattern responsible for it.

Our key contributions, then, are to (1) develop a heuristic to measure worst-case throughput, and (2) provide an extensive benchmarking of a variety of topologies using a variety of TMs — TMs generated from real-world measurements, synthetic TMs, and finally our new worst-case TMs. We discuss each of these in more detail.

(a) **We evaluate whether our head-to-head metrics, such as bisection bandwidth and space cut, solve the problem of estimating worst-case throughput.** A number of studies (e.g., [6], [8], [9], [13], [25], [36]) employ cut metrics. It has been noted [2] that bisection bandwidth does not always predict throughput in a limited setting. [2] But do our metrics measure worst-case throughput? We show that it does not, in proving the existence of two families of networks, $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ has a higher cut metric even though $\mathcal{B}$ supports a exponentially higher worst-case throughput. The absence of a well-specified benchmark complicates research on network design, making it difficult to evaluate and compare networks. Our goal is to build a framework for accurate and consistent measurement of the throughput of network topologies, and this framework is benchmark, proposed data center and HPC topologies.

To accomplish this, we need metrics for comparison of throughput, and this turns out to be a subtle problem. When put as a question by using particular workloads, or traffic matrices (TMs), the immediate question is: What TM can one measure to be a variety of workloads? We show that one could possibly insight into the effect of topological structure on particular use cases reflected by these specific TMs. However, we argue it is useful to go beyond this. In HPC and data center networks, TMs may vary widely depending on the use-case and across time as applications spin up and down or migrate [19, [23], [25]]. Although some applications may well utilize certain topologies and known worst-case traffic patterns [3], [3] can be avoided in such cases, a mix of multiple applications could still produce an unintended difficult TM. In fact, [2] observes that network contention among applications sharing an HPC system will worsen in the near future (even after accounting for the expected increase in network speeds) if HPC applications grow in size and variety. Hence, it is useful to understand the worst-case performance of a topology for any TM. However, currently, there does not exist a systematic way to evaluate the worst-case throughput achievable in a network and to identify the traffic pattern responsible for it.



Abstract

While prior work has explored many proposed datacenter designs, only two designs, Clos-based and expander-based, are generally considered practical because they can scale using commodity switching fabric. Prior work has used different metrics, bisection bandwidth and throughput, for evaluating these topologies at scale. Little is known, theoretically or practically, how these metrics relate to each other. Explaining these metrics of these topologies, we present an upper bound on their throughput, then show that this upper bound better estimates worst-case throughput than all previously proposed throughput estimators and scales better than most of them. Using this upper bound, we show that for expander-based Datacenter topology designs, Two distinct classes of topology designs have emerged in recent years. Clos [33] based designs include Fat-trees [1], VLL2 [16], Jupiter [6], and Facebook Faber [1], and failure-resilient variants, such as F10 [36]. These hierarchical designs are regular in which a switch either connects to 4 servers, or more at all (Figure 1). More recent alternative designs target lower installation costs and/or lower power management than Clos-based topologies. These designs employ an expander graph to interconnect switches, and include Jellyfish [41], Xander [4], and FatSpine [32]. These topologies are *not* regular: every switch connects to n servers (Figure 1). In both cases, each server connects to exactly one switch.

Measures of throughput capacity. The capacity of the data center network limits the performance of applications it hosts. Intuitively, a topology with enough capacity to permit every server to send traffic at full line rate implies full application designs operate on plane application resources anywhere in the network without impacting performance, and this placement flexibility enables applications to be more cost-efficient and more robust to correlated failures (e.g., of an entire rack or power domain) [33, 21, 35, 42].

More prior work [16, 1, 33, 35, 42] has used the network’s bisection bandwidth, the smallest aggregate capacity of the links crossing the worst-case cut in the network, to estimate the network’s bisection bandwidth, as a measure of its capacity. A topology has full bisection bandwidth if its bisection bandwidth is at least equal to half the total aggregate capacity of the links crossing the topology permits arbitrary application placement.

Other work [24, 26, 27, 35, 51] has explored an alternative measure of network capacity, *throughput*, defined as follows. The throughput of a topology is the maximum number of flows that the topology can support the traffic matrix, T [7, 47].

Other throughput designs, such as Jellyfish [41] and Jellyfish [41], do not scale the size of the network data centers, so we do not consider them in this paper [37].

Reconfigurable Datacenter Networks

- Generalization of the design space: *Topology can change over time*
- Static networks are a special case

Reconfigurable Datacenter Networks

- Generalization of the design space: *Topology can change over time*
- Static networks are a special case



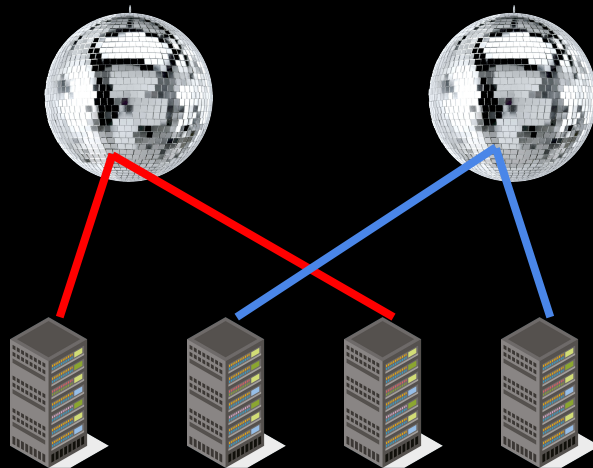
Sirius [Sigcomm 2020]

Reconfigurable Datacenter Networks

- Generalization of the design space: *Topology can change over time*
- Static networks are a special case



Sirius [Sigcomm 2020]



ProjectoR [Sigcomm 2016]

Reconfigurable Datacenter Networks

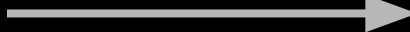


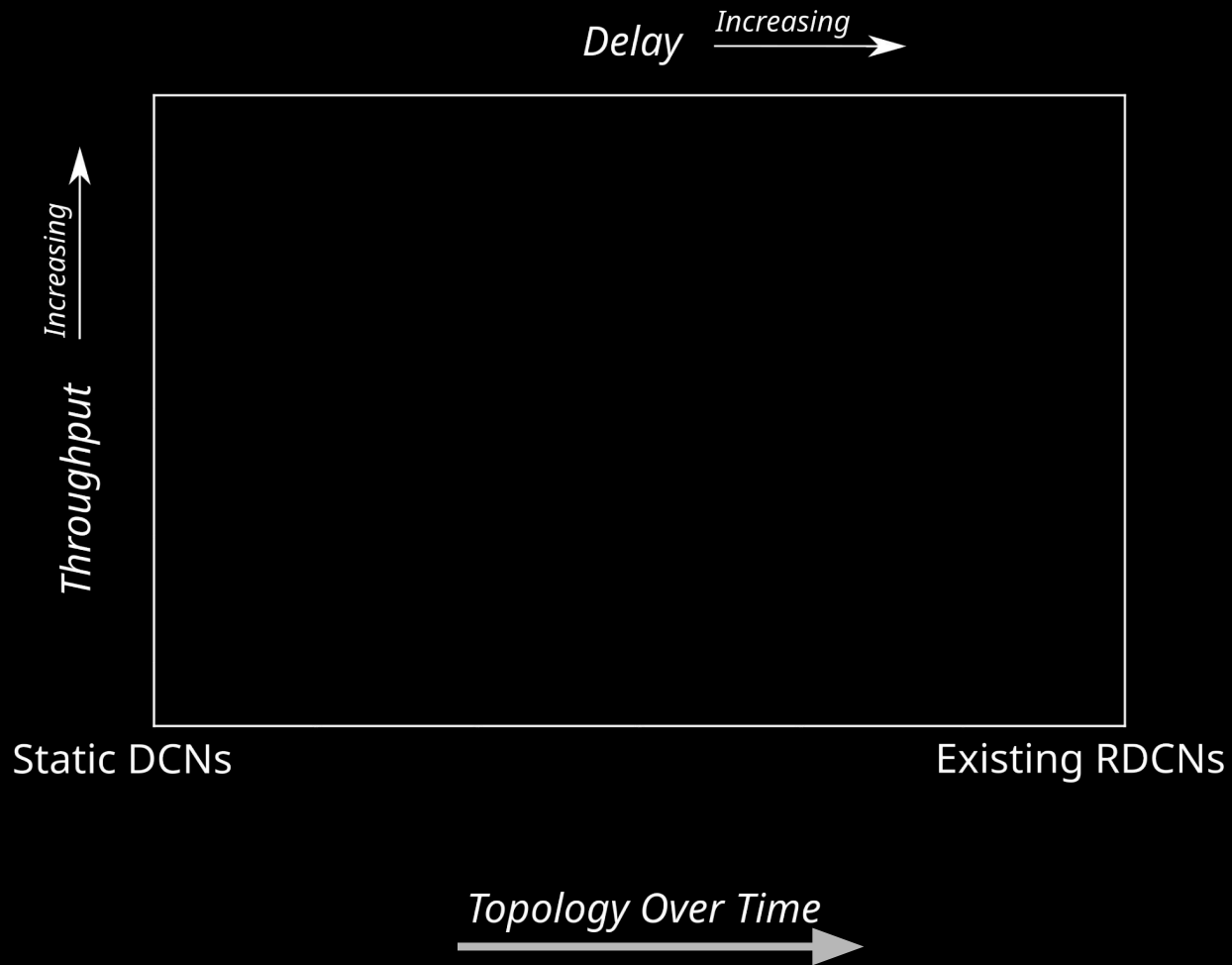
Reconfigurable Datacenter Networks

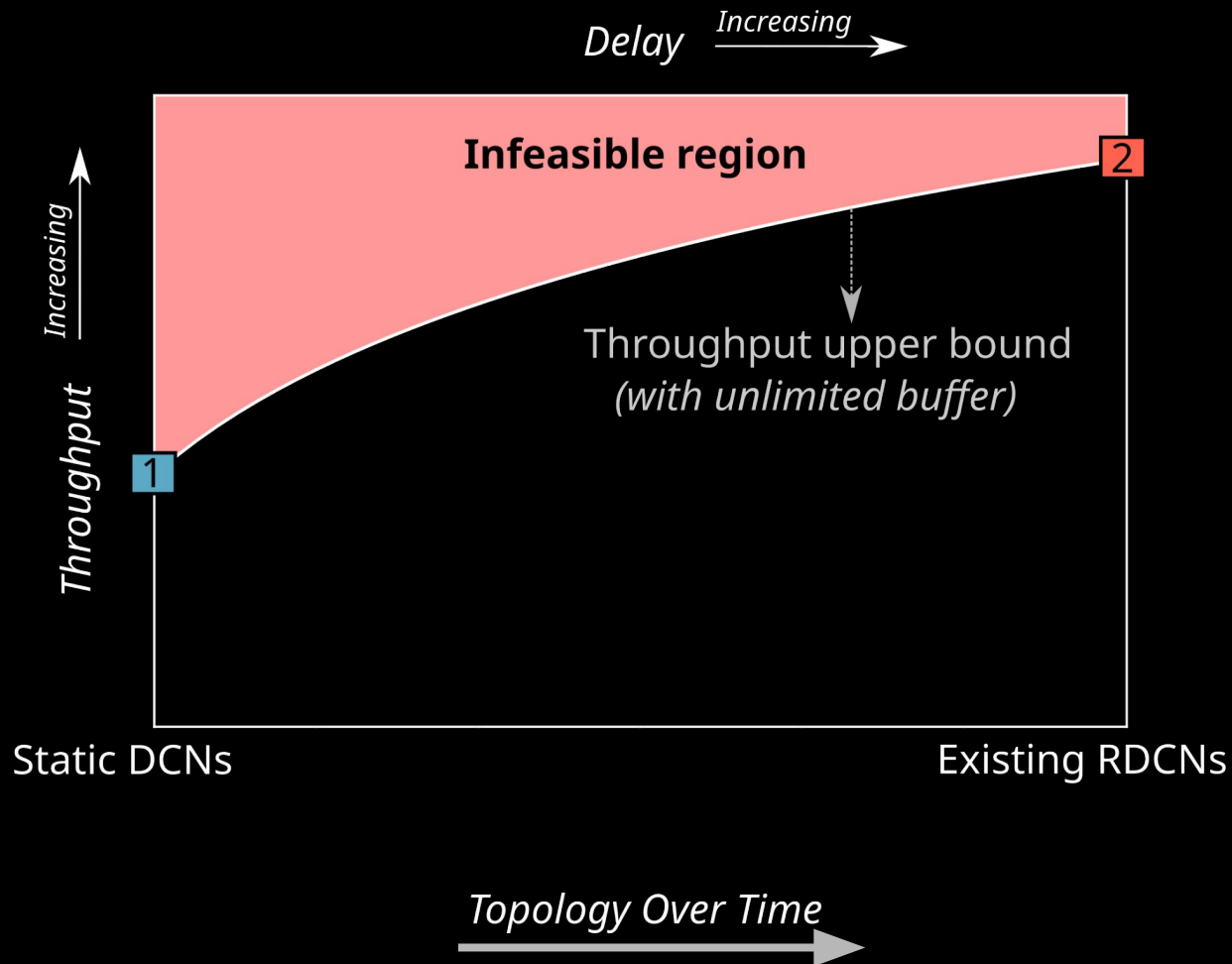
Which topology has better throughput?

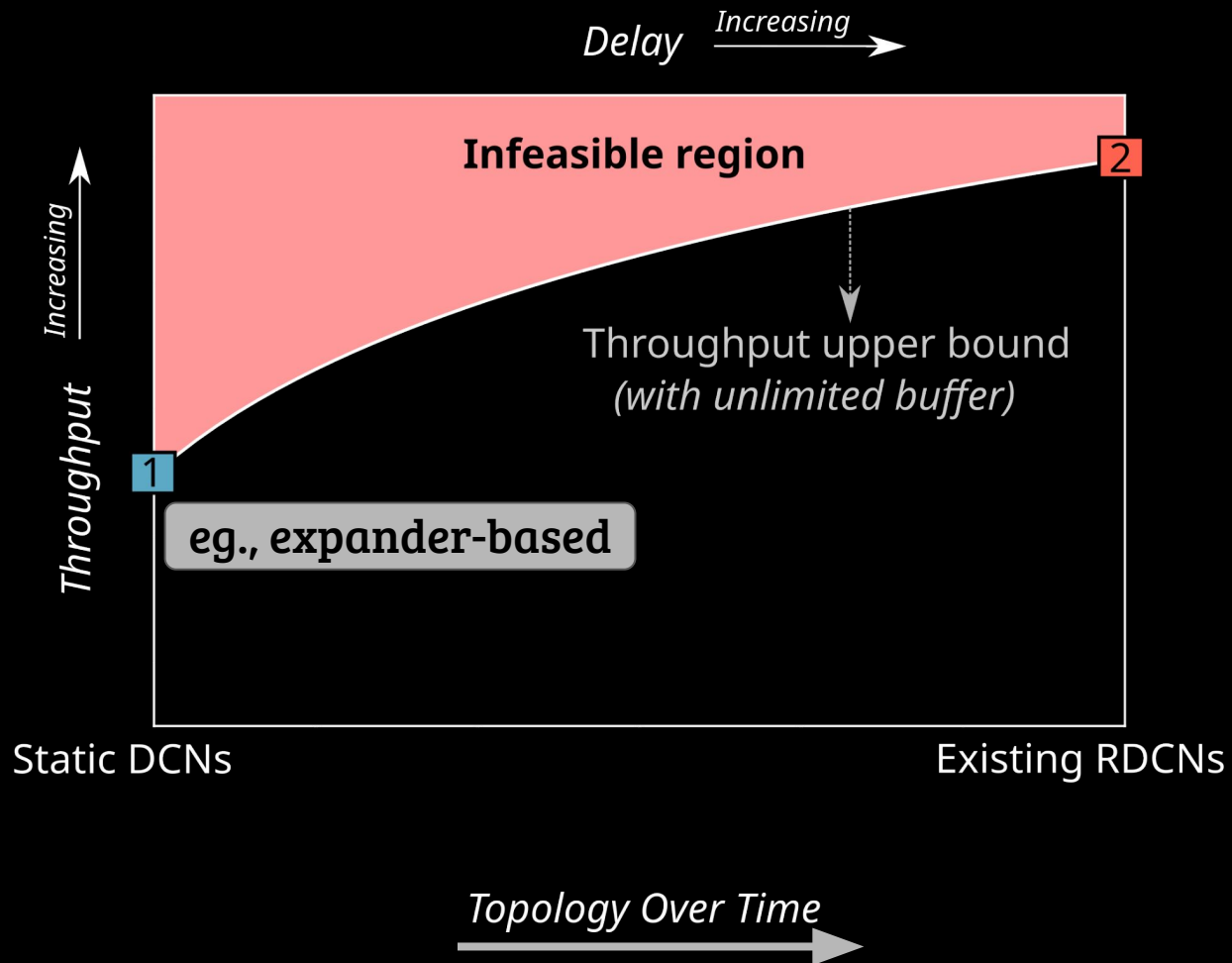


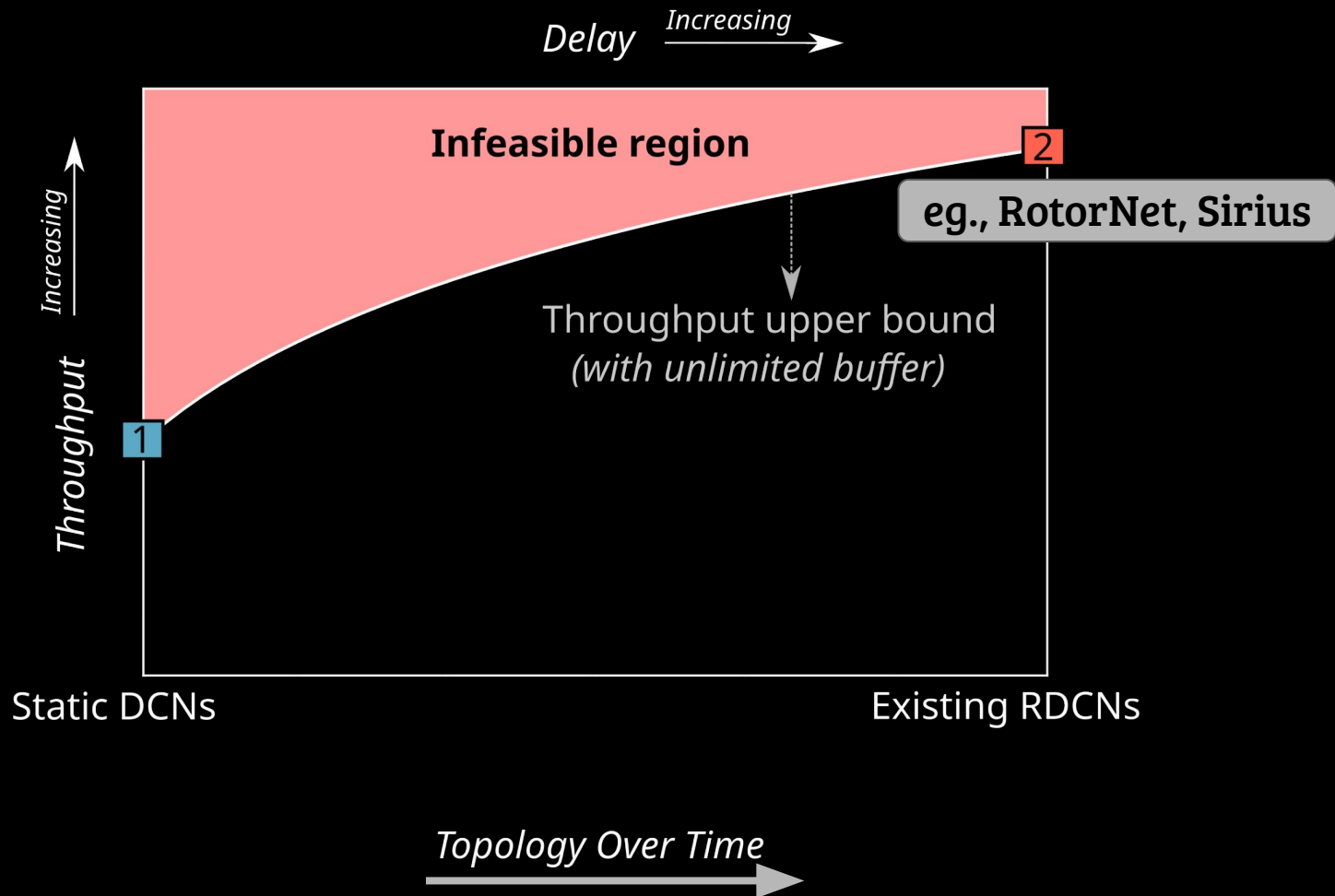
Topology Over Time

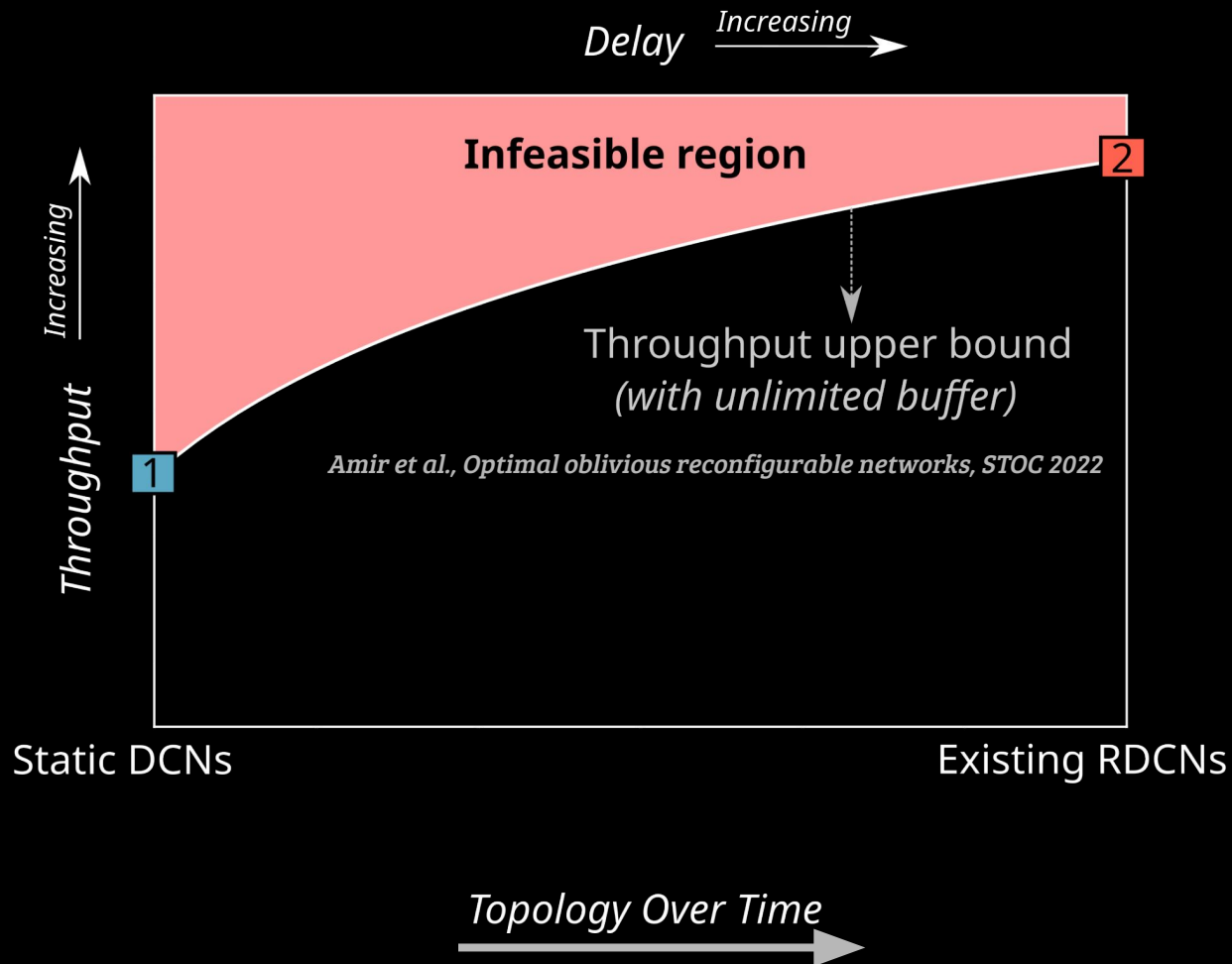


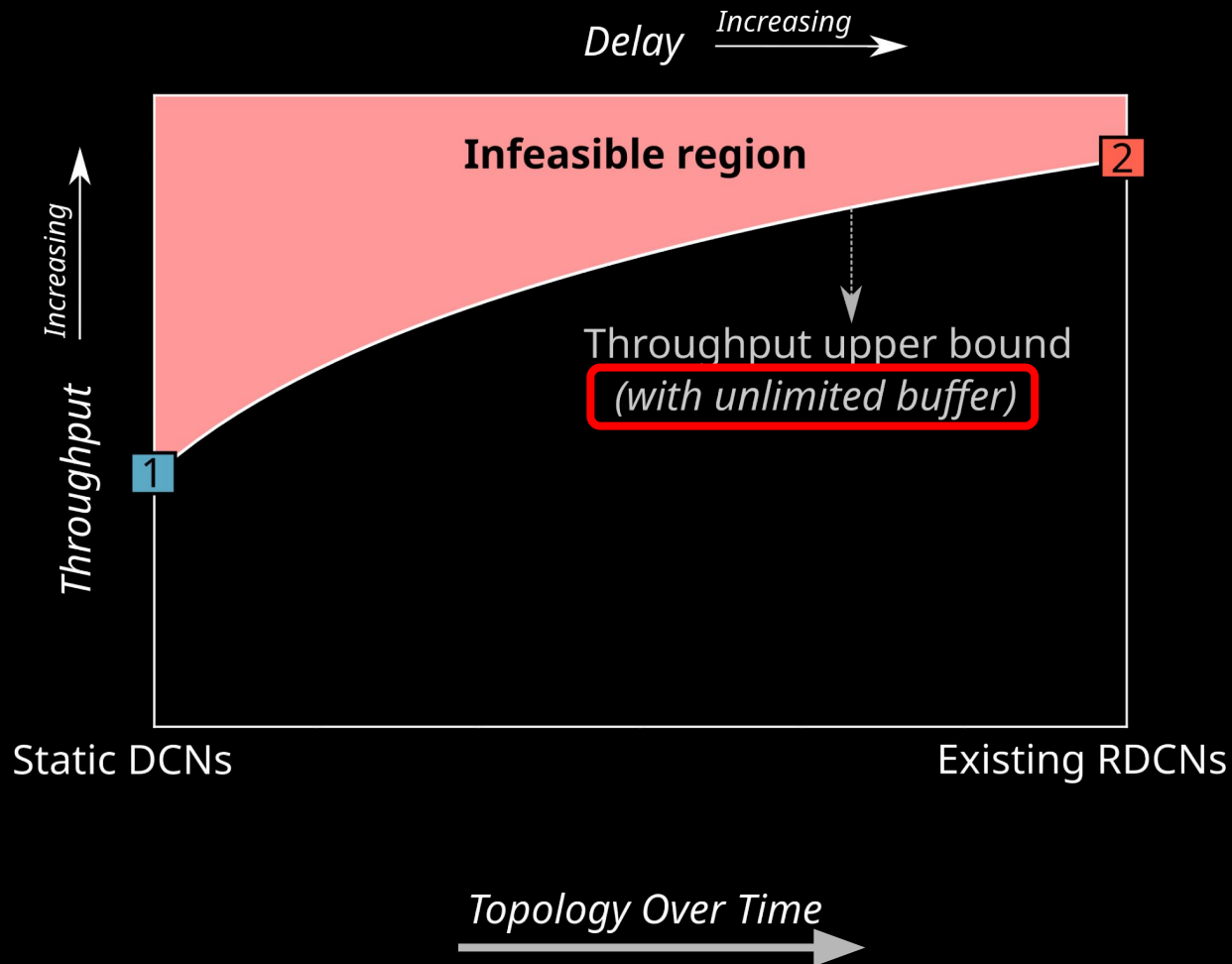


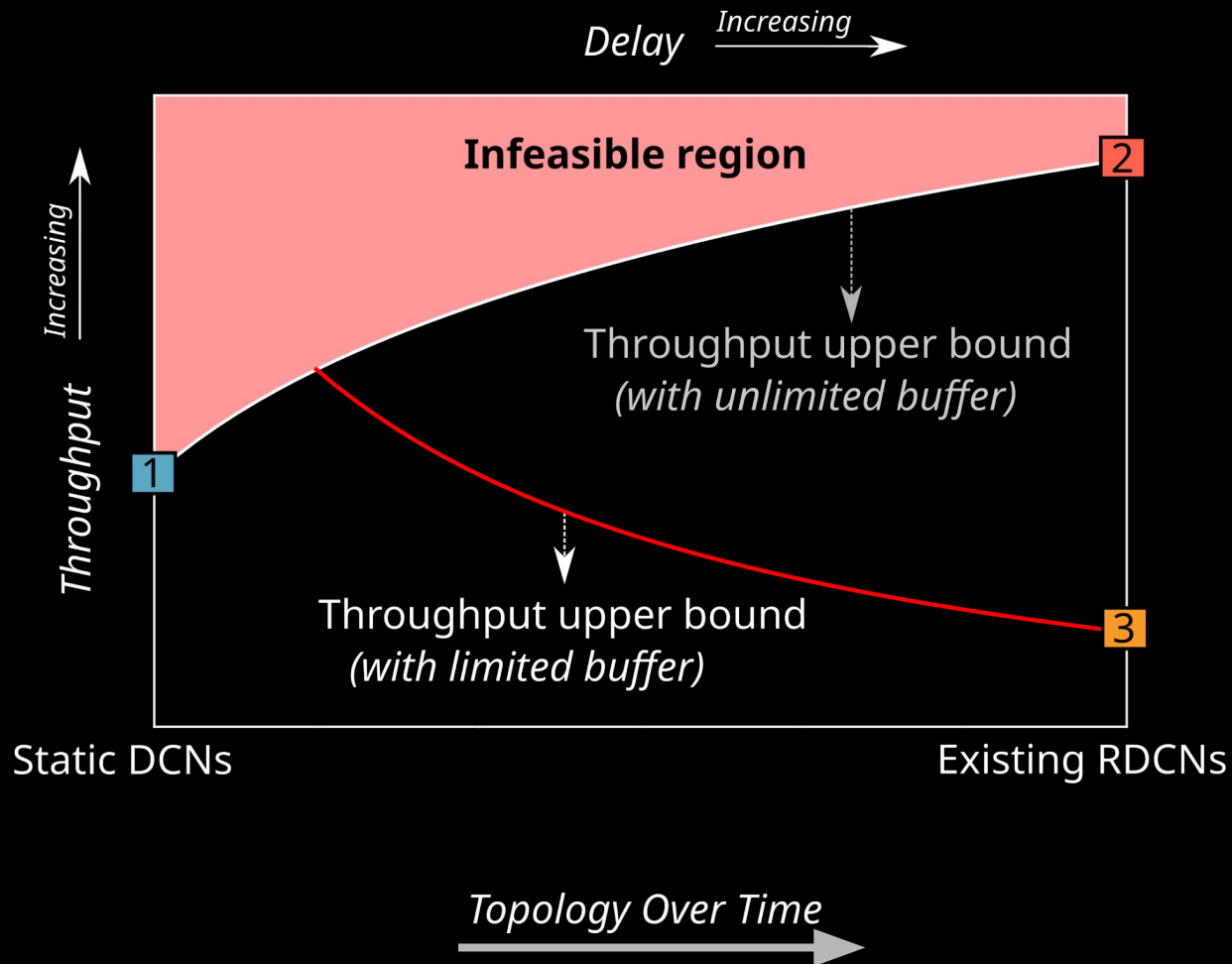


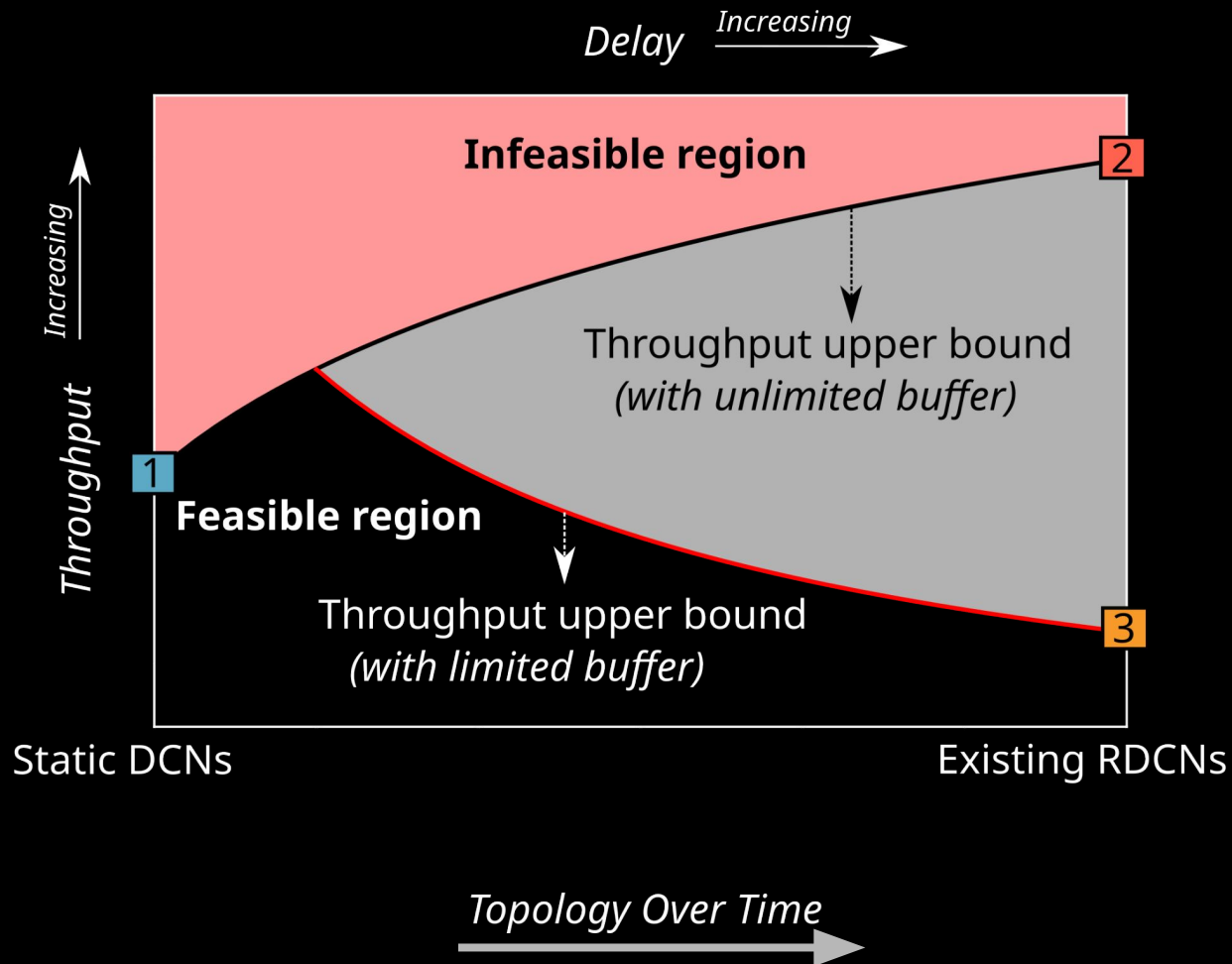


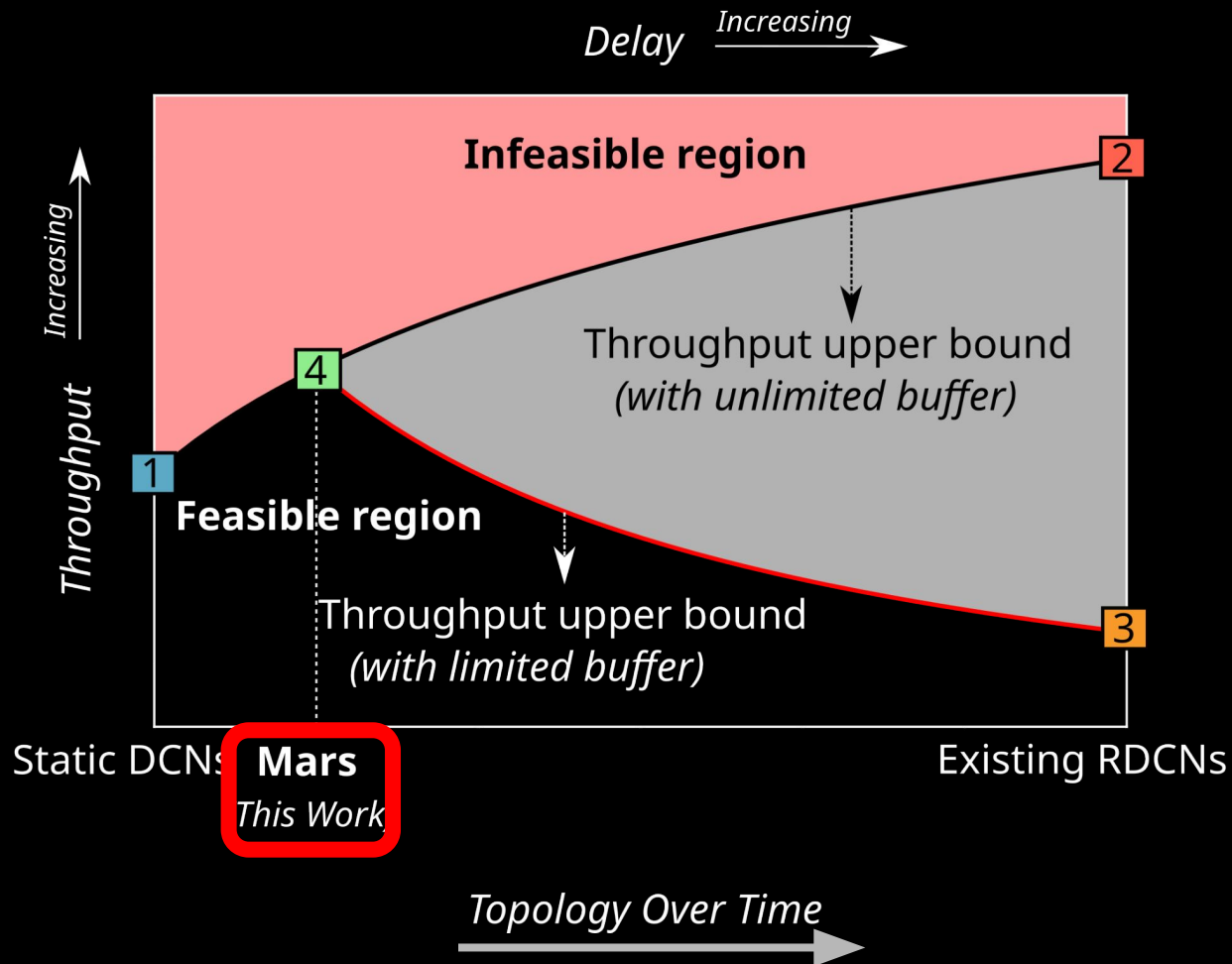


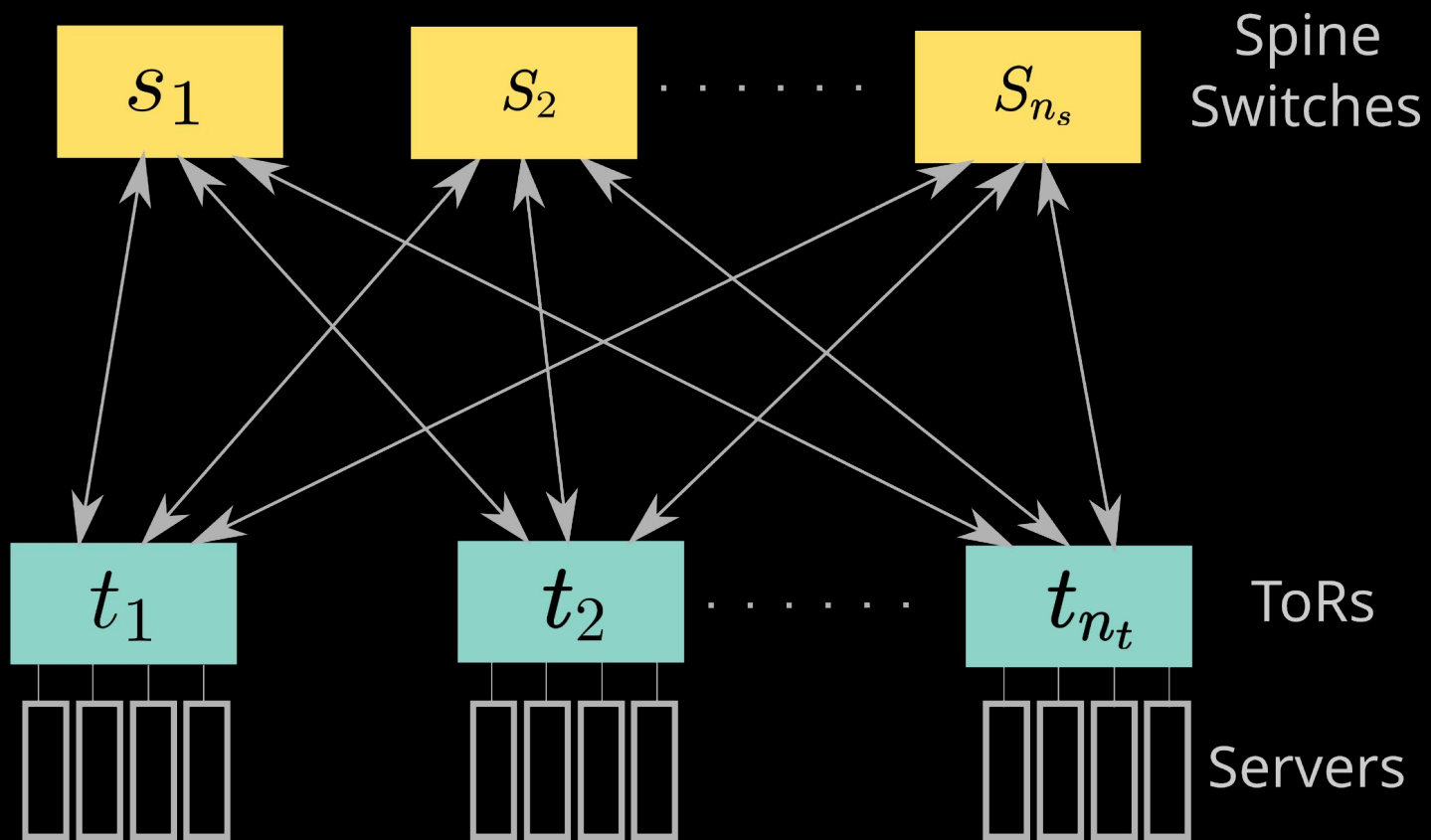








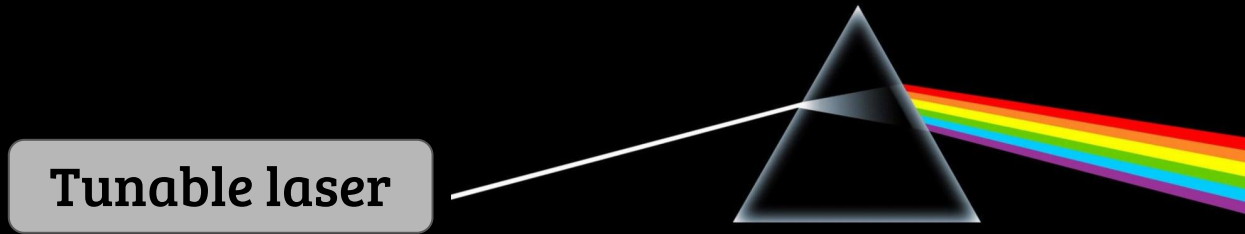




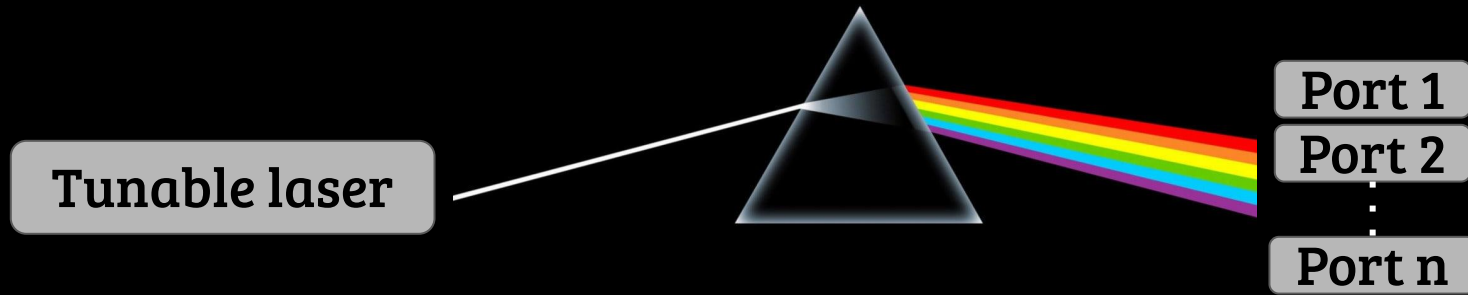
Emerging Technologies - Optical Networks on the Rise



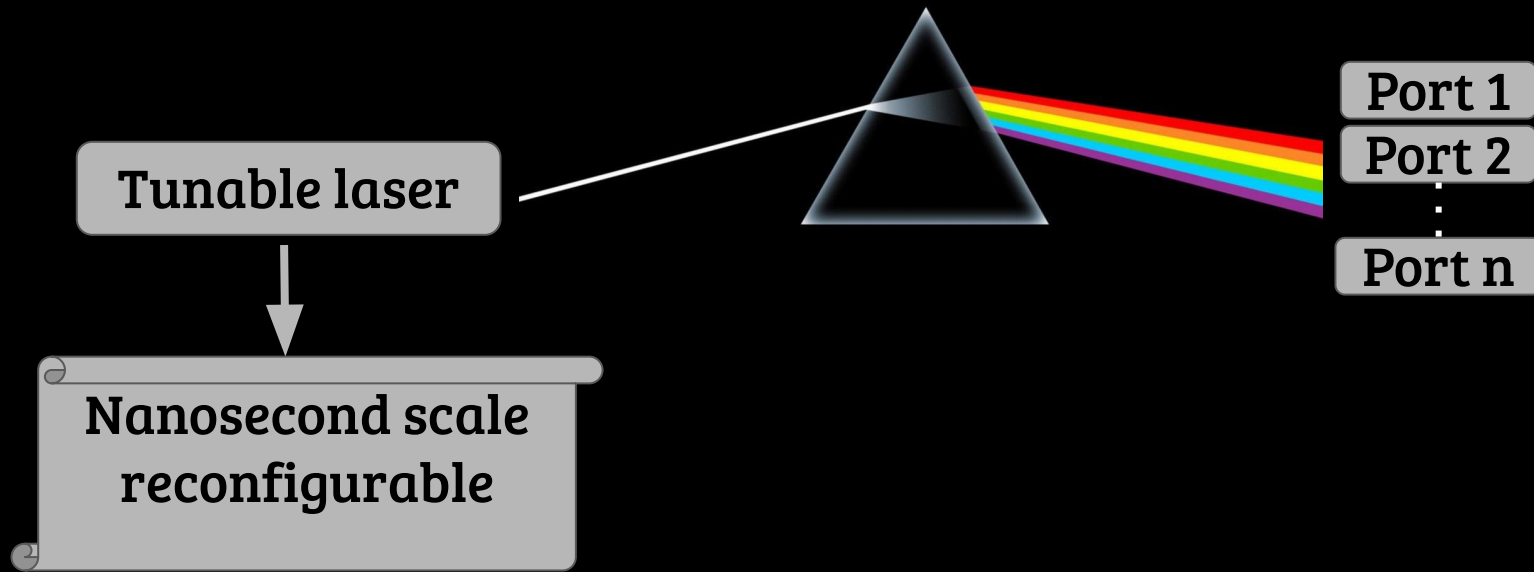
Emerging Technologies - Optical Networks on the Rise



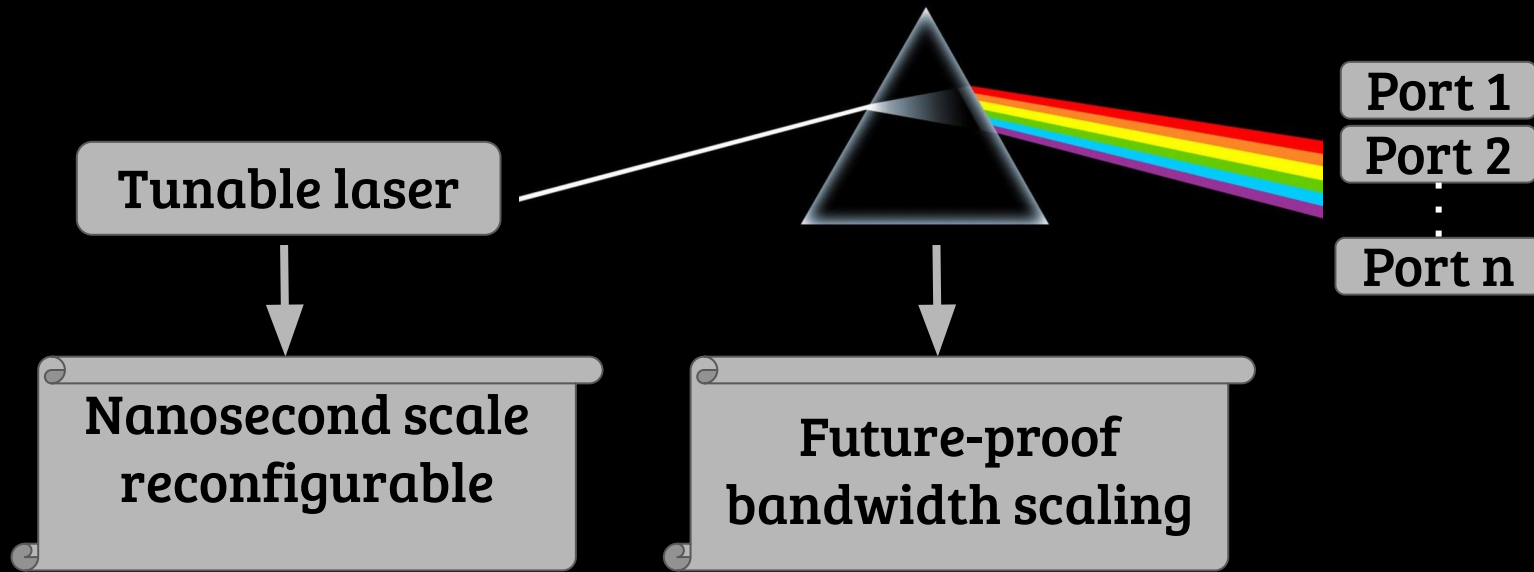
Emerging Technologies - Optical Networks on the Rise



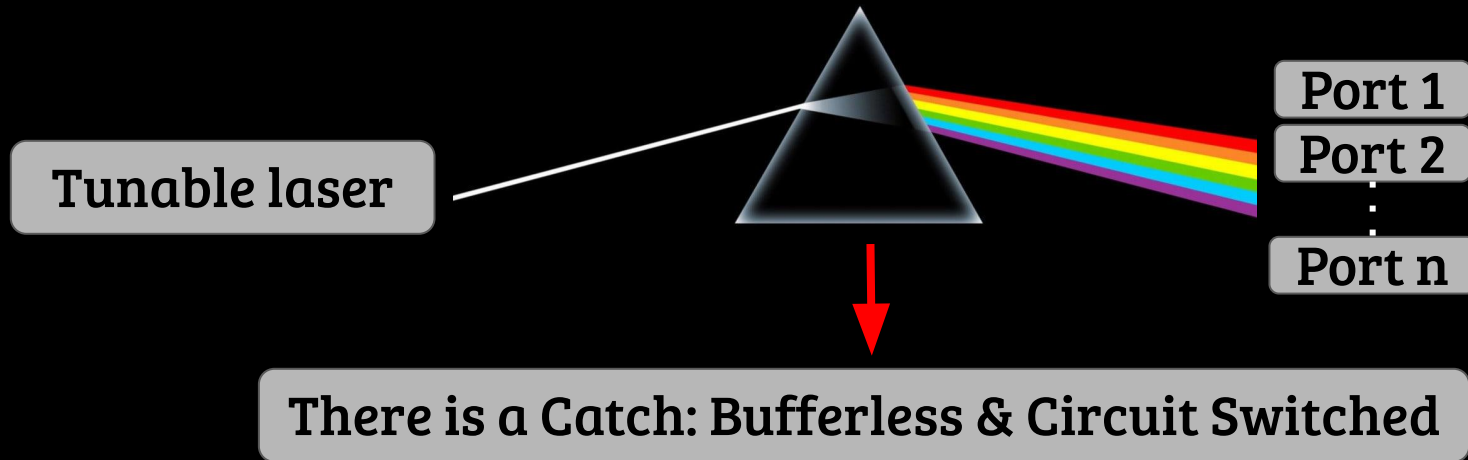
Emerging Technologies - Optical Networks on the Rise

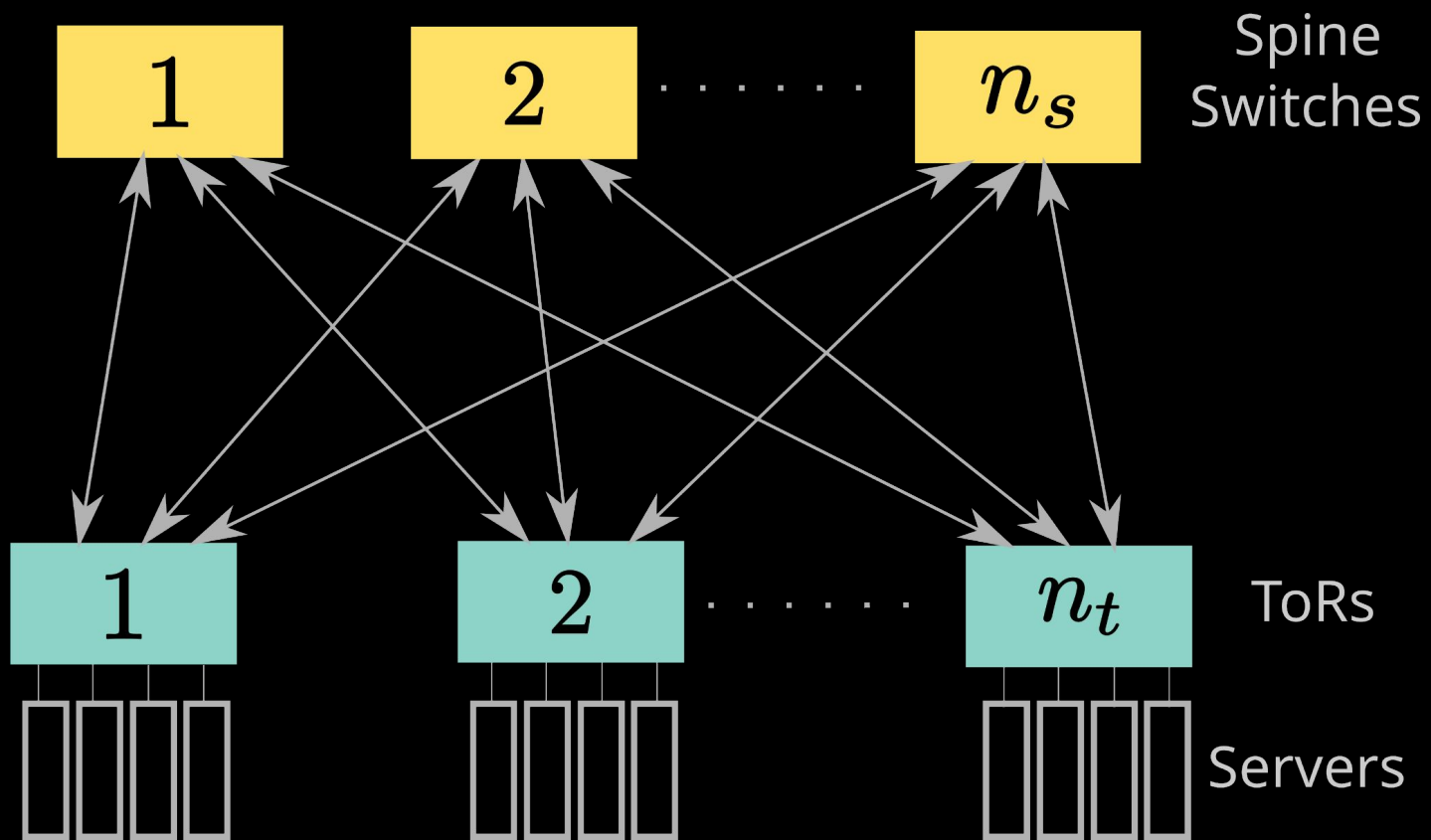


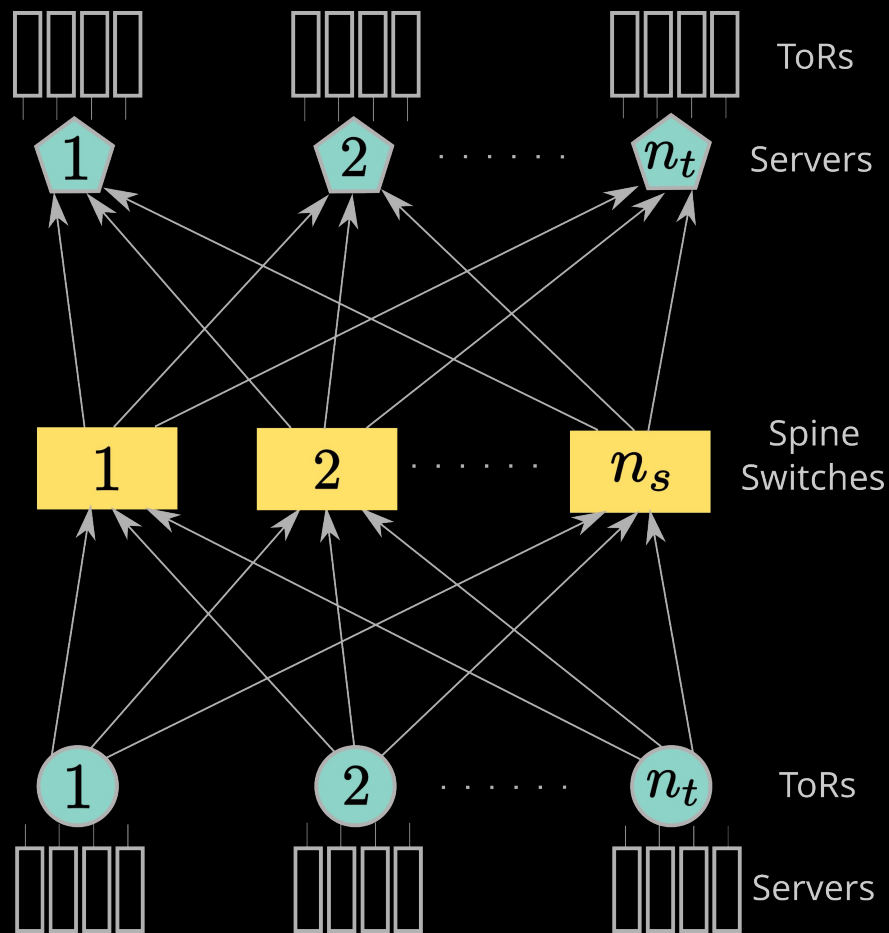
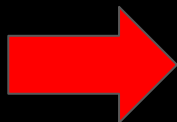
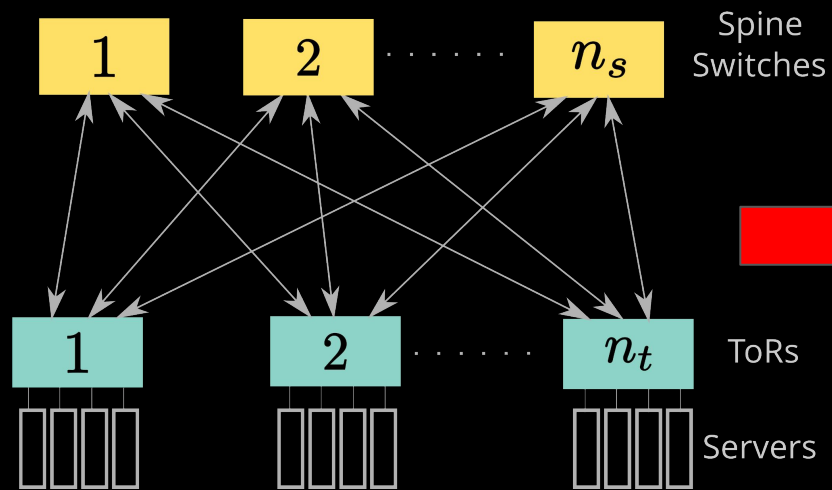
Emerging Technologies - Optical Networks on the Rise

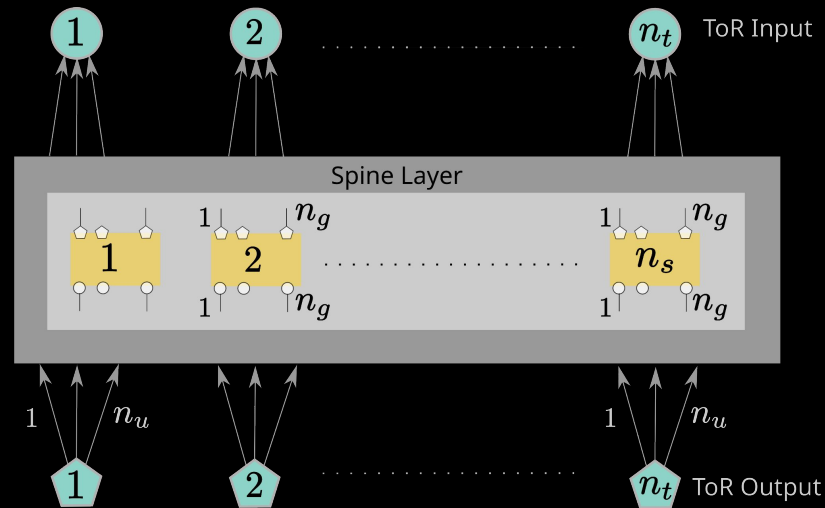
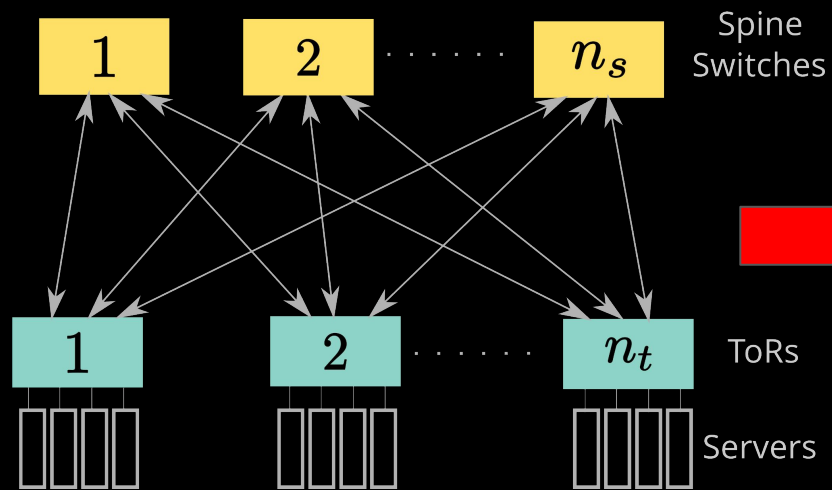


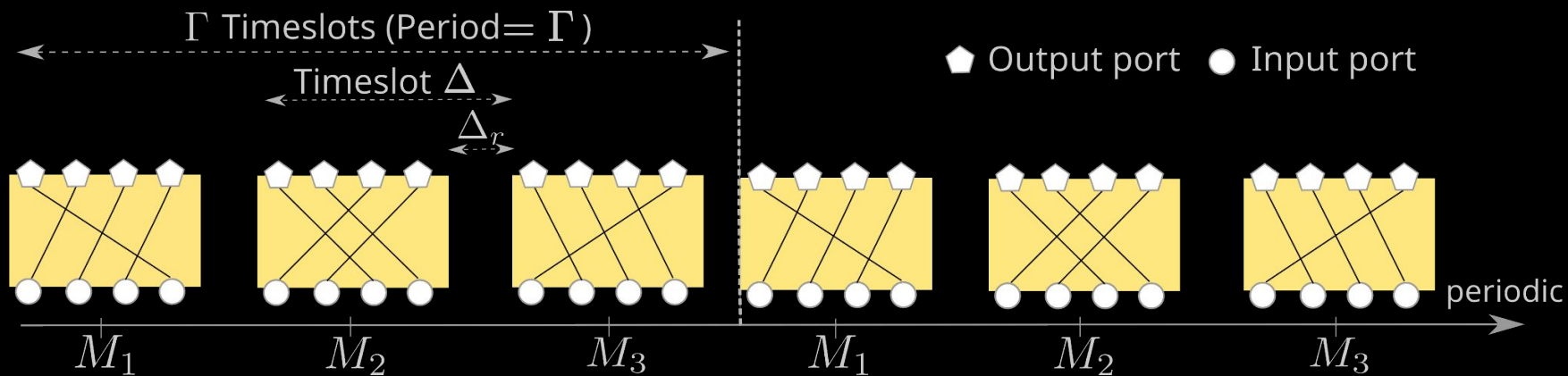
Emerging Technologies - Optical Networks on the Rise

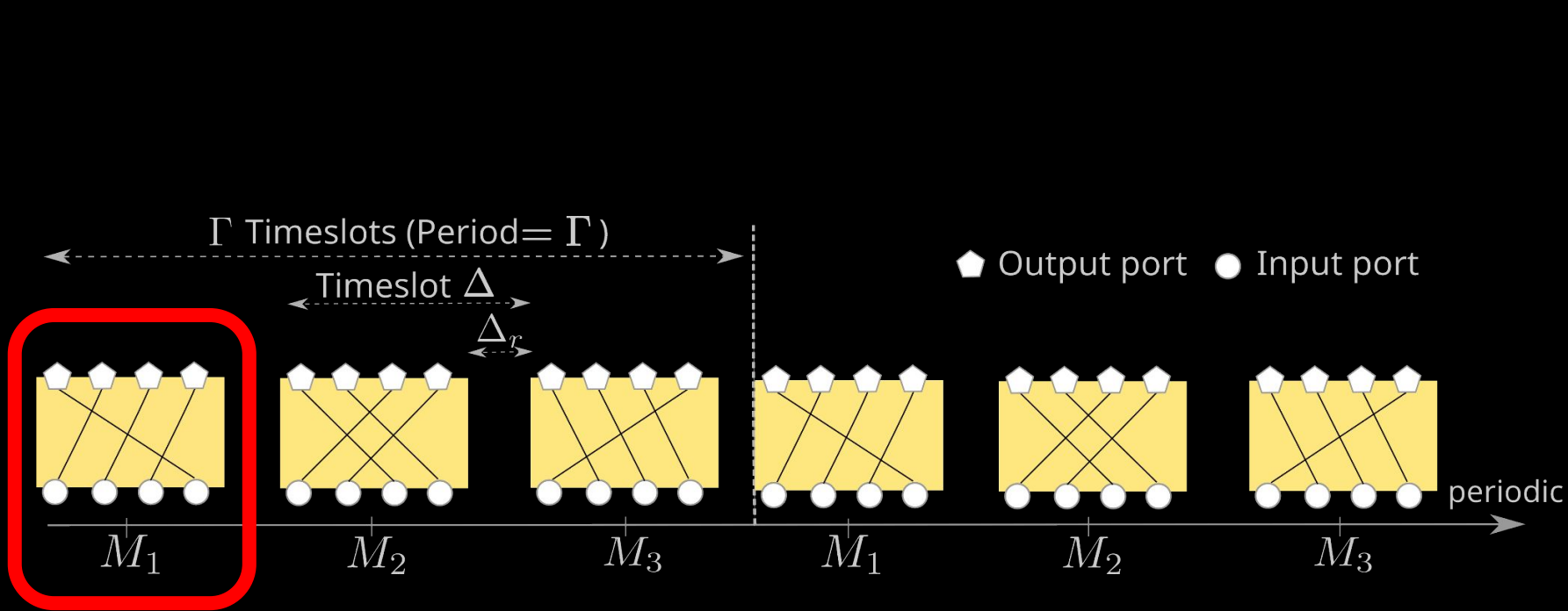


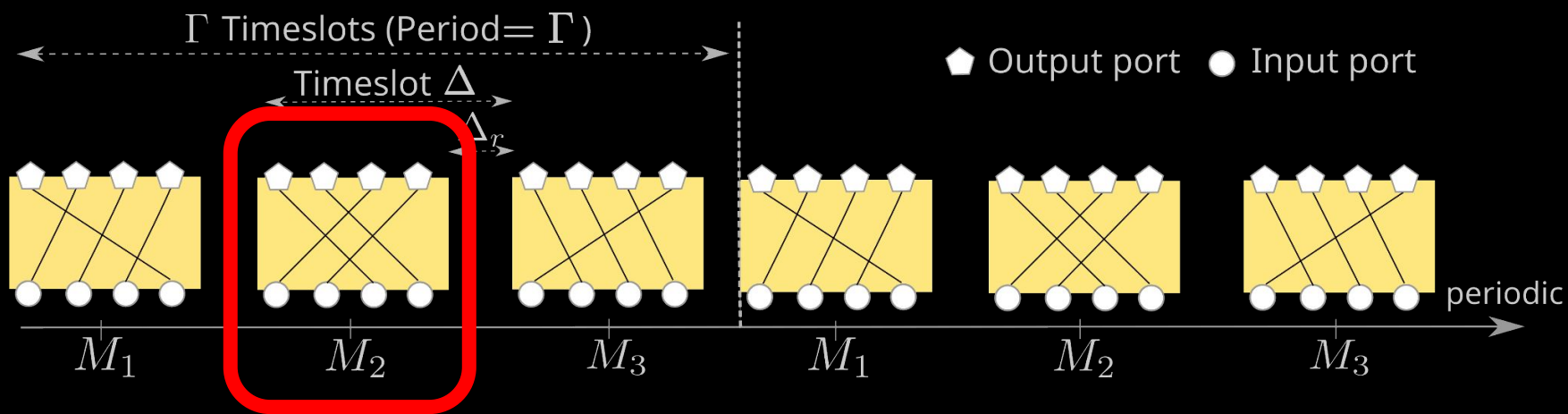


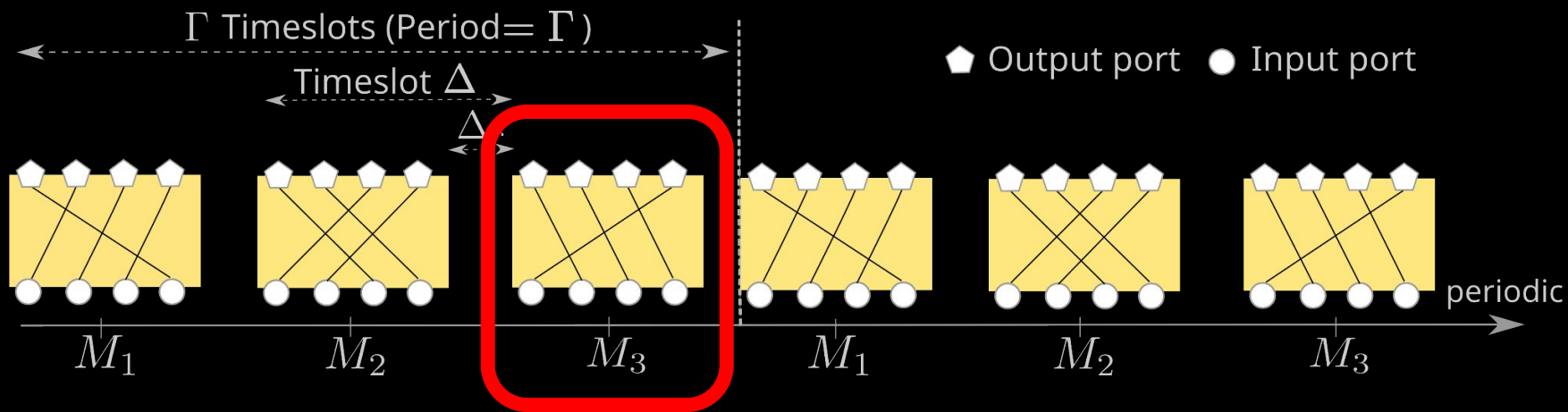


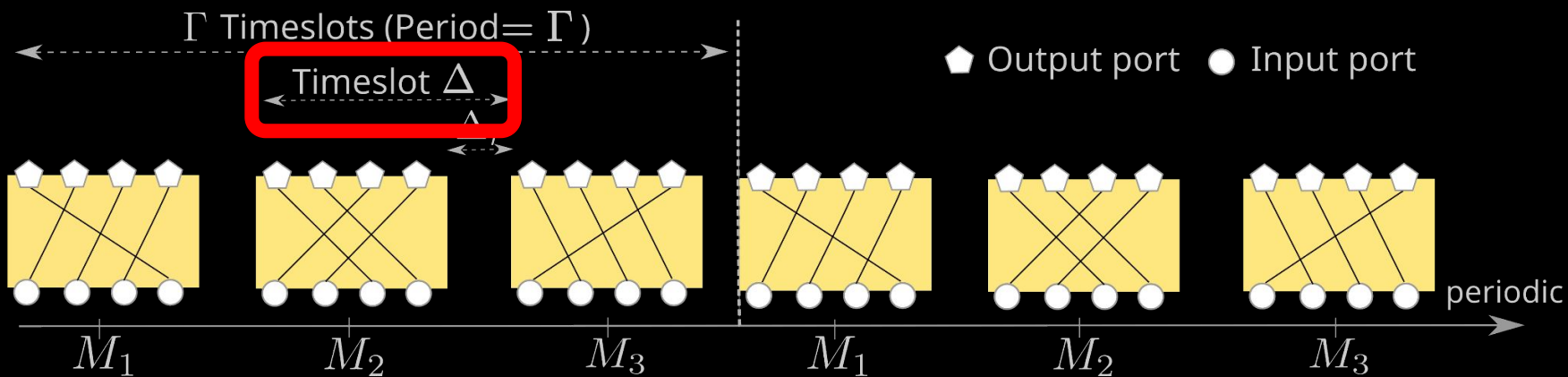


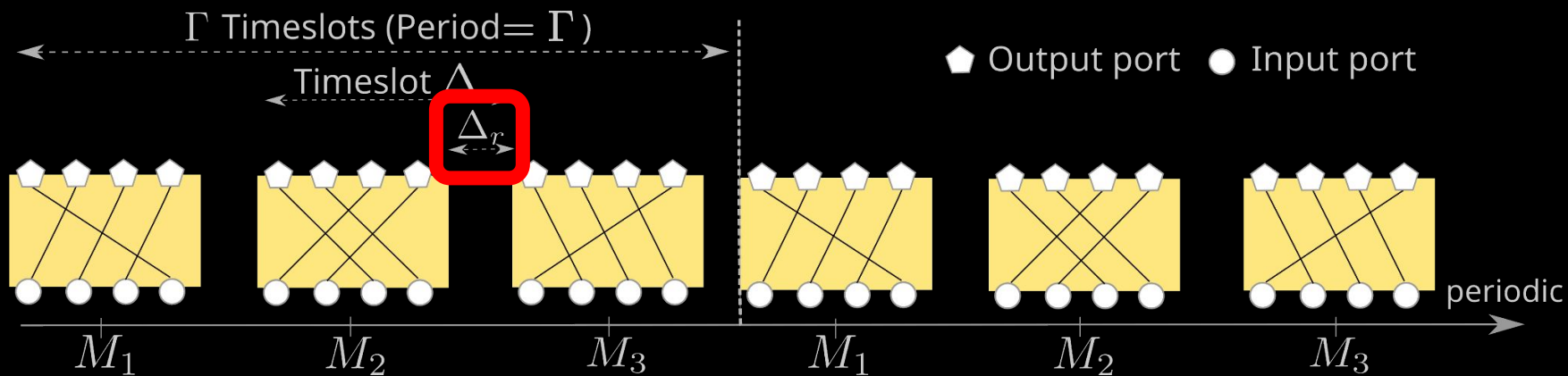


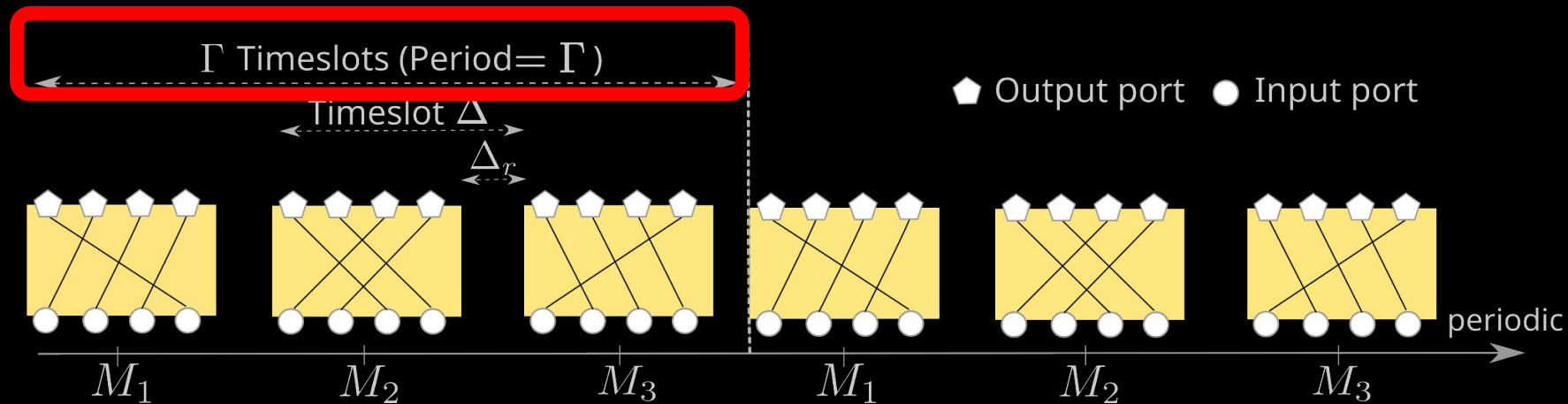


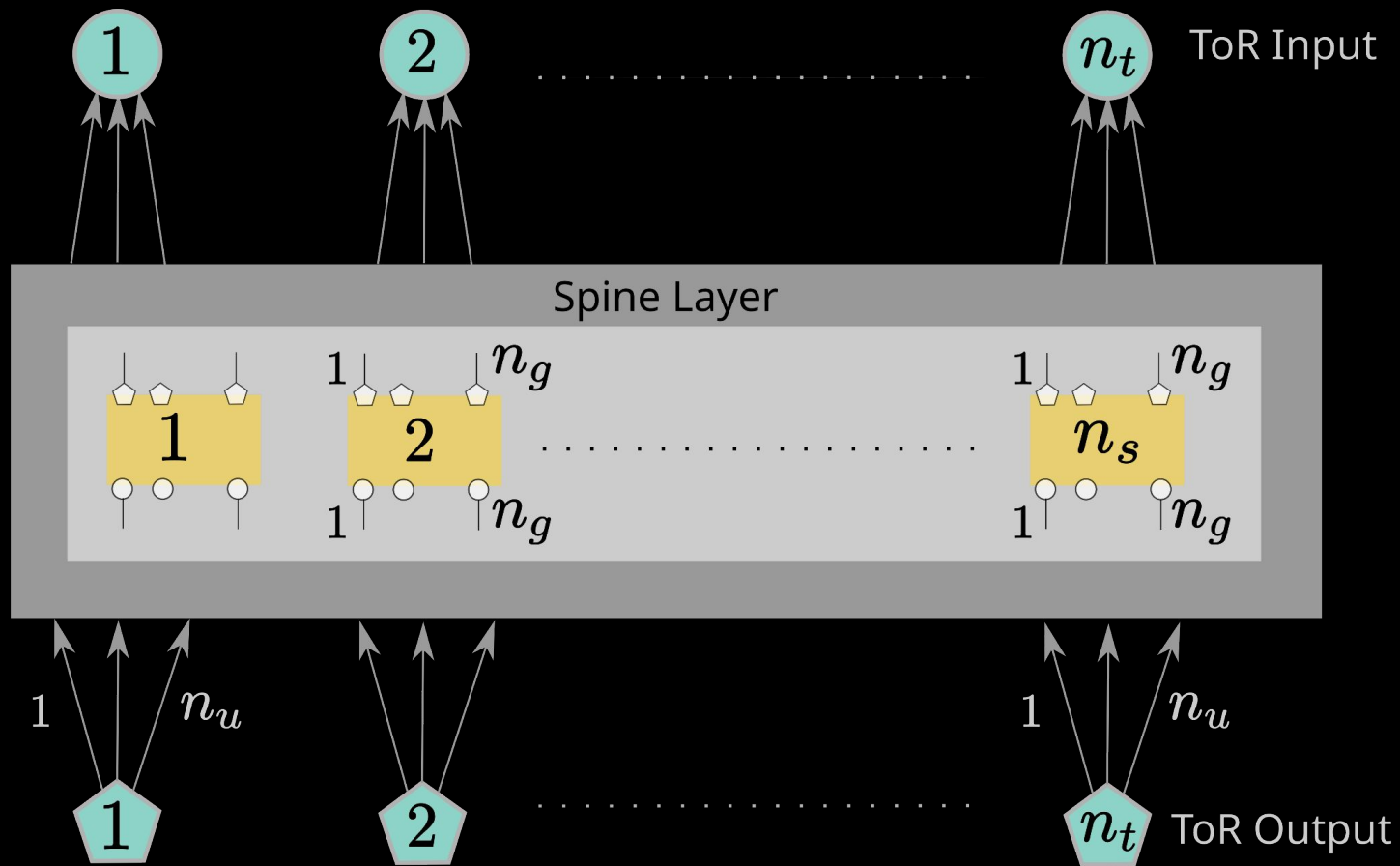




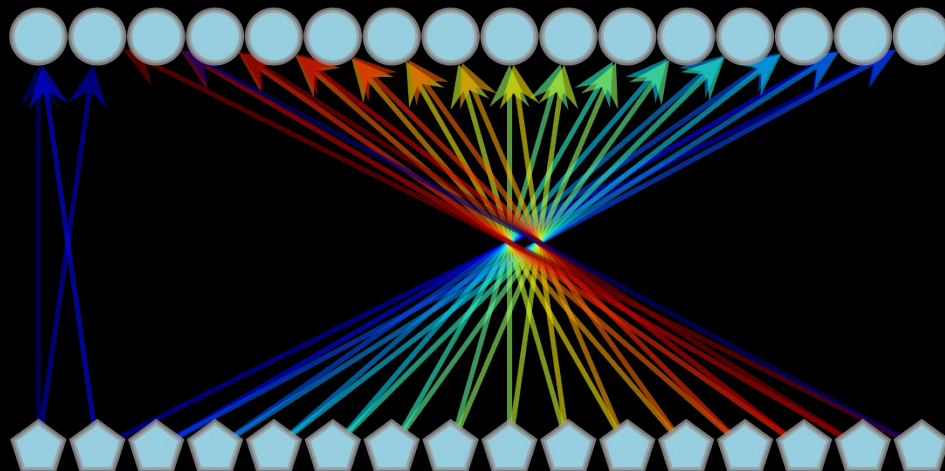




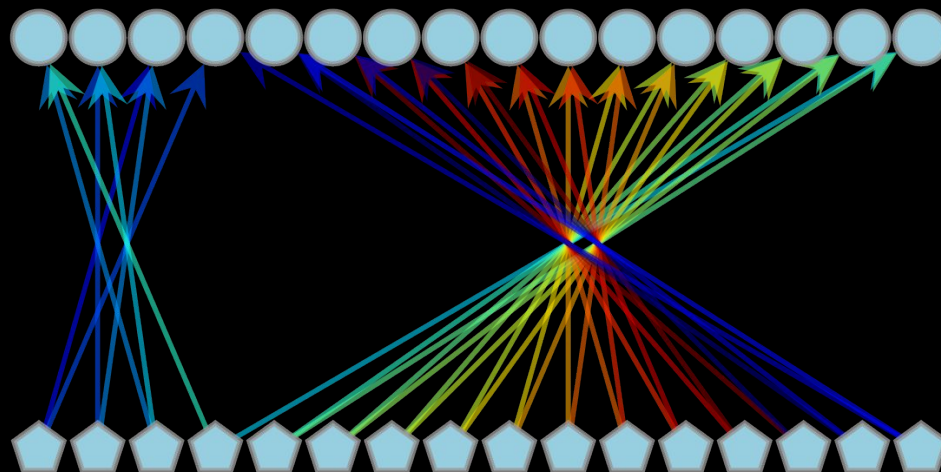




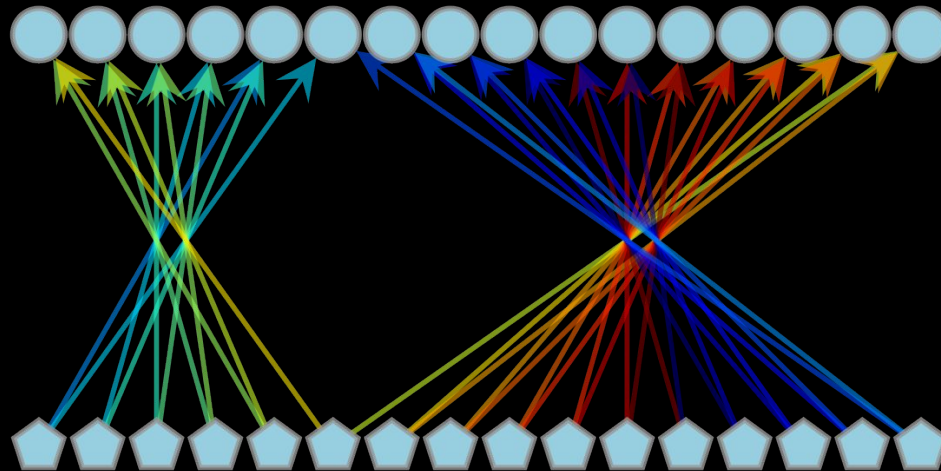
Timeslot 1



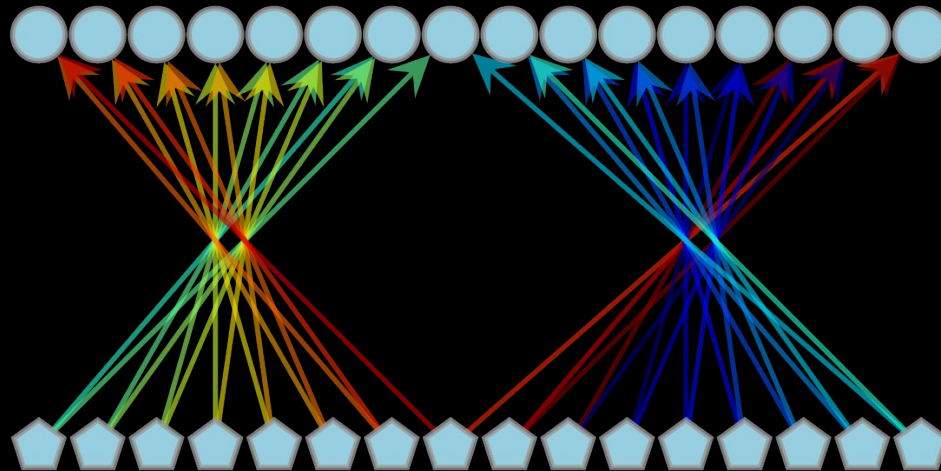
Timeslot 2



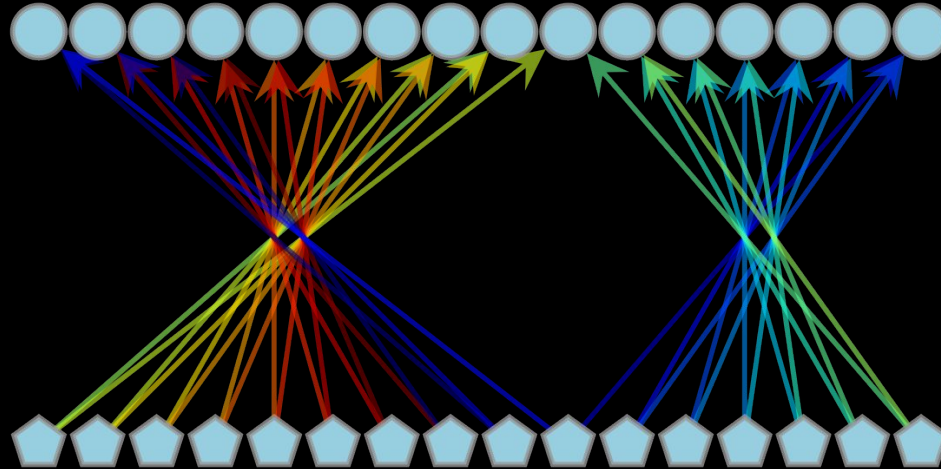
Timeslot 3



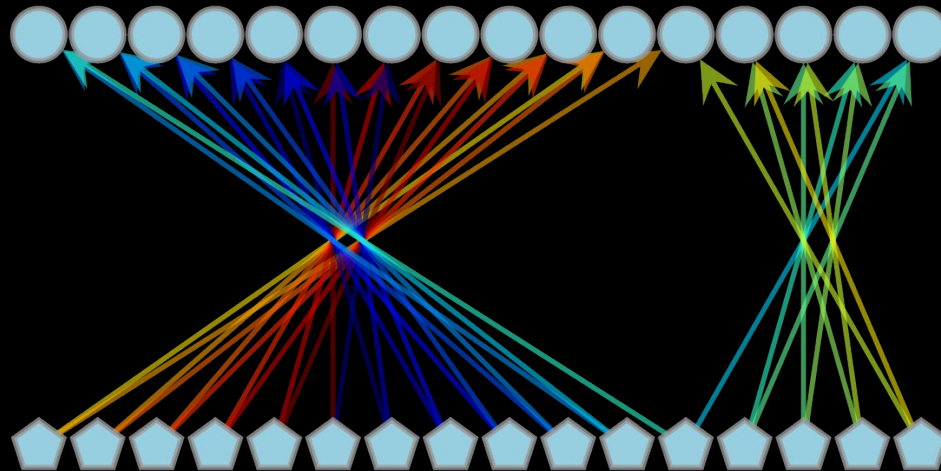
Timeslot 4



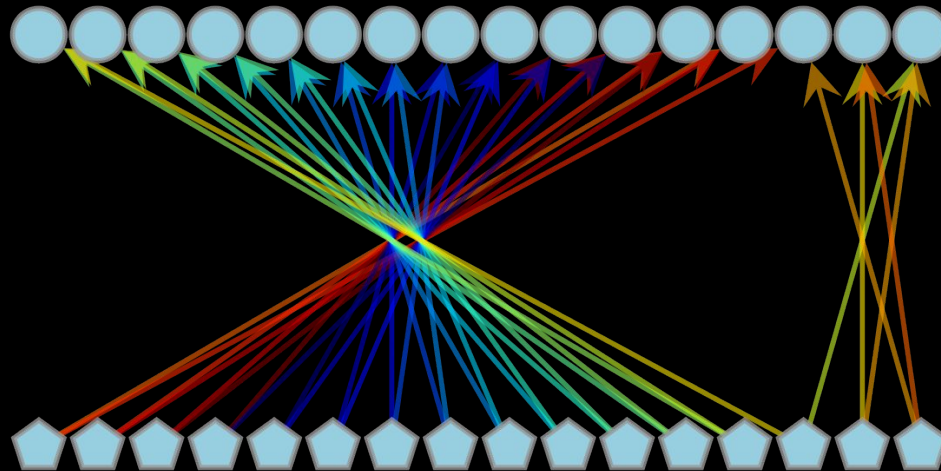
Timeslot 5



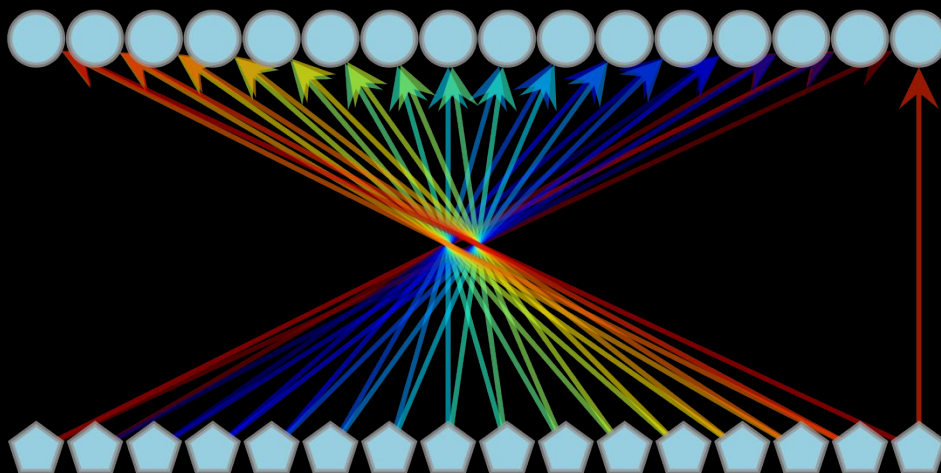
Timeslot 6



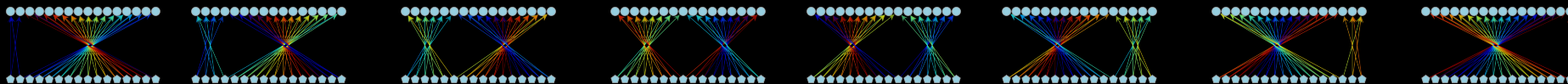
Timeslot 7



Timeslot 8



Periodic Graph



Optimal Topology

Input: Demand Matrix $\mathcal{M}$

Number of nodes n

Degree bound d' in each timeslot

Output: Periodic graph

Maximize: Throughput

Throughput

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

Demand Matrix

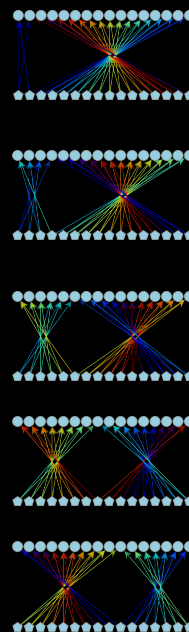
$\mathcal{M}$

$\times$

$\theta(\mathcal{M})$



Periodic Graph



t

Throughput

*Highest scaling factor such
that the scaled demand θ
($\mathcal{M}$) is feasible in the
periodic graph*

Throughput of the Periodic Graph

Input: Periodic Graph $\mathcal{G}$

Demand Matrix $\mathcal{M}$

Objective: Maximize $\theta(\mathcal{M})$

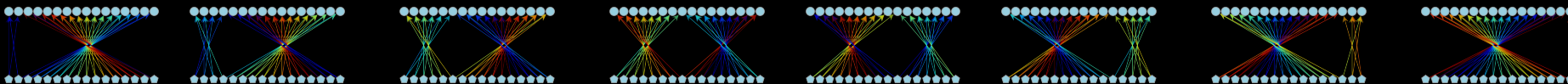
Output: $\theta(\mathcal{M})$ and a feasible *flow**

**subject to conservation, demand and capacity constraints*

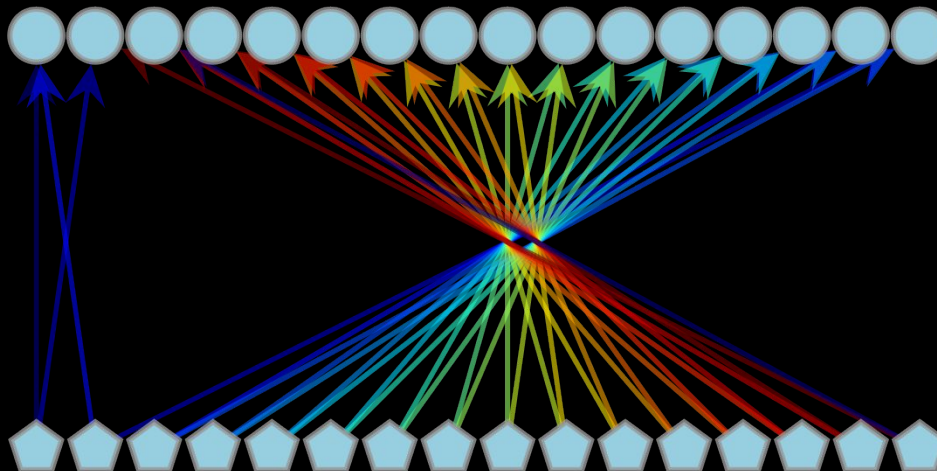
Theorem 1: Periodic Graph $\longleftrightarrow$ Static Graph

- The periodic graph has the **same throughput** as that of a static graph it emulates - *Static Emulated Graph*

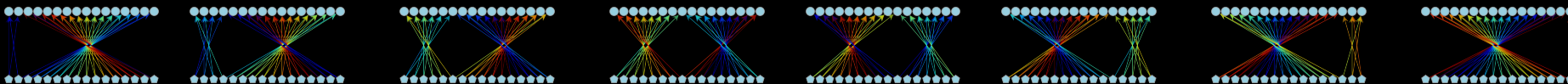
Periodic Graph



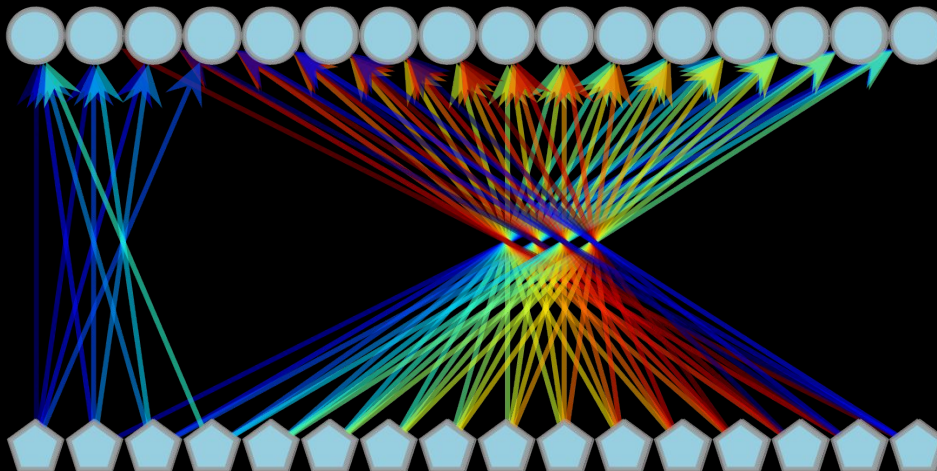
Static Emulated Graph



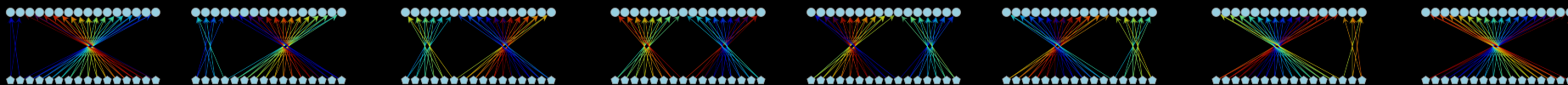
Periodic Graph



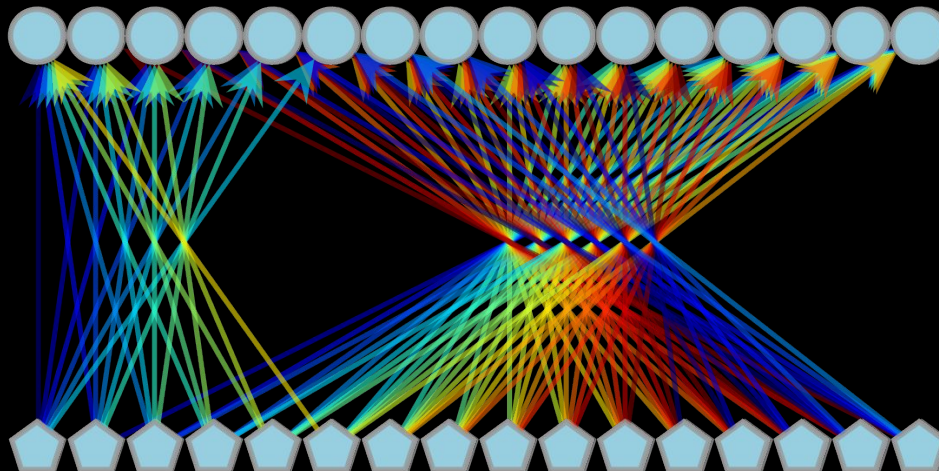
Static Emulated Graph



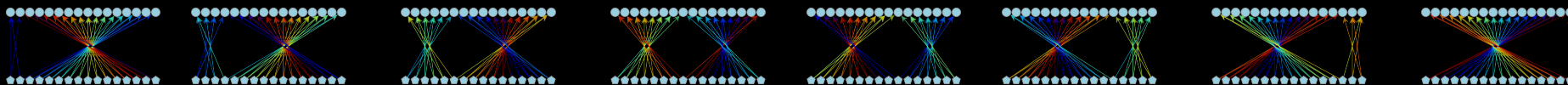
Periodic Graph



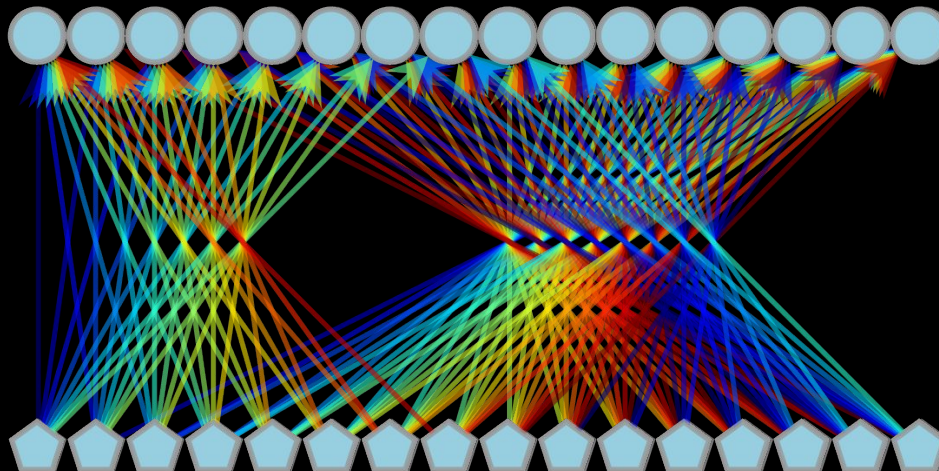
Static Emulated Graph



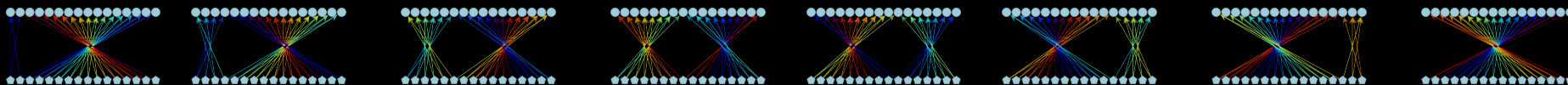
Periodic Graph



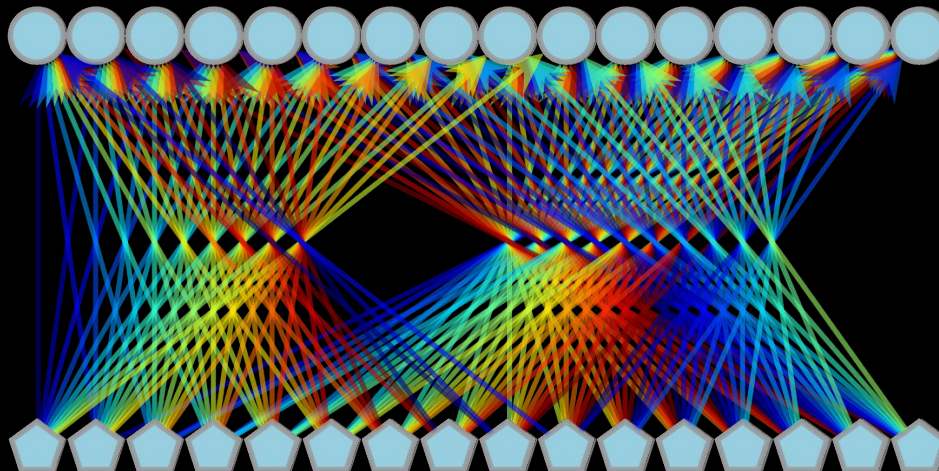
Static Emulated Graph



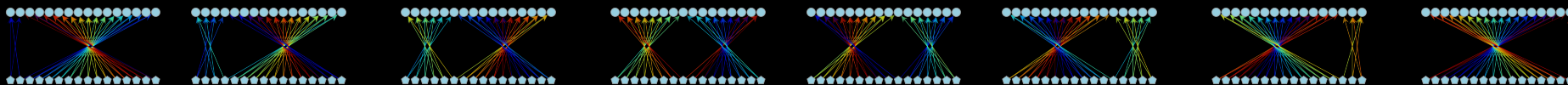
Periodic Graph



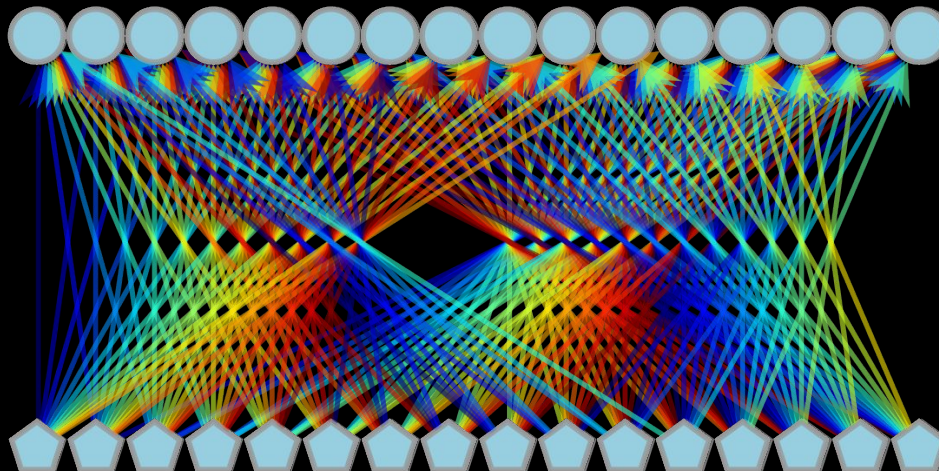
Static Emulated Graph



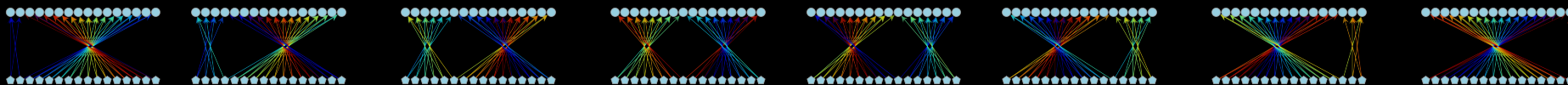
Periodic Graph



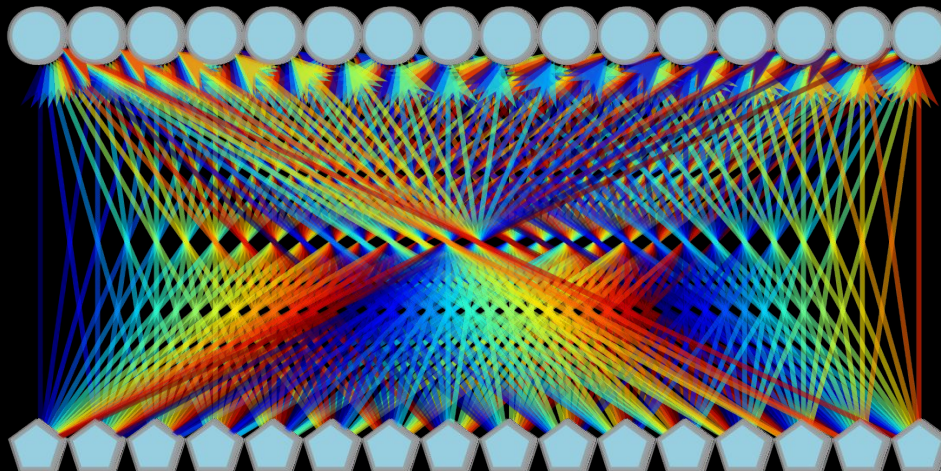
Static Emulated Graph



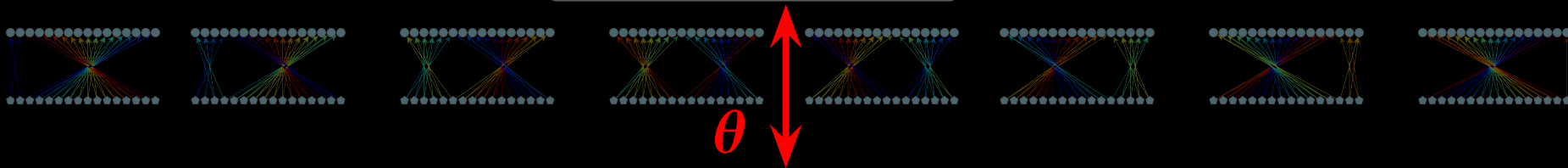
Periodic Graph



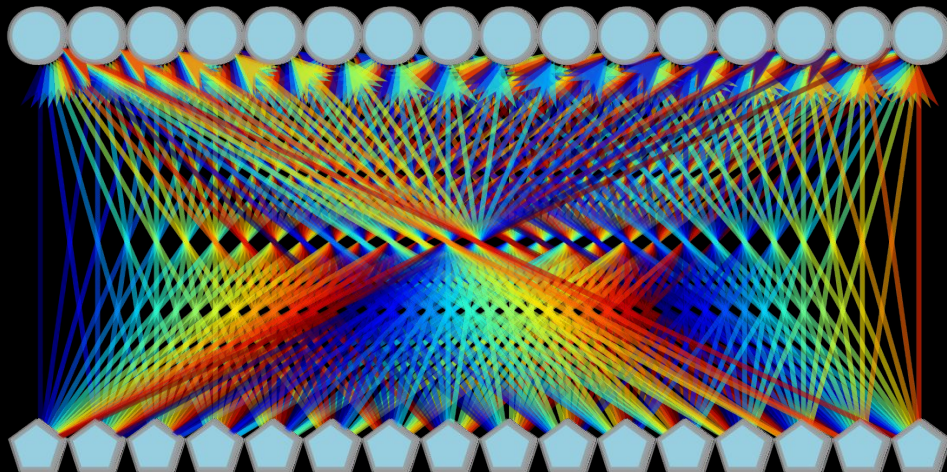
Static Emulated Graph



Periodic Graph



Static Emulated Graph



Throughput of the Periodic Graph

Input: Periodic Graph $\mathcal{G}$

Demand Matrix $\mathcal{M}$

Objective: Maximize $\theta(\mathcal{M})$

Output: $\theta(\mathcal{M})$ and a feasible *flow**

**subject to conservation, demand and capacity constraints*

Throughput of the Periodic Graph

Input: ~~Periodic Graph $\mathcal{G}$~~ Static Emulated Graph G

Demand Matrix $\mathcal{M}$

Objective: Maximize $\theta(\mathcal{M})$

Output: $\theta(\mathcal{M})$ and a feasible *flow**

**subject to conservation, demand and capacity constraints*



Theorem 2: Throughput Upper Bound

- Throughput is a function of *Average Route Length*

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

Theorem 2: Throughput Upper Bound

- Throughput is a function of *Average Route Length*

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

Theorem 2: Throughput Upper Bound

- Throughput is a function of *Average Route Length*

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

Theorem 2: Throughput Upper Bound

- Throughput is a function of *Average Route Length*

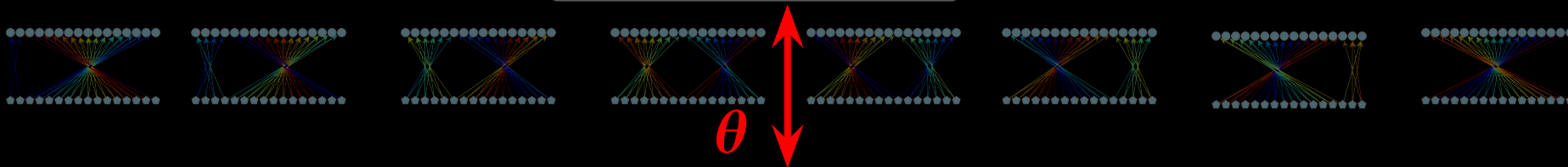
$$\theta(\mathcal{M}, F) \leq \frac{\hat{C}}{M \text{ ARL}(\mathcal{M}, F)}$$

Theorem 2: Throughput Upper Bound

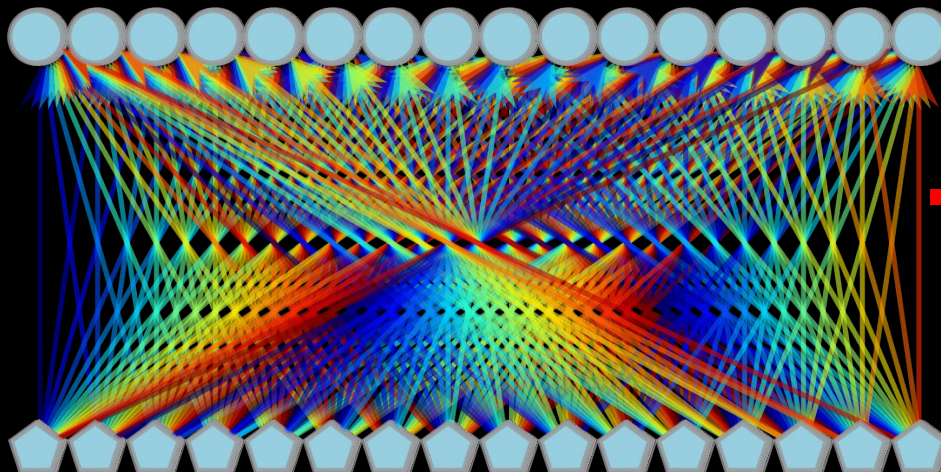
- Throughput is a function of *Average Route Length*

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

Periodic Graph

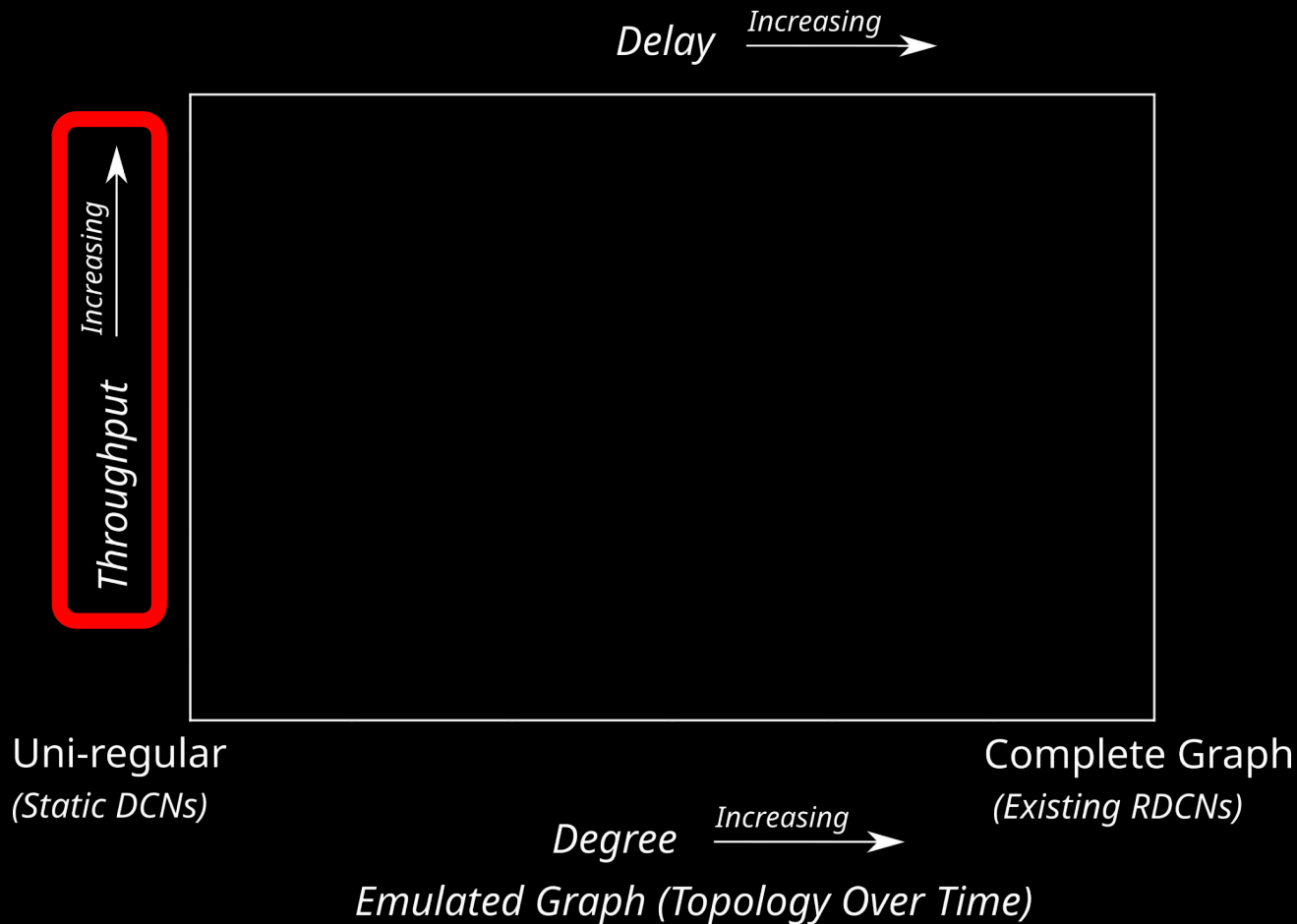


Static Emulated Graph



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

Capacity
Demand x ARL



Theorem 3: Delay

- Delay bound is a function of:
 - Degree d of the emulated graph
 - Duration of the period $\Gamma \cdot \Delta$
 - Throughput θ

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

Theorem 3: Delay

- Delay bound is a function of:
 - Degree d of the emulated graph
 - Duration of the period $\Gamma \cdot \Delta$
 - Throughput θ

$$\boxed{L_{max}} \geq \text{ARD}(\mathcal{M}, \mathcal{F}) = \text{ARL}(\mathcal{M}, \mathcal{F}) \cdot \Gamma \cdot \Delta$$
$$\geq \Omega \left(\frac{d \cdot \Delta}{n_u \cdot \theta(\mathcal{M}, \mathcal{F})} \right)$$

Theorem 3: Delay

- Delay bound is a function of:
 - Degree d of the emulated graph
 - Duration of the period $\Gamma \cdot \Delta$
 - Throughput θ

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

Theorem 3: Delay

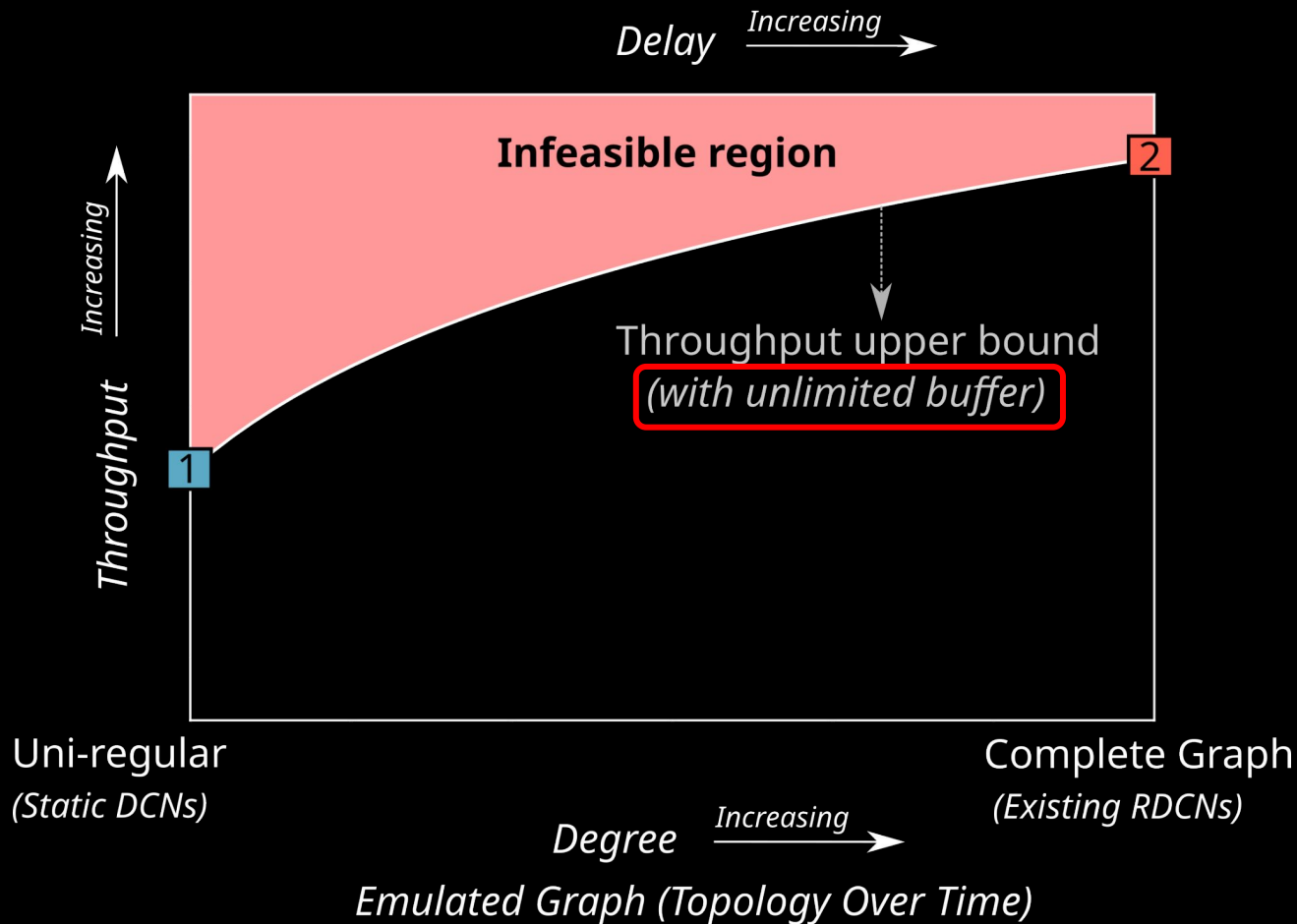
- Delay bound is a function of:
 - Degree d of the emulated graph
 - Duration of the period $\Gamma \cdot \Delta$
 - Throughput θ

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

Theorem 3: Delay

- Delay bound is a function of:
 - Degree d of the emulated graph
 - Duration of the period $\Gamma \cdot \Delta$
 - Throughput θ

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



Theorem 4: Buffer Requirements

- The required buffer is at least the *throughput delay product*

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

Theorem 4: Buffer Requirements

- The required buffer is at least the *throughput delay product*

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

Theorem 4: Buffer Requirements

- The required buffer is at least the *throughput delay product*

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

Theorem 4: Buffer Requirements

- The required buffer is at least the *throughput delay product*

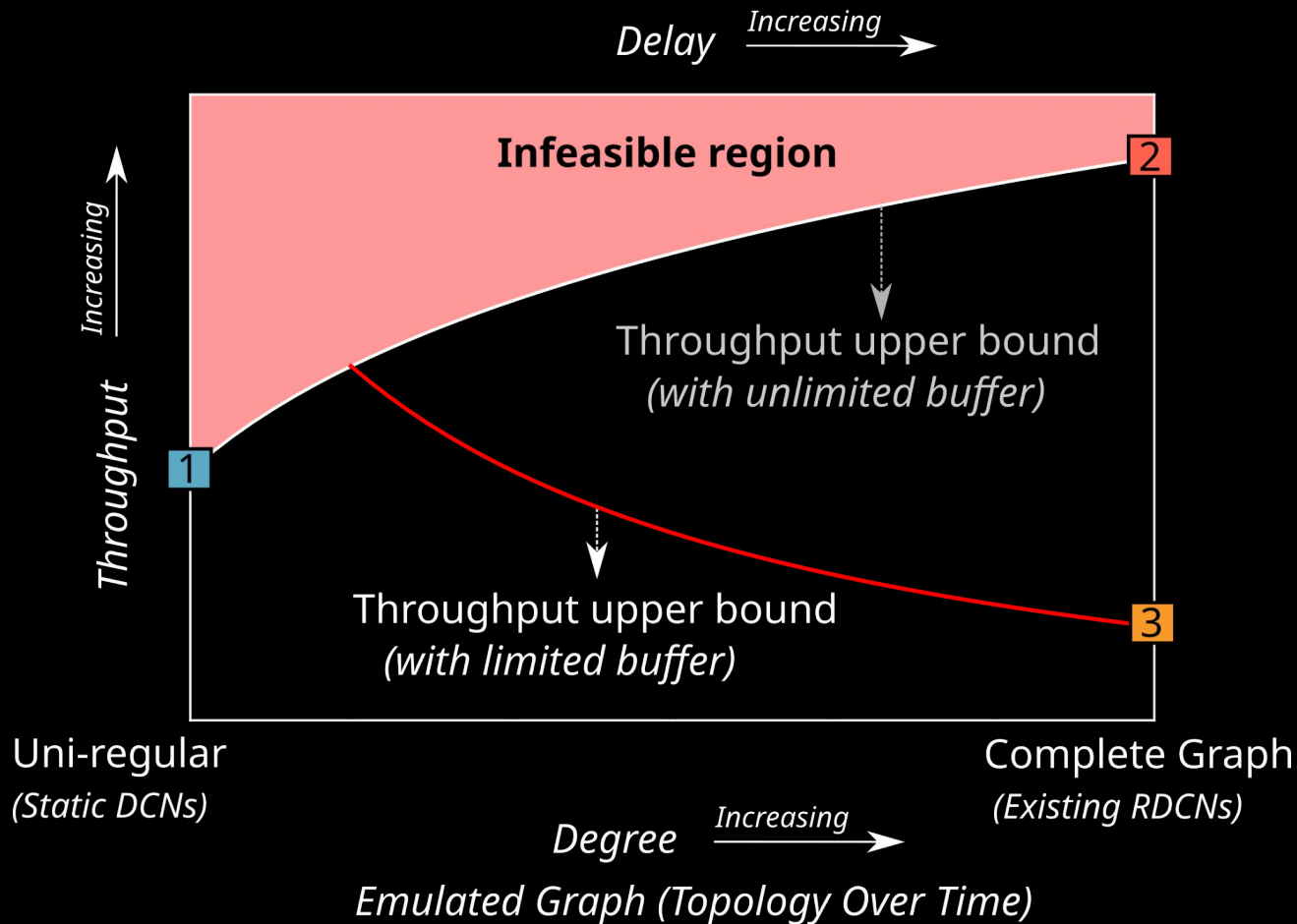
$$\hat{B} \geq (\theta(\mathcal{M}, \mathcal{F}) \cdot M) \cdot \text{ARD}(\mathcal{M}, \mathcal{F})$$

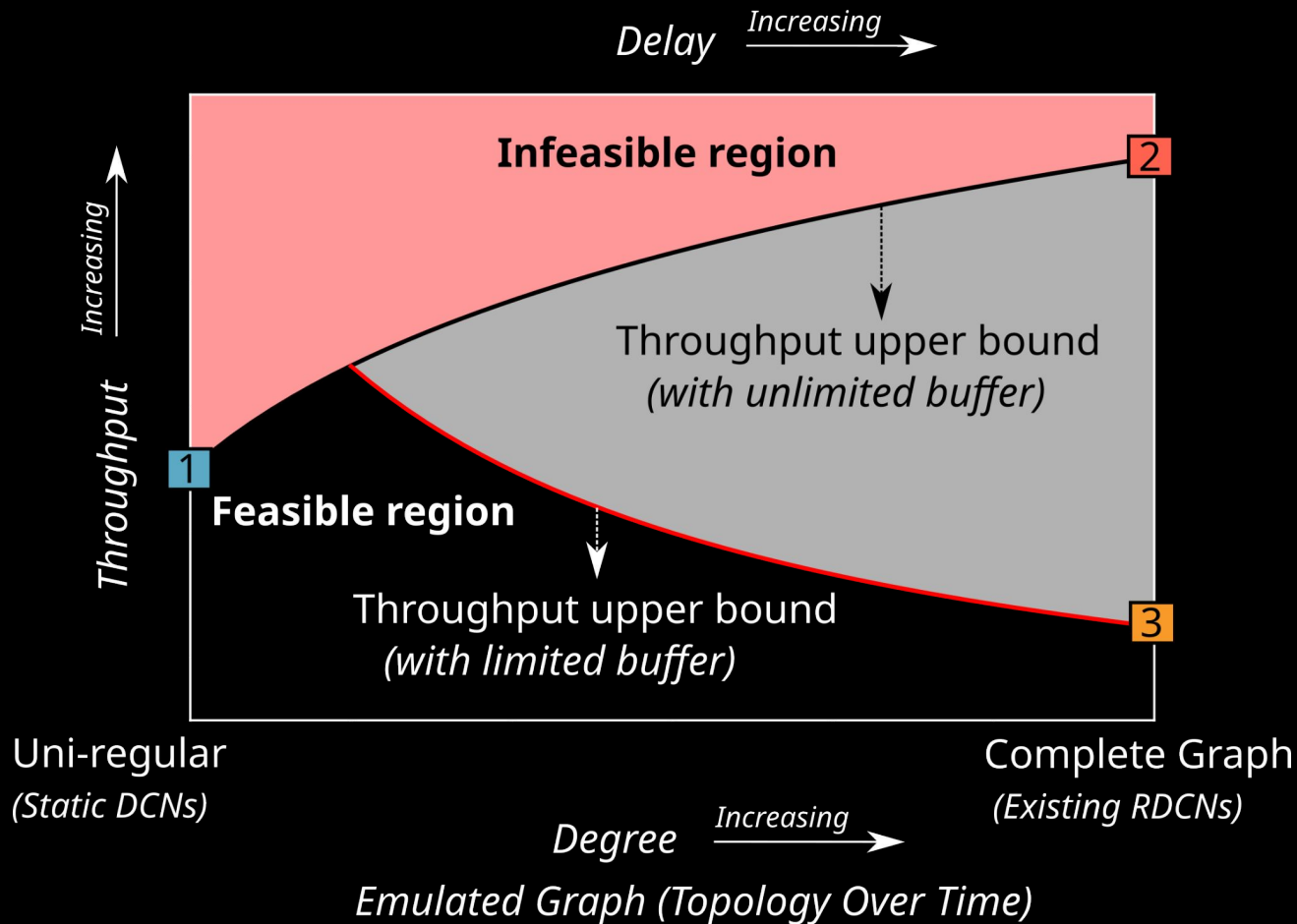
Theorem 4: Buffer Requirements

- The required buffer is at least the *throughput delay product*

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

Buffer > Bandwidth x Delay





Goals

- **Maximize Throughput**
- **Minimize Latency**
- **Minimize Buffer Requirements**

Optimal Oblivious Topology with Buffer Constraints

Input: ~~Periodic Graph~~ $\mathcal{G}$

Hose model demand matrix set

Available buffer size B at each node

Output: Degree d of the emulated graph $\longrightarrow$ Periodic graph

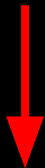
Objective: Maximize the worst-case throughput

Optimal Oblivious Topology with Buffer Constraints

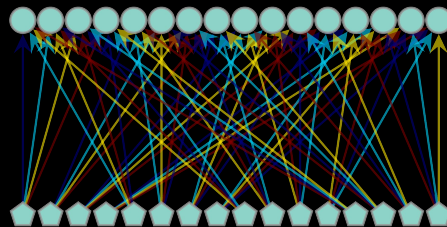
Output: $d = B / (c \cdot \Delta)$

Optimal Oblivious Topology with Buffer Constraints

Output: $d = B / (c \cdot \Delta)$

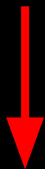


d-regular directed deBruijn graph



Optimal Oblivious Topology with Buffer Constraints

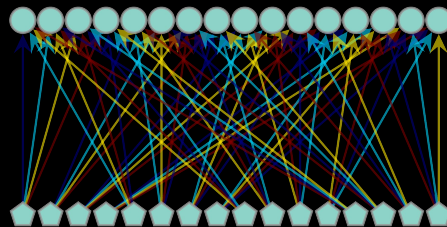
Output: $d = B / (c \cdot \Delta)$



d-regular directed deBruijn graph



Decomposition to d matchings



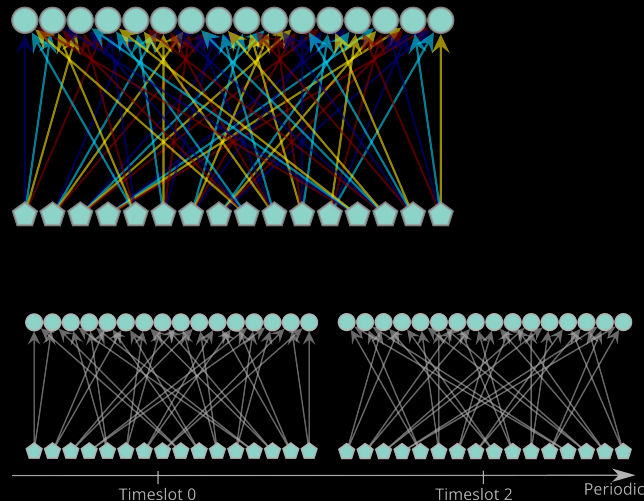
Optimal Oblivious Topology with Buffer Constraints

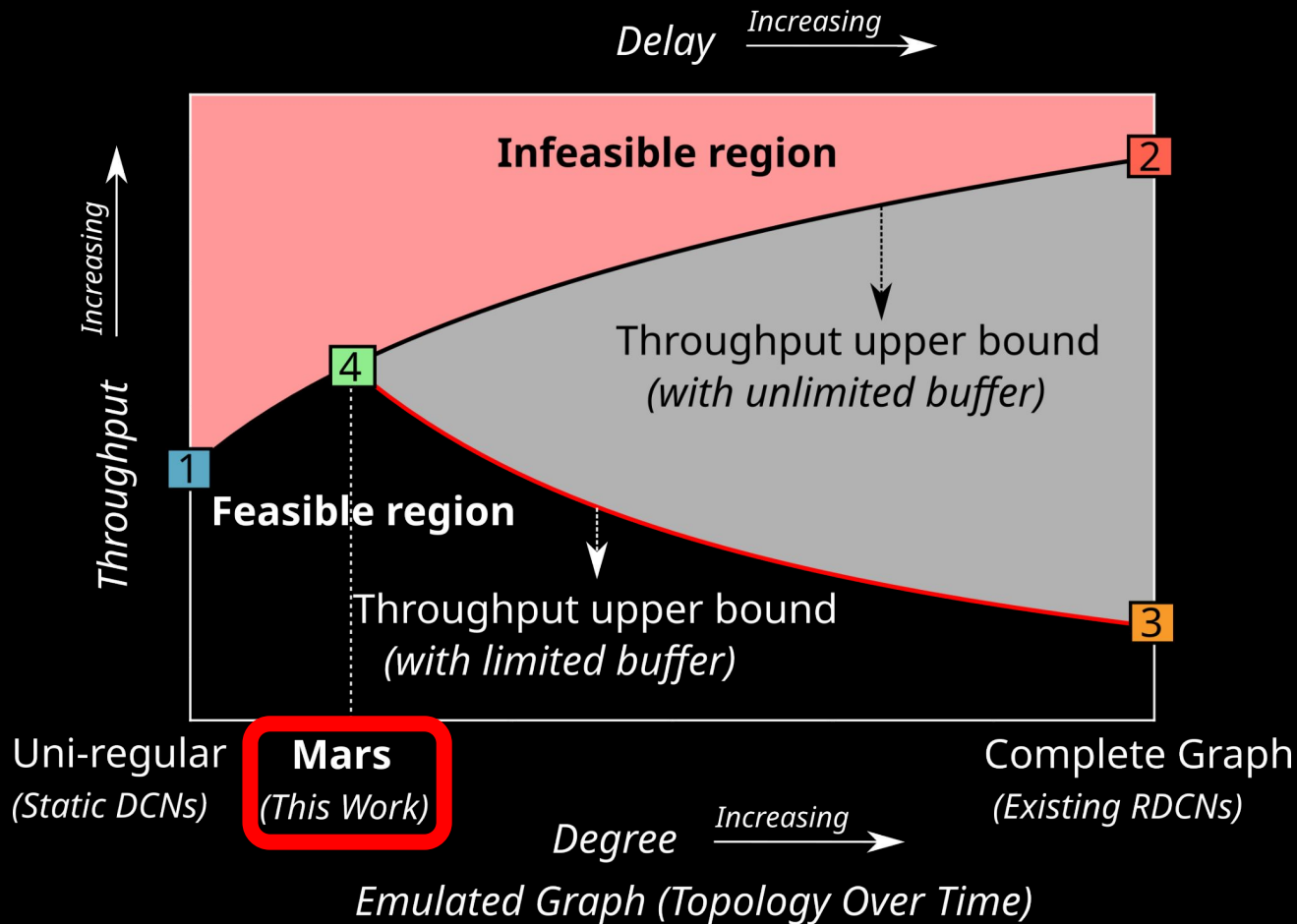
Output: $d = B / (c \cdot \Delta)$

d-regular directed deBruijn graph

Decomposition to d matchings

Periodic graph





Optimal Oblivious Topology Implications & Future Outlook

Output: $d = B / (c \cdot \Delta)$

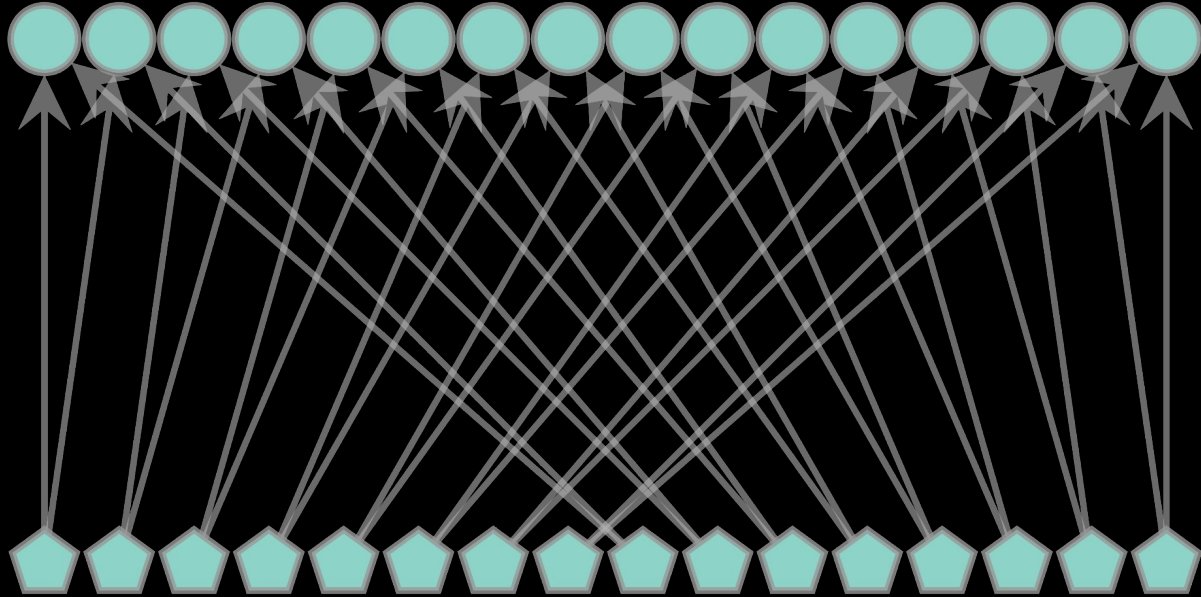
If the reconfiguration technology (Δ) remains same:

- Buffer sizes (B) $\uparrow$ must keep up with the increase in capacity (c) $\uparrow$

If Buffer sizes (B) do not keep up:

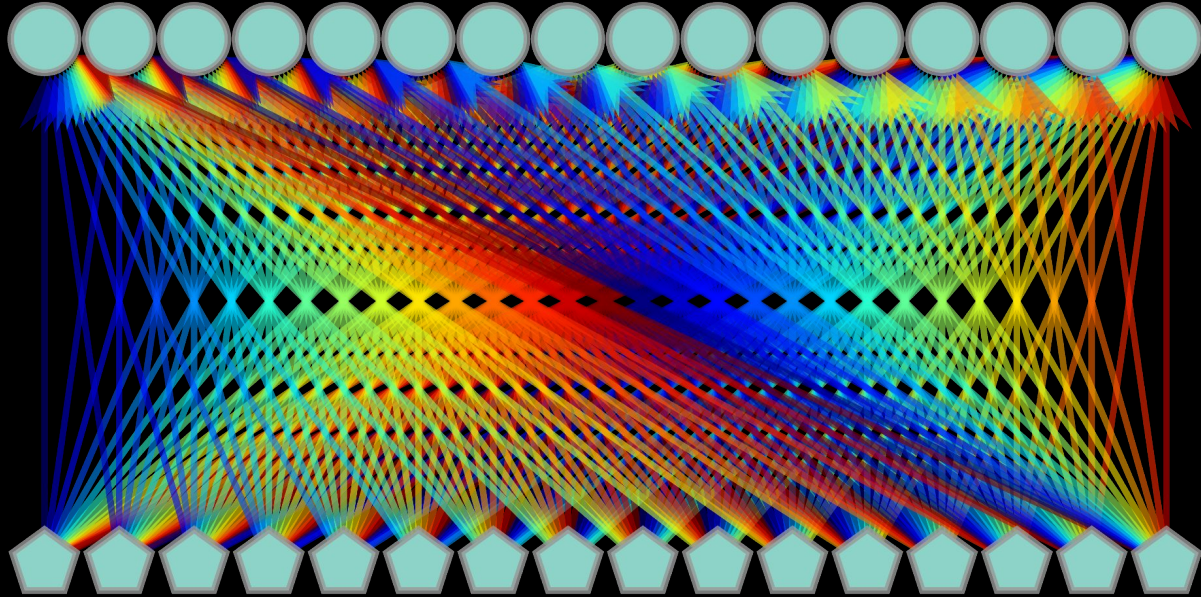
- Increase in capacity (c) $\uparrow$ must be accompanied by decrease in reconfiguration times (Δ) $\downarrow$
- If not, reducing the degree (d) $\downarrow$ of the emulated graph is inevitable to optimize throughput $\rightarrow$ eventually reaching the case of static topologies.

Static DCNs (*uni-regular*)



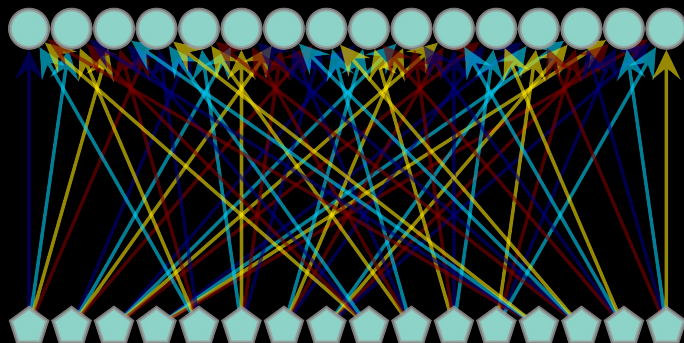
Low throughput but low delay and buffer requirements

Existing RDCN designs (*Emulating a complete graph*)

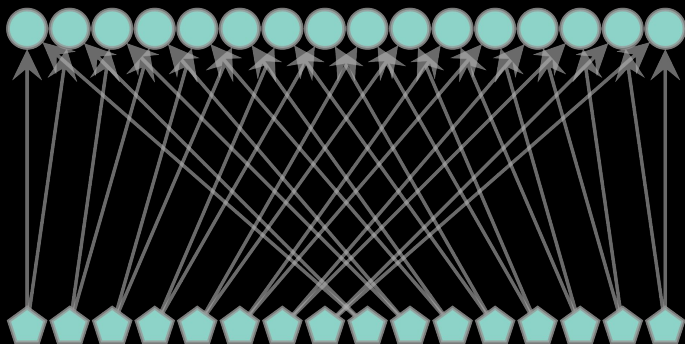


High throughput **but high delay and buffer requirements**

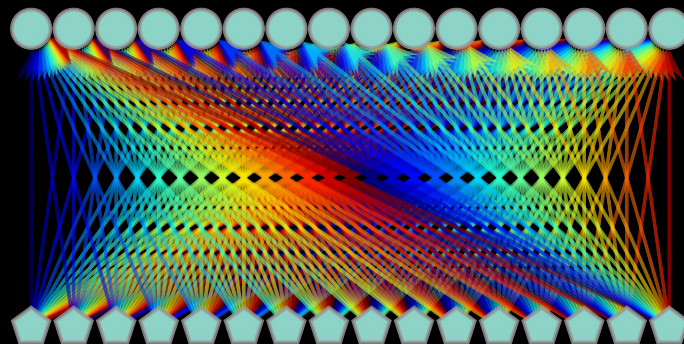
MARS



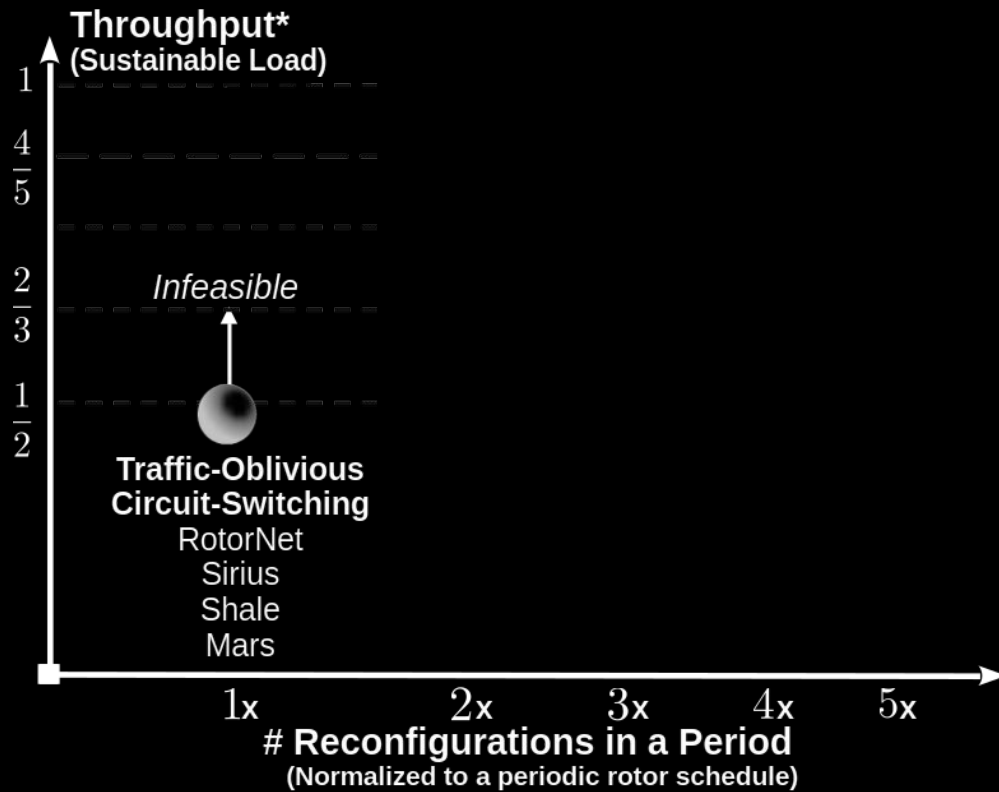
Near-optimal throughput within the available buffer



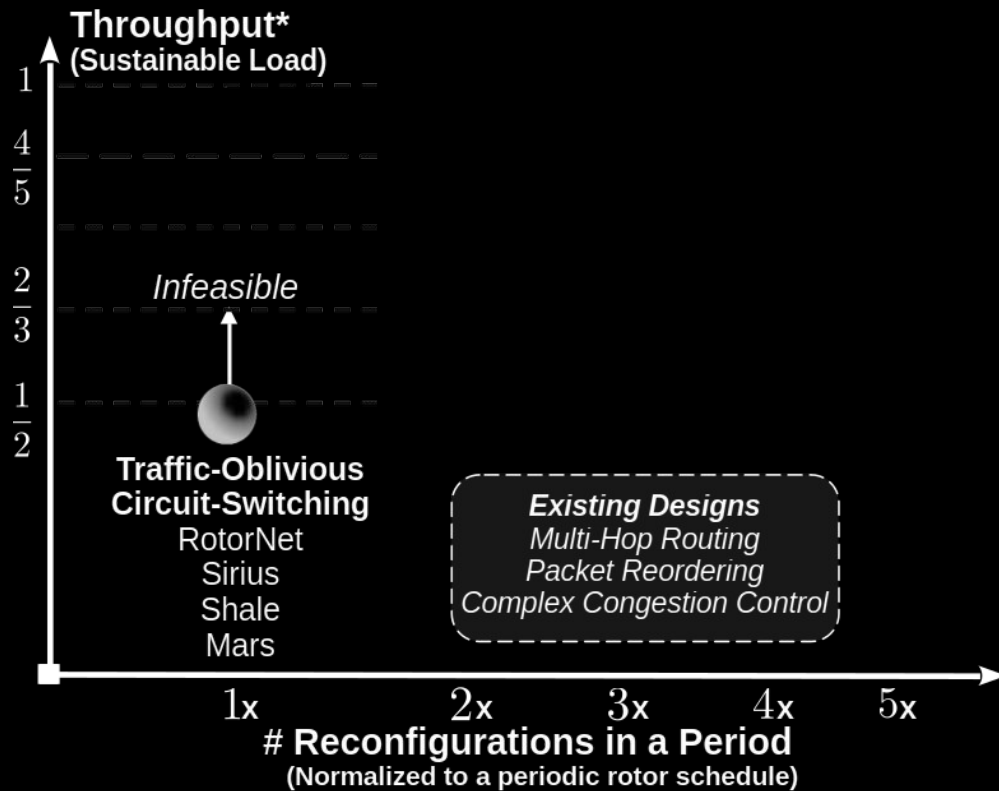
Static DCN: Low Throughput



Existing RDCN: High Delay and buffer

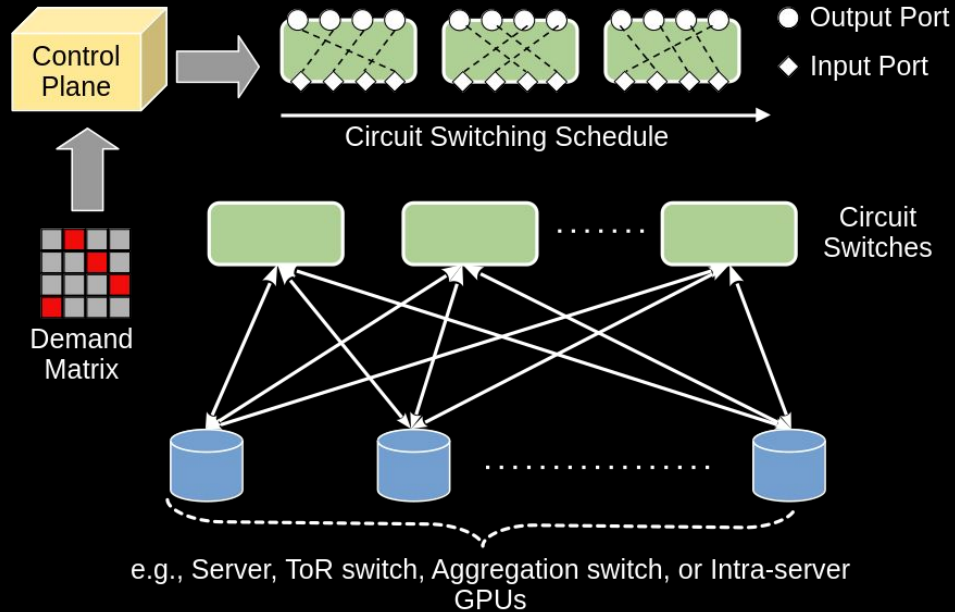


**worst-case throughput*

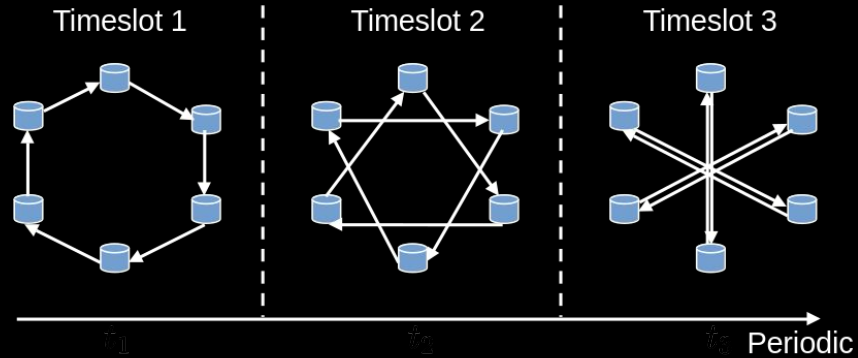


**worst-case throughput*

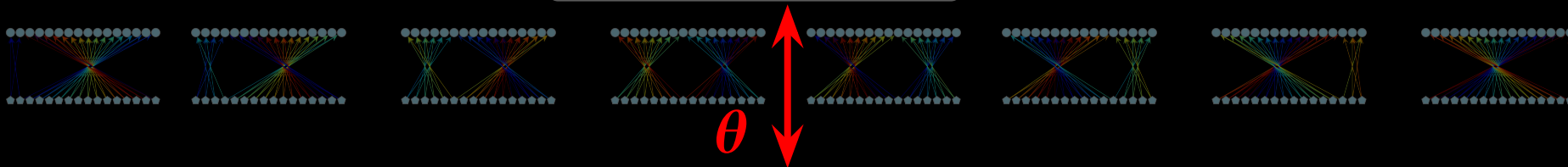
A Traffic-Aware Approach



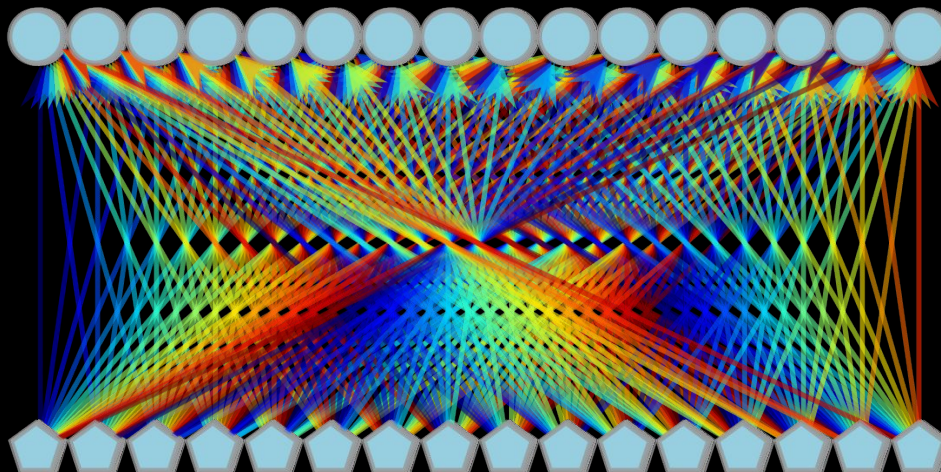
Dynamic Reconfigurable Topology



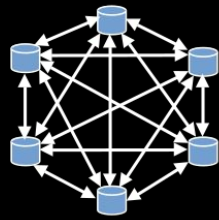
Periodic Graph



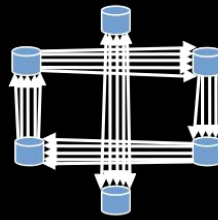
Static Emulated Graph



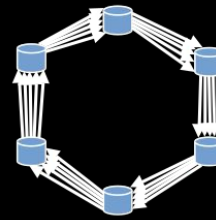
Emulated Topology



All-to-All

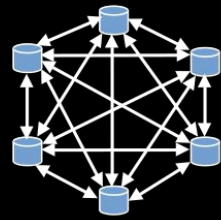


Permutation

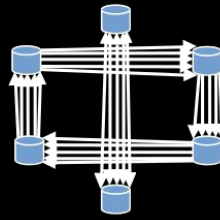


Ring

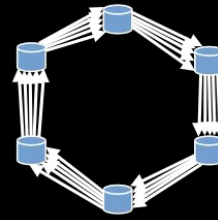
Emulated Topology



All-to-All



Permutation



Ring

Which topology is optimal for throughput?

Revisiting Optimal Topology Problem

Input: Demand Matrix $\mathcal{M}$

Number of nodes n

Degree bound d' in each timeslot

Output: Periodic graph

Maximize: Throughput

Revisiting Optimal Topology Problem

Input: Demand Matrix $\mathcal{M}$

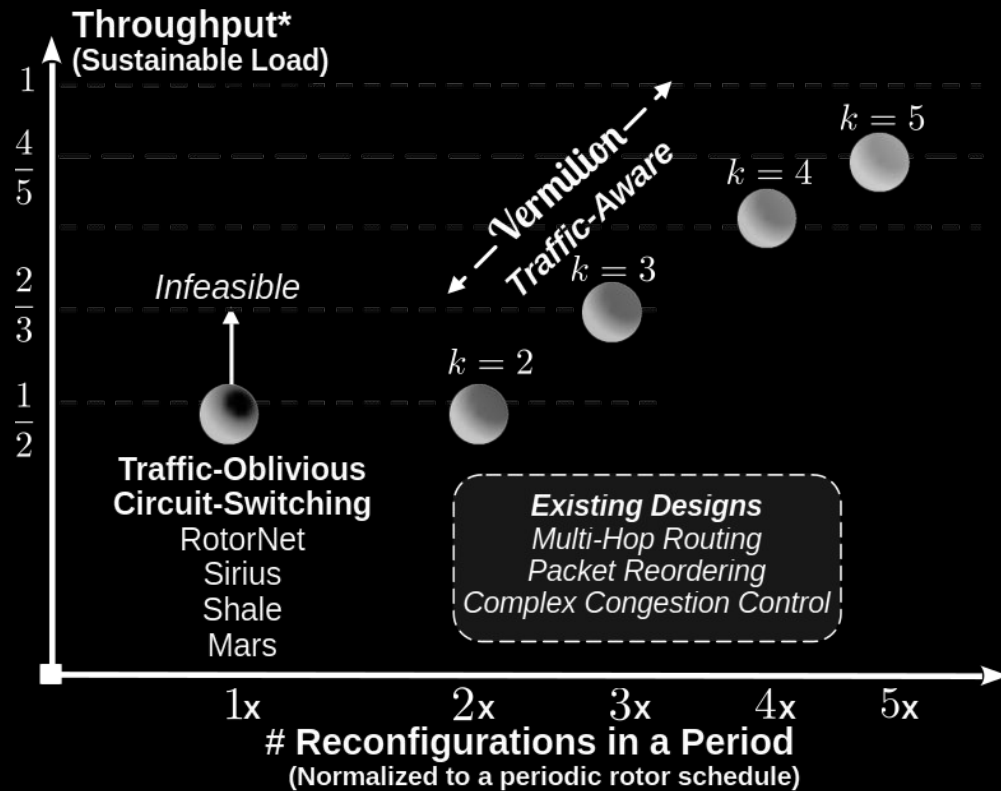
Number of nodes n

Degree bound d' in each timeslot

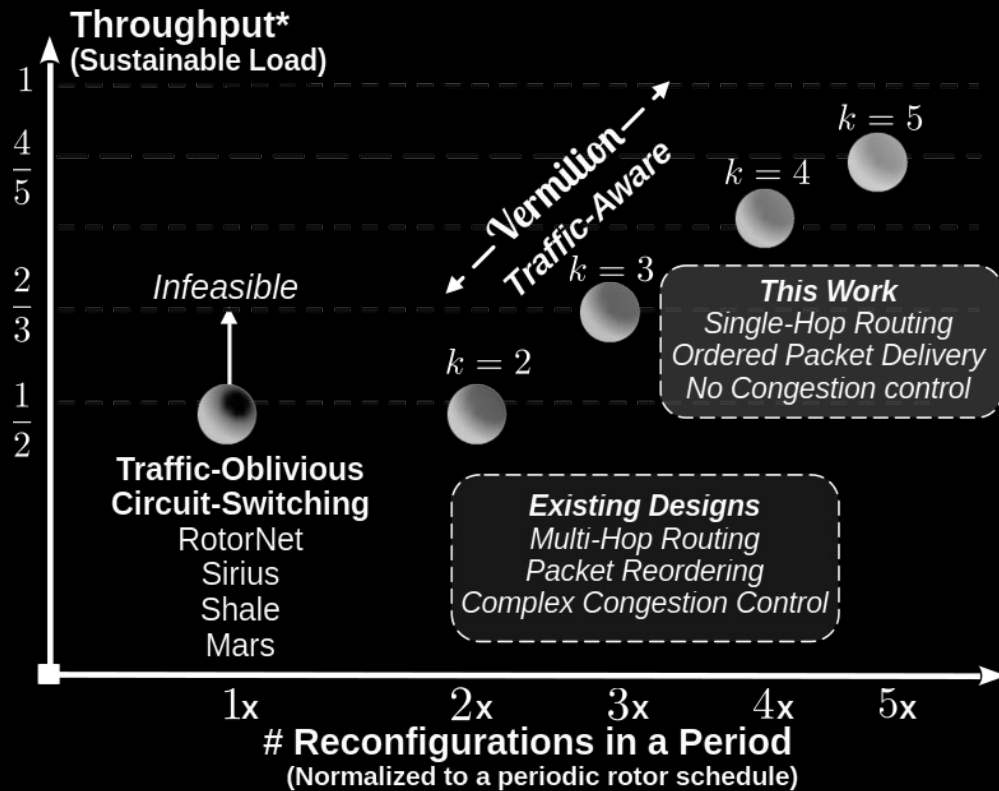
Period bound Γ

Output: Static degree $d = \Gamma \times d'$ multigraph $\rightarrow$ Periodic graph

Maximize: Throughput

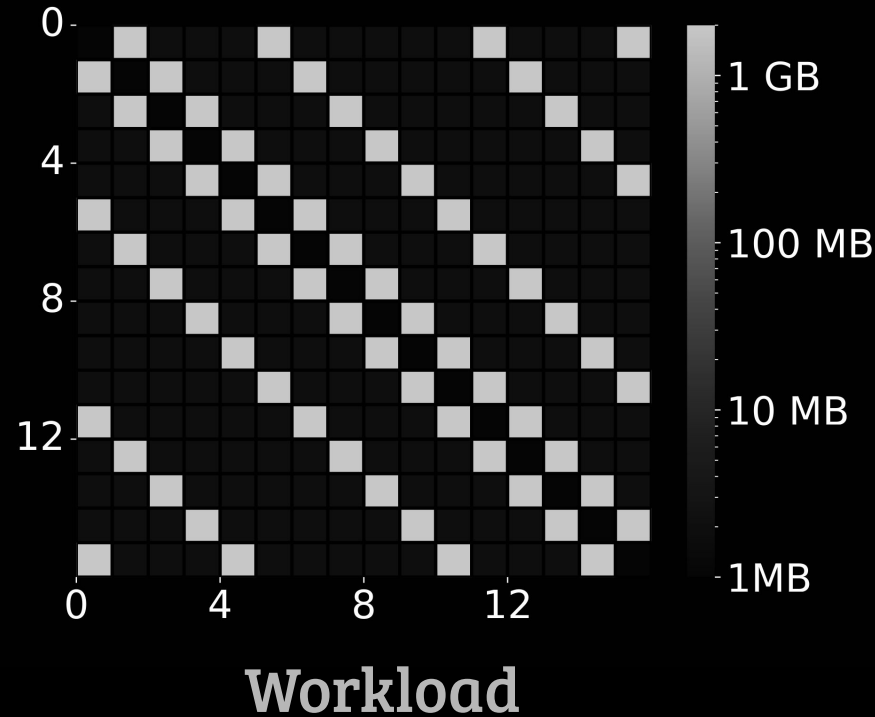


**worst-case throughput*

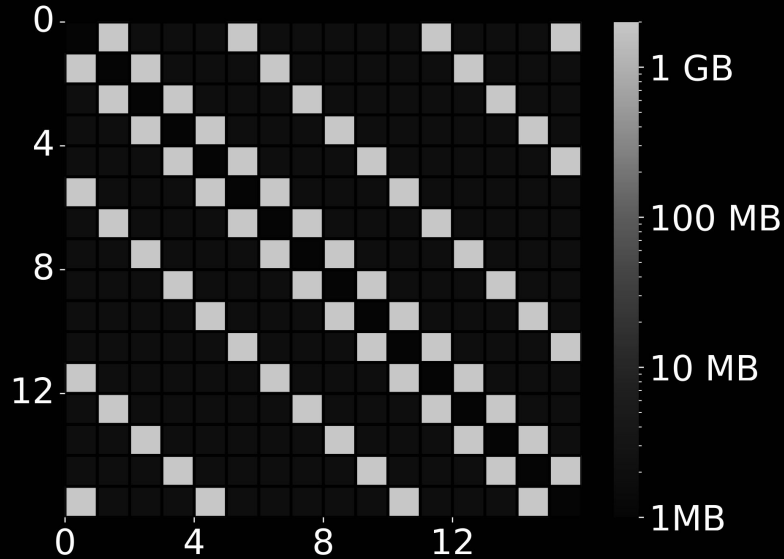


**worst-case throughput*

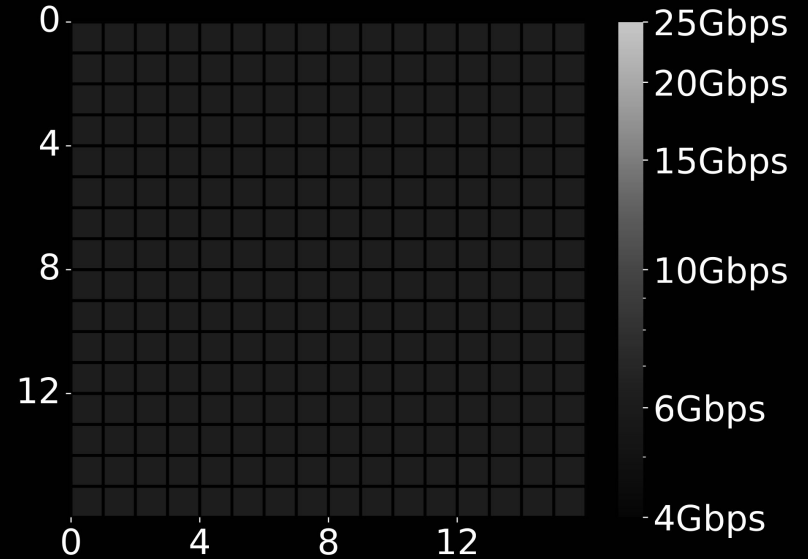
Example: Deep Learning Recommendation Model Workload



Example: Deep Learning Recommendation Model Workload

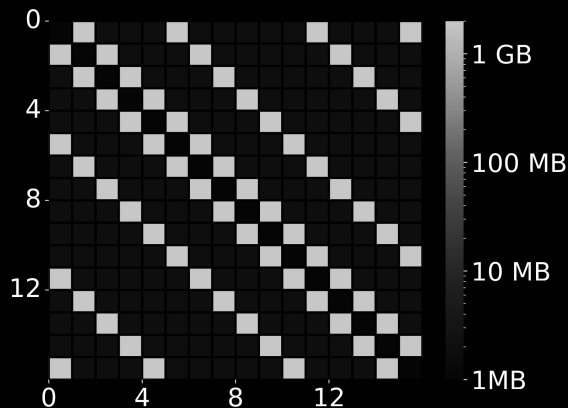


Workload

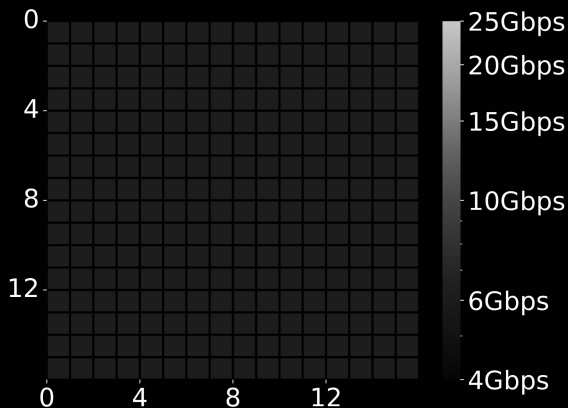


Oblivious Topology

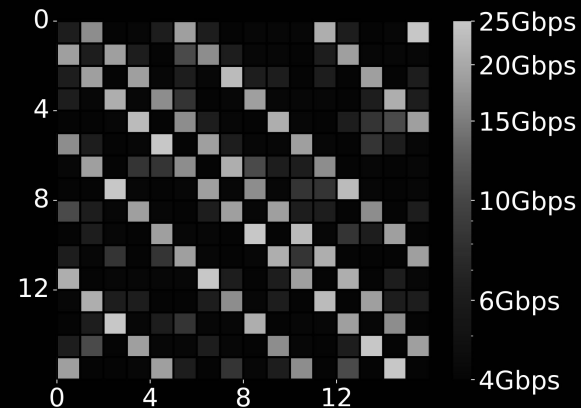
Example: Deep Learning Recommendation Model Workload



Workload

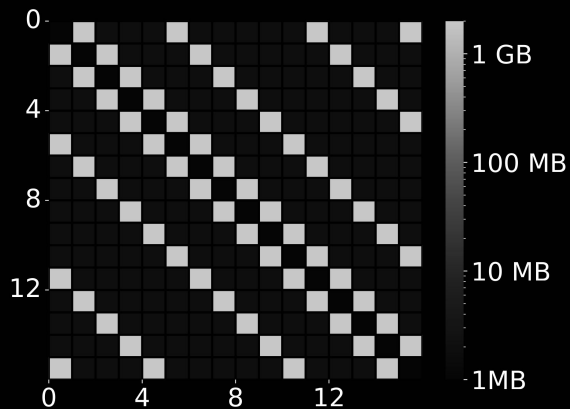


Oblivious

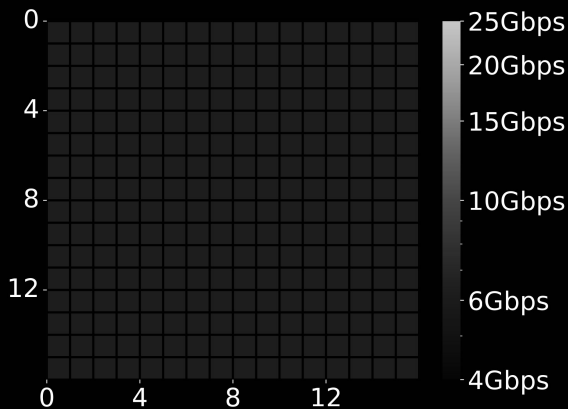


Vermilion

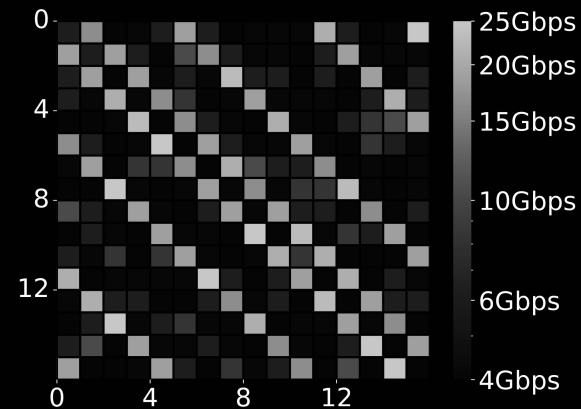
It's a Match!



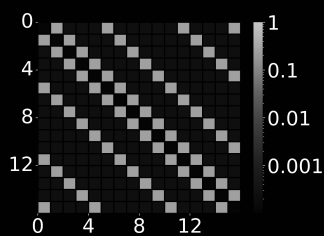
Workload



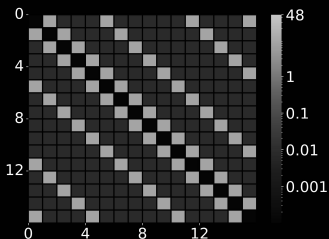
Oblivious



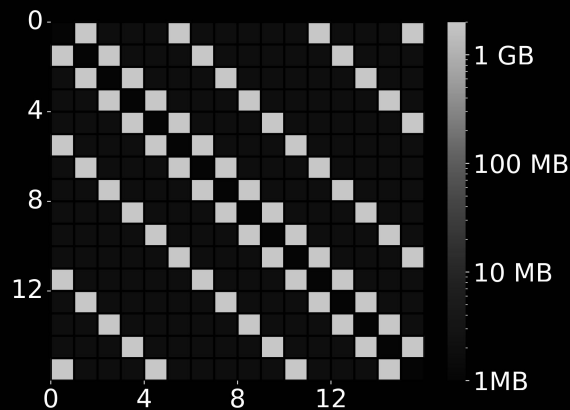
Vermilion



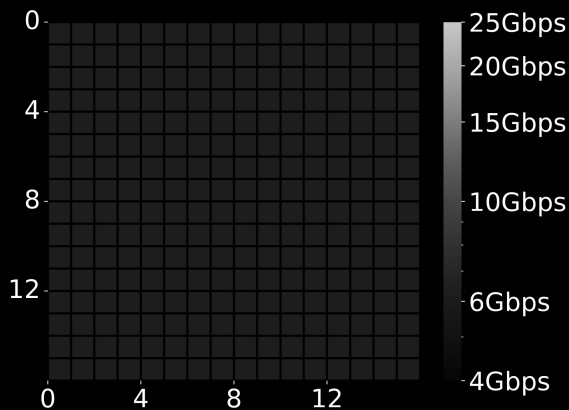
Normalization



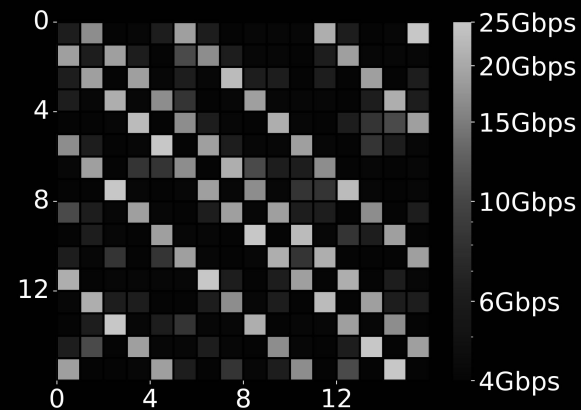
Upscale



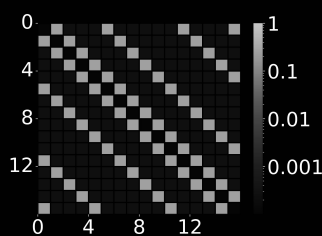
Workload



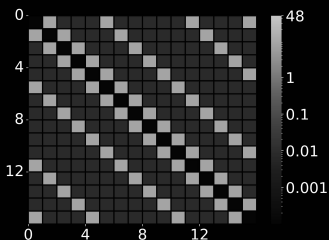
Oblivious



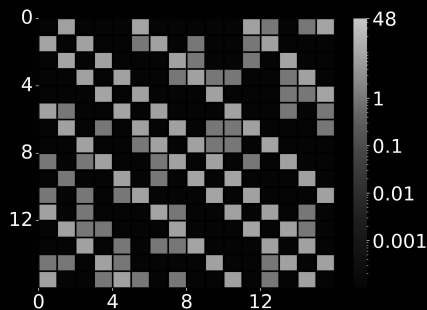
Vermilion



Normalization

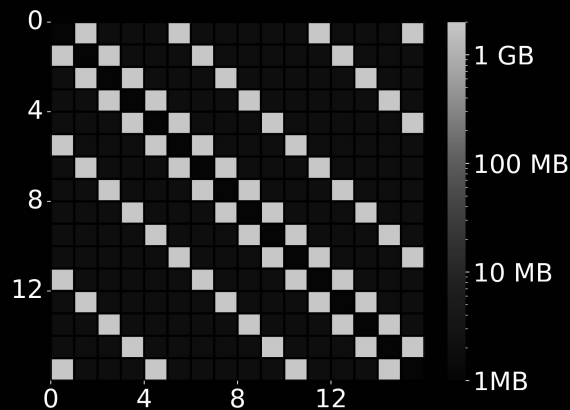


Upscale

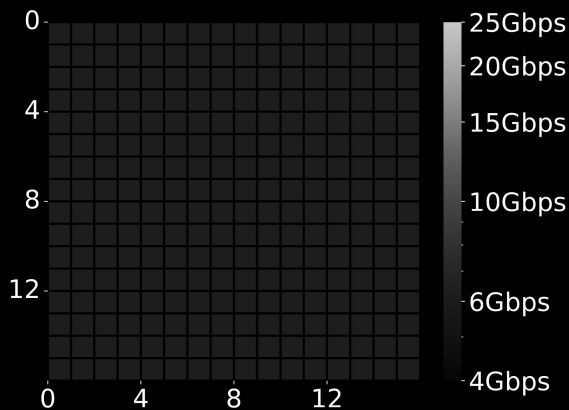


Matrix
Rounding

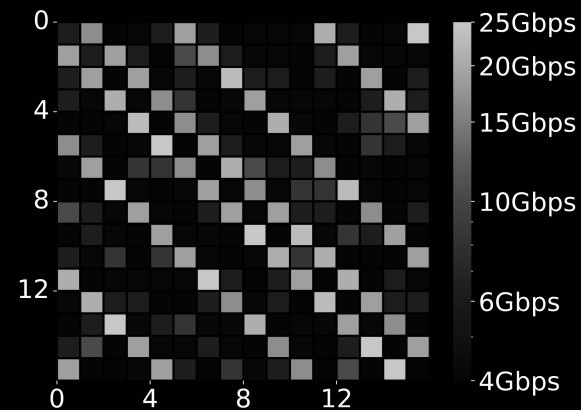
Traffic-aware schedule



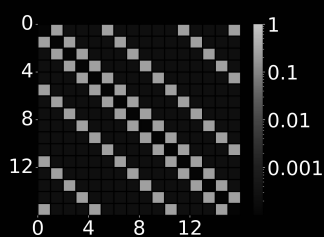
Workload



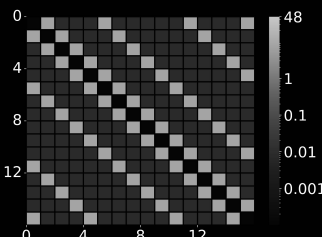
Oblivious



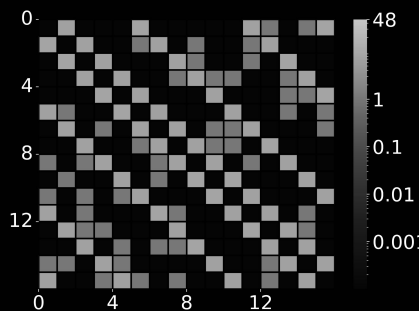
Vermilion



Normalization

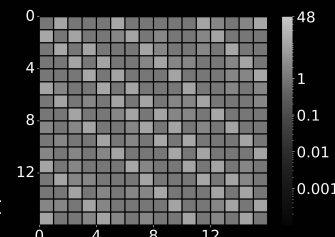


Upscale

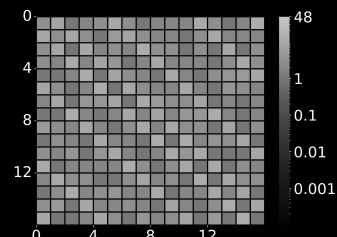


Matrix
Rounding

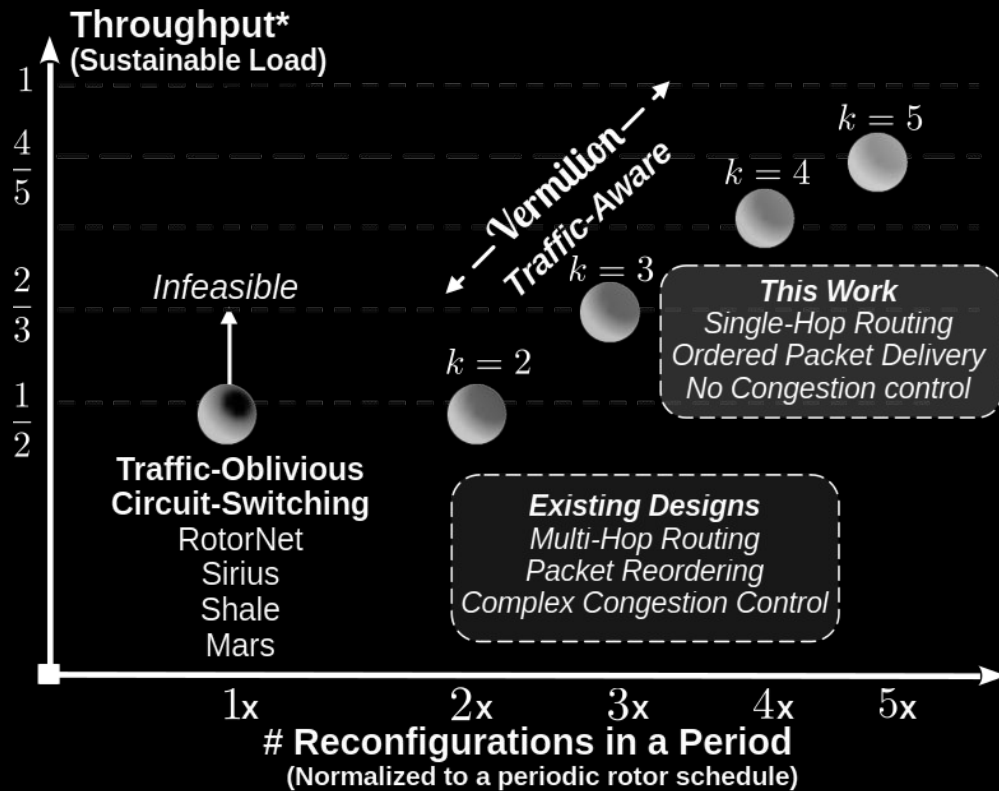
Traffic-aware schedule



+ Oblivious
schedule



+ Random
configuration



**worst-case throughput*

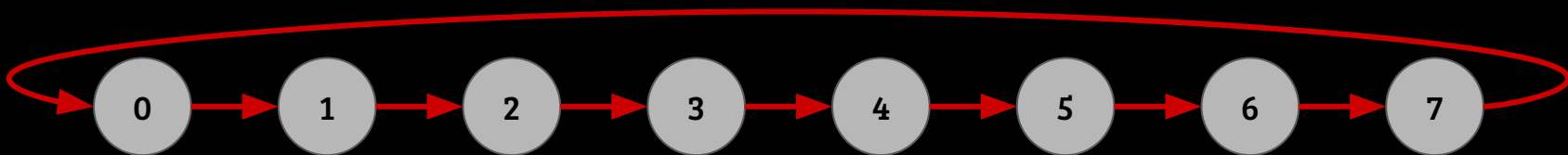
Insights & Reflection on Metrics for Collective Communication

Throughput and Reconfiguration Delay

- Δ = Fraction of time lost in reconfigurations
- Throughput bounds typically scale down by a factor of $1-\Delta$
 - e.g., Throughput bound of periodic switching is $\frac{1}{2}(1-\Delta)$
- **How does the absolute value of reconfiguration time impact performance?**
 - Is 1 microsecond a satisfactory reconfiguration delay?
 - What about 100 milliseconds?
 - **Note:** Δ can be small enough irrespective of the reconfiguration delay

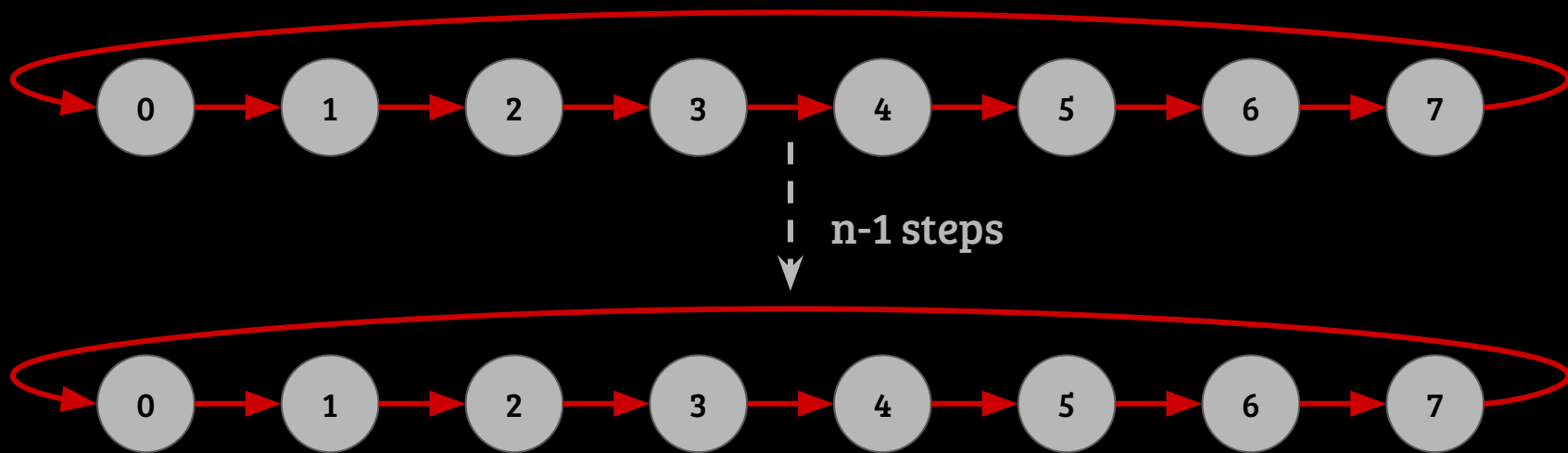
Ring AllReduce

- The communication pattern is a *matching*



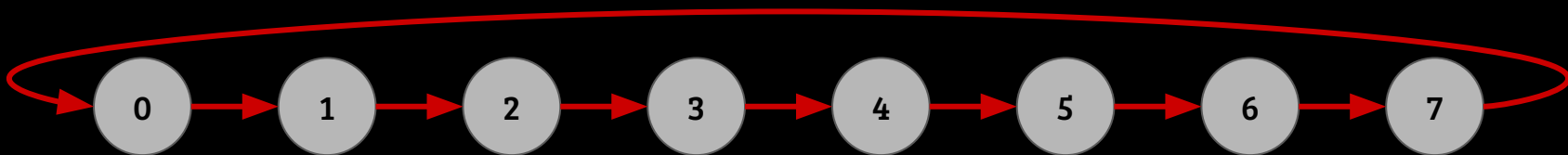
Ring AllReduce

- The communication pattern is a *matching*



Ring AllReduce

- The communication pattern is a *matching*



- Periodic circuit switching
 - Prior analysis suggests a throughput of $1/2$ for the above communication pattern
 - Throughput can in fact be as low as $1/8$
 - Demand matrix abstraction may be the culprit for these contradictions!
 - **Note:** New “demand” (next step) only arrives after completing the previous demand

Key Takeaways

1. Throughput as a metric cannot capture the impact of reconfiguration delay
2. Demand matrix abstraction cannot capture the dependencies in “demand” observed in collective communication

Modeling the Completion Time using α, β Cost Model

α : Initialization time for sending out data

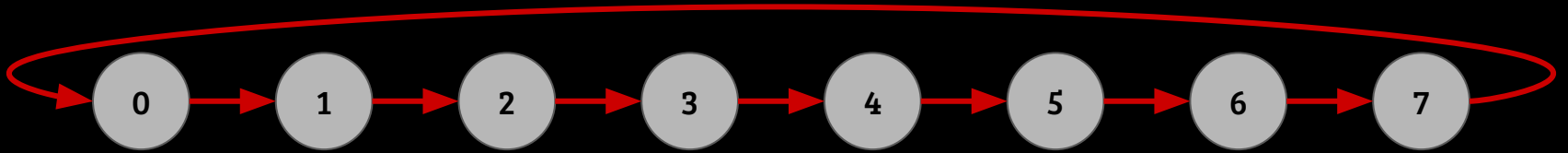
β : Transmission delay for sending one bit at line rate

λ : Congestion factor (number of flows sharing bandwidth)

$\alpha + m \beta \lambda$ = Time taken to send m bits in a single step of the algorithm

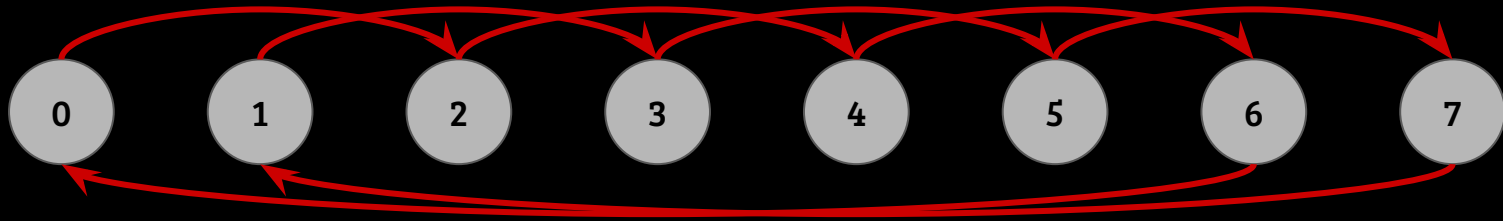
Recursive Doubling

- Step 1: Matching



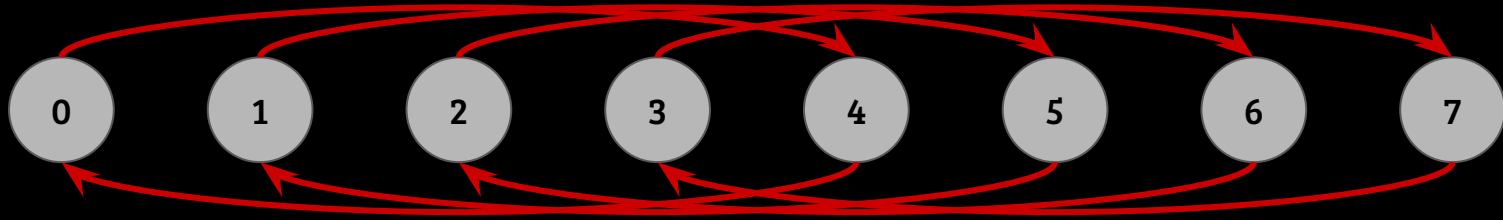
Recursive Doubling

- Step 2: Matching



Recursive Doubling

- Step 3: Matching



Modeling the Completion Time using α, β Cost Model

α : Initialization time for sending out data

β : Transmission delay for sending one bit

λ : Congestion factor (number of flows sharing bandwidth)

$\square$: **Reconfiguration delay**

$\alpha + \square + m \beta$: Reconfigure and align topology to communication matching

$\alpha + m \beta \lambda$: Mismatch

Circuit Switching for Collective Communication

- Each step of the algorithm is a matching
- The schedule for circuit switching is given!
- Our goal is to minimize the completion time of the collective
- Decision in each step: Reconfigure or not (a binary variable)

A perfect opportunity for optimization!

More details coming soon :)

References

- [1] Addanki V, Avin C, Schmid S. [Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks](#). Proceedings of the ACM on Measurement and Analysis of Computing Systems. 2023 Feb 28;7(1):1-43.
- [2] Addanki V, Avin C, Knabe GD, Patronas G, Syrivelis D, Terzenidis N, Bakopoulos P, Marinos I, Schmid S. [Vermilion: A Traffic-Aware Reconfigurable Optical Interconnect with Formal Throughput Guarantees](#). arXiv preprint arXiv:2504.09892. 2025 Apr 14.

Thank you

Vamsi Addanki

vamsi@inet.tu-berlin.de

 @Vamsi_DT