LCDN: Providing Network Determinism with Low-Cost Switches

Philip Diederich
Chair of Comm. Networks
Tech. Univ. of Munich

Yash Deshpande
Chair of Comm. Networks
Tech. Univ. of Munich

Laura Becker
Chair of Comm. Networks
Tech. Univ. of Munich

David Raunecker
Chair of Comm. Networks
Univ. of Würzburg

Alexej Grigorjew
Chair of Comm. Networks
Univ. of Würzburg

Tobias Hoßfeld
Chair of Comm. Networks
Univ. of Würzburg

Wolfgang Kellerer
Chair of Comm. Networks
Tech. Univ. of Munich

***Abstract*—**
The demands on networks are increasing at a fast pace. In particular, real-time applications have very strict network requirements. However, setting up a network that hosts real-time applications is a cost-intensive endeavor, especially for experimental systems such as testbeds. Systems that provide guaranteed real-time networking capabilities usually work with expensive, high-rate software-defined switches. In contrast, real-time networking systems based on low-cost hardware face the limitation of lower link speeds. This paper fills this gap and presents Low-Cost Deterministic Networking (LCDN), a system designed to work with inexpensive, common off-the-shelf switches and devices. LCDN works at Gigabit speed and enables powerful testbeds to host real-time applications with strict delay guarantees. LCDN's performance is similar to industrial- and production-grade solutions. This paper also provides an evaluation of the determinism of a low-cost switch and a Raspberry Pi used as an end-device to demonstrate the applicability of LCDN for inexpensive, low-power systems.

***Index Terms*—network performance, network calculus, network controller, time-sensitive networking.**

I. Introduction

Emerging applications such as the Internet of Things (IoT), real-time multimedia, and financial services have increased the amount for time-critical traffic [1], [2]. These applications require precise guarantees of latency and successful packet delivery. Traditional Ethernet networks, which operate on a best-effort communication model, prioritize network stability over ensuring real-time data delivery. IEEE 802.1 Time Sensitive Networking (TSN) has been developed to tackle these challenges at the data link layer, offering deterministic connectivity and Quality of Service (QoS) within Ethernet networks. The TSN task group has established comprehensive standards addressing various methods to ensure latency bounds.

Traditionally, achieving real-time networking capabilities requires specialized and feature-rich switches, such as those supporting TSN standards. However, these TSN switches' high cost and complexity significantly hinder their widespread adoption in cost-sensitive deployments [3]. Additionally, many traditional TSN methods exhibit low network utilization, making running high-bandwidth applications and those requiring deterministic communication challenging. As a result, there is a growing need for more affordable and efficient solutions to enable deterministic networking in a broader range of applications.

This paper proposes Low-Cost Deterministic Networking (LCDN)[1], a centralized method to provide high-utilization and deterministic data delivery over low-cost switches. The motivation for LCDN is driven by the increasing feature richness of low-cost switch chipsets. These advancements present an opportunity to implement deterministic networking capabilities on more economical hardware. The paper contributes a novel scheme enabling deterministic networking with low-cost switches. LCDN offers strategies that allow users to fine-tune the system. The paper details the system's architecture and implementation, the design choices made, considering the complexity at each step. The paper shows via measurements that low-cost switches based on mass-produced ASICs are sufficiently deterministic. It shows that low-powered end hosts, such as the Raspberry Pi, can be used in LCDN. Furthermore, the paper compares LCDN's performance to RAP, a distributed protocol in TSN, and Chameleon, a high-performance network controller.

The rest of the paper is organized as follows: Sec. II reviews deterministic networking and related SOTA. Sec. III outlines LCDN's building blocks. Sec. IV explains its implementation and design choices. Sec. V presents switch and host measurements. Sec. VI compares LCDN to other controllers. Sec. VII discusses limitations, and Sec. VIII concludes the paper.

II. Background and related work

The primary goal of deterministic networks is to maximize the number of accepted flows in the network, ensuring efficient resource management while maintaining the End-to-End (E2E) latency requirements for each flow [2]. Meeting this goal involves optimizing two main components: routing and scheduling, known as Joint Routing and Scheduling (JRS).

This work received funding from the Deutsche Forschungsgemeinschaft (DFG) - 316878574, from the Bavarian State Ministry for Economic Affairs, Regional Development and Energy (StMWi) project KI.FABRIK (grant no. DIK0249), and the Federal Ministry of Research, Technology and Space (BMFTR) project "6G-Life" (16KISK002). The corresponding Author is reachable under philip.diederich@tum.de

[1]The Code and Hardware measurement results are available under https://github.com/tum-lkn/lcdn-controller

[2]End-to-end latency of the packet refers to the time taken from the network hardware of the source endpoint to the network hardware of the destination endpoint.

978-3-903176-75-1 ©2025 IFIP

In a given network topology comprising nodes and links, the routing solution determines the path for each flow, providing a sequence of nodes from the source to the destination. Each egress port at every node features multiple queue levels. Every queue level provides different latency guarantees (i.e., worst-case) based on the scheduling method. The scheduling solution then assigns each flow to the appropriate queue level.

One key TSN method is the Time-Aware Shaper (TAS), which facilitates time-triggered transmission for scheduled traffic using gate-controlled queues to manage packet transmission. TAS provides precise transmission slots for traffic with stringent timing requirements. The Asynchronous Traffic Shaper (ATS) does not schedule traffic based on time but uses a token-bucket shaper for each flow to control the traffic flow and decrease its burstiness. TAS requires accurate time-synchronization across devices [4], [5]. Credit-Based Shaper (CBS), Strict Priority (SPQ), and other per-class scheduling methods with latency guarantees offer more flexibility and better utilization than TAS and also do not require time synchronization.

All these scheduling methods can use a centralized resource management algorithm on the control plane to ensure efficient allocation of time and queues at the switches. The computational complexity at this centralized controller quickly becomes infeasible for TAS with an increasing number of flows and size of the network [6], [7]. Moreover, an effective resource management protocol must handle dynamic changes in network conditions, such as varying traffic loads and potential link failures.

Path diversity between two endpoints in the network must be leveraged to improve the flow acceptance rate. Routing packets using just the minimum spanning tree or shortest path causes more stress at bottlenecks and leads to lower real-time capacity. Given a list of flows, their latency requirements, and the network graph, an Integer Linear Programming (ILP) can be formulated to obtain the best route for any flow [8]. However, the runtime for the solvers becomes quickly intractable [7]. TAS requires recomputation when a new flow is added or removed from the network and hence struggles under frequent reconfigurations. TAS also demonstrates poor network utilization for sporadic traffic [6], [9]. Chameleon [10] and Loko [11] use SPQ and greedy routing algorithms to achieve fast reconfigurations and high utilization with network determinism. However, they require *OpenFlow* [12] based Software-defined networking (SDN) switches. Since Chameleon is intended for data center networks, it uses expensive rack mount switches, while Loko uses the Zodiac FX, which provides only a 100 Mbps rate on each of its four ports. Besides, the Zodiac FX switches are still more expensive than the low-cost ones used in LCDN. Finally, one can drastically reduce the JRS problem complexity by using distributed TSN protocols such as Stream Reservation Protocol (SRP) and Resource Allocation Protocol (RAP) [13]. However, due to their limited view of the network, they have reduced network utilization and require signaling overhead to deal with reconfiguration.

Distributed and centralized protocols require specialized, complex switches [3]. Our survey did not find any TSN switches commercially available under 600 EUR, largely due to limited adoption and not due to the technologies used. However, cost might decrease in the future with more widespread adoption.

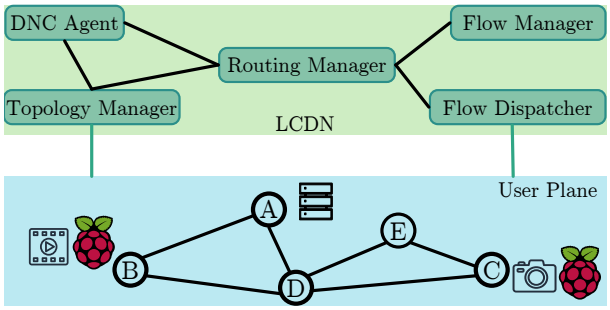


Fig. 1: Overview of the LCDN architecture.

The more practical per-class scheduling methods, such as CBS, SPQ, and Interleaved Weighted Round Robin (IWRR), are more readily available on low-cost switches. However, a JRS problem solution for them to achieve deterministic packet delivery is deficient in the state-of-the-art (SOTA). The low-cost switches built for general-purpose local area networks (LAN) cannot perform arbitrary routing. Moreover, a deep evaluation of the performance of these switches with respect to their predictability and determinism is also missing in the SOTA. LCDN aims to bridge this gap to show that a real-time deterministic network that achieves high utilization can also be achieved using low-cost commercial network hardware.

III. System Description

This section introduces the LCDN System and describes its components. Fig. 1 shows the conceptual deployment of the LCDN System. There are two main parts: The User Plane and the Centralized LCDN Manager. All the user equipment and switches that comprise the physical network live in the User Plane. End-hosts are devices where flows originate or terminate. All components in the User Plane are connected to the LCDN Manager, which configures the switches and end-hosts to adhere to the real-time requirements for all the flows in the network. A lightweight middleware runs on all end-hosts to communicate with the LCDN manager. This middleware is implemented entirely using Linux utility and network tools, and it remains transparent to the applications generating and receiving the flows.

The LCDN Manager comprises five parts: 1. The Deterministic Network Calculus (DNC) Agent, 2. The Routing Manager, 3. The Topology Manager, 4. The Flow Dispatcher, and 5. The Flow Manager. The functionalities provided by some parts are similar to that of Chameleon [10] and Silo [14]. However, the features provided by low-cost switches differ from these works, hence the LCDN implementation. Silo does not utilize the diversity provided by per-class scheduling in the network that LCDN exploits [14]. Chameleon goes one step further by arbitrarily jumping priority classes at potentially every hop for the same flow [10], which cannot be implemented in low-

cost L2 switches. Hence, LCDN solves the JRS problem for a network under the constraint that the priority class of a flow should be preserved from source to destination. Moreover, like Chameleon and unlike Silo and QJump, LCDN can reconfigure the path of previously embedded flows when a new flow is requested.

Deterministic Network Calculus Agent: The DNC agent is at the heart of the scheduling part of the JRS in LCDN. DNC is a mathematical framework used to analyze the real-time performance of flows in a network [15], [16]. DNC uses models to define traffic patterns and device behavior in the network. The models of traffic patterns from data sources are defined in the so-called arrival curves. Arrival curves describe the amount of data the source produces over time. Arrival curves have a constant rate r and a maximum deviation at time t from this rate called burst b. The switch behavior is modeled using service curves. Service curves model the processing times of a switch in different scenarios and the rate at which the switch can forward traffic at a per-class or per-flow level. Given a path of a flow and the devices that lie in that path, Deterministic Network Calculus (DNC) uses min-plus algebra to provide the delay bound, backlog bound, and throughput bound provided that the service curves are strict [16]. If the delay bound for a flow is higher than its required specifications, then the DNC agent cannot directly embed the flow, since its deadline would be violated. If the backlog bound exceeds the buffer capacity of any switch or end host in the path, then the flow may experience packet loss, and this flow cannot be embedded in the network. Low-cost switches can schedule packets based on their priority according to SPQ, WRR, and IWRR, for which strict service curves are derived in [15]–[17]. To determine the service curve of a switch, one may use the methodology described in Section V-A, use information from the datasheet, or an RFC2544 [18] test report if available.

Topology Manager: The Topology Manager builds the Network's topology and translates it into a format the DNC Agent can understand. This task is mainly executed during the setup phase or when changes in the network occur due to the loss of a link or a device. Moreover, it maintains $Q \in \mathbb{N}^+$ copies of the topology where Q is the number of priority classes available in the network. It then weights each link i with its DNC-provided delay bound for that class. These weights depend on the arrival curves of embedded flows, the degrees scheduling method, and the parameters of the switch (See Section V-A).

Routing Manager: The topology manager provides the routing manager with a *queue-level* topology where each link is weighted according to its current worst-case delay. The routing problem now involves finding a path where the sum of all weights across the path is less than the specified delay bound for that flow. Since flows have the same priority on the whole path, the lowest delay will always be on the highest priority queue. Checking only a single queue simplifies routing within LCDN. In LCDN, we assume that flows are embedded one by one. It does not need to find the path of all flows simultaneously. Similar to Chameleon [10], LCDN uses a greedy routing algorithm for finding the path with the lowest delay. Also, LCDN can re-evaluate embedding decisions. For the rerouting, the Routing Manager analyzes embedded flows that share the same link as the flow that could not fit. Next, the Routing Manager checks if the network allows for the embedded flows to be routed in a different queue or path. LCDN provides different strategies for initially embedding flows and for rerouting. Section IV discusses the strategies and highlights their computational complexity.

Flow Dispatcher: The Flow Dispatcher is the component that connects to the end hosts. A lightweight middleware in the end-hosts ensures that a specified flow's rate and burst parameters are adhered to. It must tag every packet for a flow with the specified routing and priority values in its header. For small testbeds, especially with cost constraints, the middleware offers a good solution for ensuring proper flow parameters. However, mistakes or malicious misconfiguration of the middleware can result in a misbehaving network. Enforcing the flows' parameter on an ingress switch would mitigate this. Unfortunately, such features are not necessarily common in low-cost switches. Since LCDN's application focuses on testbeds with full control over the whole network, LCDN only uses the middleware to enforce network stability. Moreover, to embed a given flow with a specified source and destination, many other end-hosts might need to be contacted if the flows originating from them need to be reconfigured. The flow dispatcher does this. When an end host wants to send data in the network, it must talk with the Flow Dispatcher via the middleware by sending a flow embedding request, which contains the source, destination, rate, burst, and deadline of the flow. After the DNC Agent and routing manager decide, the Flow Dispatcher either notifies the source that the flow cannot be admitted or sets up the end hosts to send the traffic on a specific route. The inline traffic of the Flow Dispatcher is also part of the network. It has the lowest priority and a high maximum latency to ensure that it does not interfere with other flows. The idea is that sources have to announce their flows ahead of time where the delay is less critical. Placing restrictions on inline traffic and setup time is a very interesting addition to LCDN. However, this is outside the scope of this paper.

Flow Manager: The Flow Manager keeps track of all embedded flows and requests. Every time LCDN receives a new flow request or flow deletion, the Flow Manager store the action and the result of LCDN to provide telemetry.

IV. Implementation

Switches with certain capabilities are suited for real-time networking. These capabilities are (1) A priority-based scheduler at their egress for which a strict service curve is present, (2) They should be manageable to the extent that the topology manager can connect to them and determine the physical network topology, (3) They should be manageable to the extent that the routing manager can route packets arbitrarily in the network. We find that almost all COTS-managed switches are

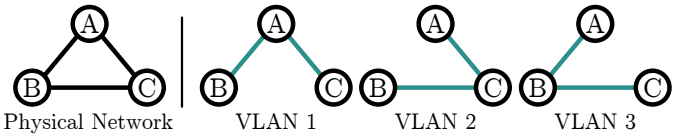


Fig. 2: Example assignment of VLAN identifiers to multiple spanning trees. End-hosts are excluded as they have one connection to a switch and, hence, are already loop-free.

capable of this because they are able to (1) Connect to a management interface over telnet, HTTP or Simple Network Management Protocol (SNMP) (2) VLANs can be assigned to each link, (3) Implement schedulers based on VLAN-priority - also known as Priority Code Point (PCP).

Topology Manager: The setup phase of the network is powered by Link Layer Discovery Protocol (LLDP), which must be enabled in the switches and the end hosts (via the middleware). The topology manager then gets the LLDP report from the devices and builds a graph. The routing manager needs the port numbers interconnecting the switches, while the DNC agent needs the maximum link speed for each edge to model the service curve. Most commercial low-cost switches can process VLAN headers, allowing LCDN to partition the network into different VLANs and route according to a VLAN [19]–[21]. The switches can also run the Spanning Tree Protocol (STP) per VLAN. Therefore, LCDN does not have to individually populate the forwarding table of the switches but only manages multiple spanning trees across all VLANs. The routing manager processes the physical network topology from the discovery phase and partitions it into all possible spanning trees using Gabows algorithm [22], which has a high runtime complexity of $O(V + E + VN)$ where V, E and N are the number of vertices, edges and spanning trees, respectively. This will give N unique paths between all source-destination pairs in the network. The VLAN header size restricts the number of VLANs to 4096. However, VLAN ID 0 and VLAN ID 4095 are reserved; hence, if $N > 4094$, LCDN cannot assign a VLAN to that tree. Here, potential mitigation strategies are the use of Queue in Queue VLAN tags or VXLAN.

LCDN first configures only one of the N spanning trees and assigns it VLAN ID 1. Then, when the DNC agent accepts a new path that does not belong to the already configured spanning tree(s), the topology manager proceeds to configure that spanning tree. This increases the flow configuration time but reduces the overhead of pre-configuring all spanning trees. This paper leaves the problem of an optimal number of pre-configured spanning trees to reduce the flow configuration time for future work. The configuration itself involves a breadth-first search of the spanning tree and assigning the *bridge priority* to each switch ranked according to its depth. The STP settings allow us to have a maximum of 16 different bridge priorities, placing another limitation on network size.

Fig. 2 illustrates the idea of VLAN-based routing in a simple scenario with three switches. The full mesh network with three switches has three different spanning trees. Therefore, in this case, LCDN uses a unique VLAN tag for each spanning tree (VLAN 1, 2, 3 in Fig. 2), and configures the Multiple-STP (MSTP) in switches to select the root node by assigning the higher *bridge priority* to that switch for the VLAN instance of MSTP. When LCDN routes traffic from switch A to switch C, it can use the tree with VLAN ID 1. Whenever the physical resources on link A and C are exhausted, LCDN could route additional traffic from A to C via the spanning tree for VLAN 3. The LCDN middleware tags packets for a particular flow at the source with the corresponding VLAN, which the middleware removes at the destination.

Routing Manager and Strategies: The user can select multiple strategies for LCDN. The different strategies allow to form the behavior of LCDN to the user's needs. There are two main areas of focus for adjusting the behavior of LCDN: (1) Embedding of a flow, (2) Rerouting of already embedded flows. First, this section explains the two different options for embedding a flow request.

Greedy (`G`): This strategy is closely related to how Chameleon embeds flows [10]. The goal is to choose the lowest delay for the flow. Furthermore, `G` aims at providing fast embedding performance compared to the other strategies. Without rerouting, focusing on speed results in poorer path diversity and less embedded flows. To provide fast results, LCDN searches for the shortest path based on the current delays only in the highest priority queue. The greedy path finding without rerouting has a complexity of $O((V + E) \log V)$. However, LCDN still needs to check if any other violations occur in any other edge or queue. The checking step has the complexity $O(E \cdot Q)$, where Q is the number of priority queues. Combining both steps, the total complexity without rerouting is $O((V + E) \log V + E \cdot Q)$.

Not Greedy (`NG`): In comparison to `G`, `NG` focuses on leaving the higher priority queues as empty as possible. This strategy is not concerned about a flow's latency as long as it does not violate its deadline. This should reduce the number of reroutings compared to `G` at the cost of higher flow delays and embed times. For `NG`, LCDN still finds the shortest path based on the current network state. However, it chooses the lowest priority Q that still fits the flow's deadline, potentially checking up to Q queues. Therefore, `NG` results in a complexity of $O((V + E) \log V + E \cdot Q^2)$

Next, LCDN offers different variants for handling the rerouting of embedded flows. Rerouting is required when LCDN receives a flow request and the network state cannot accommodate its resources. First, LCDN still determines the shortest path for the new flow request. Since the flow cannot fit in the network, LCDN searches for reroute candidates. Reroute candidates share at least a single link with the path of the new flow. LCDN orders the candidates from most shared links to least. The general step for rerouting has complexity of $O(F \log F)$ where F is the number of flows currently in the network. Next, LCDN proceeds with the selected rerouting strategy.

Single Flow (`SF`): The Strategy has to be combined with greedy or non-greedy initial embeddings, resulting in `G-SF`

for greedy and NG−SF for non-greedy. LCDN has a maximum number of reroutings it checks before finally not accepting a flow request. With the sorted list of flows that share links, LCDN selects the first r flows, where r is the maximum number of reroutes. In this strategy, LCDN tries to reroute a flow and then embed the new flow. If this fails, the rerouted flow is put back on its original path; otherwise, it can embed the new flow. LCDN tests the rerouting for all r flows. Therefore, the SF rerouting step has a complexity of $O(r \cdot E \cdot Q \cdot F \log F)$.

Compound Flows (CF): Similar to SF, CF results in G−CF and NG−CF. With the same list of sorted candidates, CF proceeds similarly to SF. However, when a rerouted flow does not result in the embedding of the new flow, it stays on its reroute path. Then, LCDN reroutes the next candidate until the new flow is embedded or all r reroute candidates are tested. CF also has the complexity $O(r \cdot E \cdot Q \cdot F \log F)$. However, SF only needs to reroute a single flow and CF up to r flows. Rerouting a flow involves communication between the Flow Dispatcher and an end host. The communication over the network can take time and delay the flow embedding. However, the exact communication and delay analysis are not within the scope of this paper.

For both reroute strategies, LCDN can either move the flow into a lower priority or select a new shortest path by applying Yen's algorithm [23] over the *physical topology* graph.

Once a suitable route is found, the topology manager gives its corresponding VLAN ID and also configures a new tree for that VLAN ID if needed.

Data Plane: The flow admission and access control mechanisms in LCDN use network calculus (NC) via the DNC-agent to achieve predictable latencies. The arrival curves describe the continuous data rate r and a maximum deviation (burst) b at time t, which is also referred to as a token-bucket curve. In the framework, any source must send below the maximum rate and burst advertised in its specifications. To ensure that the sources in hosts comply with the arrival curves, the LCDN uses Token-Bucket Filtering (TBF) from the traffic control tc utility in Linux's network stack to shape the flows at the source.

Controller and Size Limitations: LCDN uses a centralized approach, where a single controller stores all information for every link and for every priority queue. When embedding flows, LCDN finds the shortest path based on the delay, checks the feasibility, and updates DNC parameters on every link for every priority. When re-evaluating embedding decisions, the computational resources depend on the rerouting strategy. Re-evaluating embeddings depends on the rerouting strategy and is more resource-intensive than in Chameleon [10]. Compared to decentralized TSN approaches [13], LCDN has to store more information and process more complex computations.

Another limitation is network size, constrained by the 4094 available VLAN tags for routing. The number of VLANs required depends on the topology and admitted flows. For instance, in a three-node full mesh (Fig. 2), embedding flows only introduces new VLANs when shortest paths or sections thereof are not covered by another spanning treee. Cayley's formula [24] gives the number of possible spanning trees for n nodes. With 4094 VLANs, LCDN can always support networks of up to six switches, while larger networks depend heavily on the topology.

TABLE I: Table summarizes the measured values of the switch. LCDN utilizes these values to model the service curves of the switch using DNC.

Parameter	Value [unit]
Processing Time	4.15 µs
Priority Queues	8
Priority Queueing Overhead	3.5 µs
Buffer Size (per Queue)	62500 - 500000 Byte

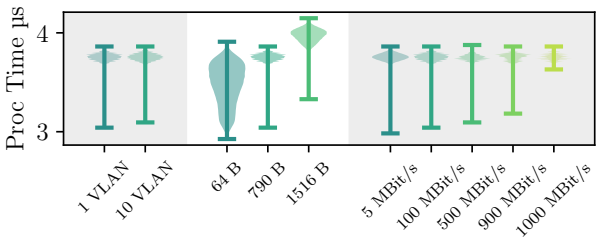


Fig. 3: Switch's processing times across scenarios

V. Hardware Measurements

A. Switch Measurements

To accurately model the real-world switches within the DNC agent, four key parameters are required for the service curve model: (1) Processing time, (2) Number of queues at the egress scheduler, (3) Priority queueing overhead at the egress scheduler, and (4) Buffer size per queue. In [25] and [11], the authors provide methods and insights on how to empirically measure determinism in switches. The measurements below follow the methods to determine if the low-cost switches used in LCDN exhibit deterministic behavior. Notably, switches with prices as low as 20€, such as TP-Link TL-SG108 and Netgear GS-108, can be used with LCDN. However, exhaustive measurement results are provided for the 80€ FS-S2805S-8TF switch, because FS provides RFC2544 reports to validate the result data. This switch features eight RJ-45 ports and two 2 SFP ports. All measurement studies consider only the 8 RJ-45 ports.

Processing Time: Within LCDN, flows with different packet sizes and data rates can traverse the switches within the network. Furthermore, LCDN sets up a varying number of VLANs at these switches. Therefore, the processing time test changes the number of VLANs, i.e., the number of spanning trees, the packet size, and the data rate. The setup from [25, Figure 1] measures the processing latency of the switch. Essentially, the measurement setup is an inexpensive method to conduct an RFC2544 [18] test with taps and a measurement card. Fig. 3 presents the results. It shows that mainly the packet sizes influence the processing time of the switch. The processing time of the switch behaves deterministic and is upper bounded by **4.15µs** for the largest packet size. The number of VLANs and spanning trees, or the load on the network, did not affect the processing time of the switch.

Priority Queues: The DNC agent requires the number of queues Q to facilitate queue-level graphs instead of link-

level graphs. Working on queue-level graphs achieves higher path diversity for packets [10]. The switch advertises to have eight priority queues on the egress side. The switch is set up according to the user manual to achieve strict priority queueing. Then, the second-highest priority queue is saturated. To see if SPQ works, higher priority traffic is sent to the same egress port. Priority queueing works when the higher priority traffic arrives at the destination first. The measurements verify the eight priority queues and their order. LCDN can choose to use only a subset of these queues.

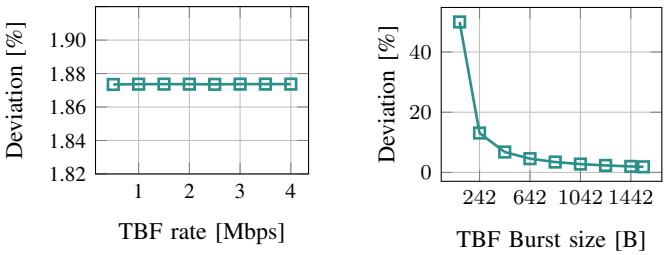
Priority Scheduling Overhead: Priority queuing allows higher-priority traffic to effectively overtake lower-priority traffic within the switch. Whenever the switch sends data out through the egress pipeline, it has to check which priority has data available to send. Then, the switch sends out the data with the highest priority. Checking and deciding which packets to send requires computation. Therefore, the switch needs to spend time calculating which queue to use. Chameleon considers this value as well. LCDN takes the measurement system from [25, Figure 4] to measure the priority queueing overhead. The measurements and their analysis show that the priority queueing overhead is **3.5 µs**. The maximum priority queueing overhead appears when a high-priority flow is interrupted by a lower-priority flow with large packet sizes (1516 Bytes). Thus, confirming that the switch adheres to the strict service curve provided for SPQ in [16].

Buffer Size: The bursts of all flows over the same physical port need to fit within its buffer. Otherwise, the system rejects new flow requests. Therefore, the DNC agent needs to know the buffer sizes. Manufacturers seldom share information about how the buffer is implemented. Since the switch has a common Realtek Switching Chip, the datasheet presents the total packet buffer capacity, which is 4 MByte. LCDN assumes that the buffer is shared equally between queues and ports. Therefore, the size of the available buffer depends on the number of queues and the ports used in the switch. The buffer sizes range from 500000 bytes when only one port is in use to 62500 bytes when all eight ports are in use (for eight priority queues).

Table I summarizes the values LCDN uses for the DNC models.

B. End Host Measurements

The following section validates the token bucket filter's (TBF) performance. The TBF enforces the arrival curve parameter for flows. In the spirit of affordability, the test uses a Raspberry Pi 4 with 8GB of RAM, assuming that any system with more computing resources and superior hardware will perform similarly or better. The test procedure is as follows: the Raspberry Pi connects directly to a PC with a measurement network card. Then, the TBF parameters are set on the interface. Next, the Raspberry Pi sends data to the measurement PC. Finally, the measurement PC receives the traffic and calculates the received rate. Fig. 4 shows the deviation of the token bucket parameters and the measured rate in percent. Fig. 4a presents the deviation for different rates with a 1542-byte burst in percent. Here, the result reveals



(a) Different Flow Rates (b) Different Burst Sizes

Fig. 4: Comparison of TBF's parameter and resulting rate

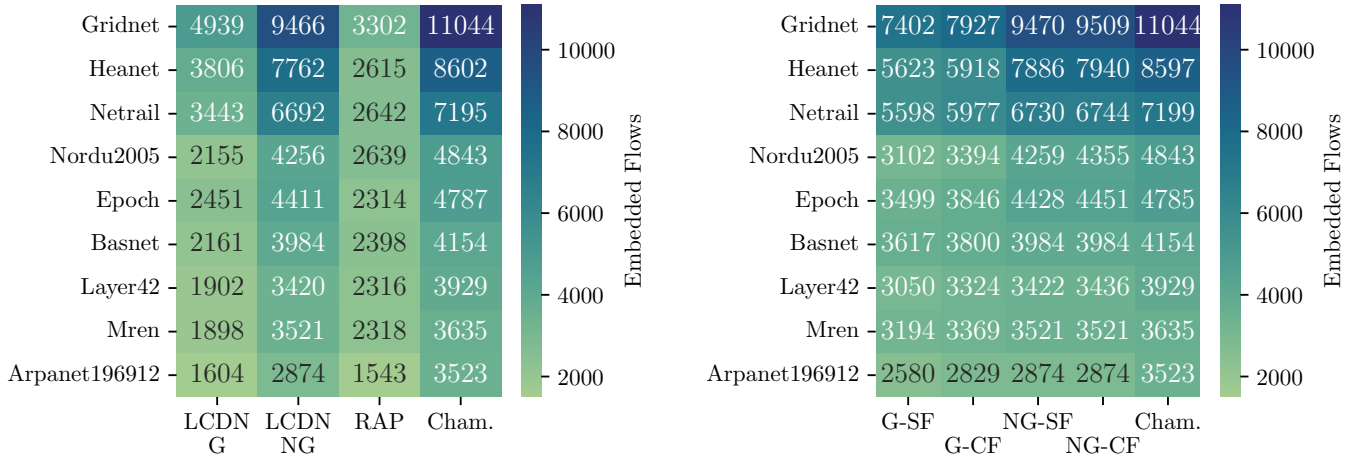
that the actual measured rate is slightly higher than the rate set in the TBF, indicating that the filter is leaky. However, the deviation is below 1.88% and relatively constant. LCDN can model the deviation of the TBF by adjusting the rate parameter of new flows when sending it to the DNC Agent. 4b illustrates the deviation of the TBF with different burst sizes and a constant rate of 3 Mbit/s in percent. Here, the deviation largely depends on the burst size. With small burst values, the deviation is large, with up to 50% deviation. With burst sizes larger than 600 bytes, the deviation is smaller than 5%. Even with the large deviation, LCDN can model the behavior of the TBF. For this, LCDN looks at the flow request, checks the burst size, looks up the deviation in a look table, changes the flow rate of the incoming request, and sends it to further processing.

VI. Evaluation

This section evaluates the performance of LCDN in comparison to Chameleon and RAP. The primary objective is to demonstrate that the performance of LCDN is comparable to Chameleon and RAP for small networks. Results for Chameleon are obtained with the controller described in [10]. For RAP, [13] presents a simulator used to gather the results. This particular implementation uses the Strict Priority latency bound from the IEEE 802.1DD standard, which focuses on simplicity rather than optimal delay bounds. The evaluation focuses on the performance difference between the controllers, not the performance difference due to hardware limitations. All configurations, such as buffer sizes, processing times of switches, and link capacity, are identical across all controllers to ensure fair comparison.

Experimental Setup: The evaluation utilizes 20 small topologies from the TopologyZoo [26]. Each node represents a switch, and each link represents a 1 Gbit/s connection with four priority queues. The TopologyZoo network does not include any end devices. Four end devices with a single priority queue are attached per switch. Using fewer end devices underutilizes the network.

All flow requests share identical traffic specifications (rate, burst, and deadline). The source and destination are chosen uniformly at random from all end hosts, but the source and destination are never the same device. LCDN, RAP, and Chameleon process requests in identical order for fairness. After receiving a flow request, each controller processes the



(a) Performance without rerouting

(b) Performance with rerouting

Fig. 5: Number of embedded flows for different scenarios

request and returns with the admission or rejects the flow due to resource limitations. Each test ends after 50 rejections and repeats five times with different seeds.

The evaluation conducts tests with and without rerouting enabled. When rerouting is active, the controller can reroute up to 10 existing flows per new flow. RAP does not support rerouting and is excluded from the evaluation with rerouting.

Metrics: This paper uses the number of successfully embedded flows as the primary performance metric. Furthermore, path lengths and resulting worst-case flow latencies indicate differences between controllers.

Results without rerouting: Fig. 5a shows the average number of embedded flows for each scenario across selected topologies. The non-greedy strategy `NG` outperforms the greedy strategy `G`. `G`'s design emphasizes minimal worst-case delays and high embedding speed, which leads to fewer embedded flows. LCDN's greedy strategy `G` performs similarly to RAP. RAP embeds more flows in smaller topologies, and `G` performs better on larger topologies. However, Chameleon performs the best, benefiting from higher path diversity.

Results with rerouting: Fig. 5b presents the average number of embedded flows when rerouting is enabled. For LCDN, both `G-SF` and `G-CF` improve significantly, performing closer to `NG-SF` and `NG-CF`. Rerouting multiple flows simultaneously yields better performance for both `G` and `NG`, without adding any meaningful system complexity. However, the performance gain from rerouting remains small for non-greedy strategies. Sending flow requests with identical traffic specifications yields a relatively uniform network utilization, limiting opportunities for reroute candidates. Chameleon reflects this quality of evaluation parameters as well by showing no significant performance difference with and without rerouting.

Path Length Distribution: Longer path lengths demand more network resources. To fairly compare different strategies based on average flow admissions, the path lengths should be equal. Fig. 6 presents the path lengths as hops. The minimum path length is two hops, validating that no source is the destination for the same flow. Furthermore, Fig. 6 reveals very similar path length distributions for all scenarios, except RAP. This particular RAP implementation evaluates the shortest path between all sources and destinations only once. Packets on shorter paths return earlier (they traverse fewer hops), biasing the flow embedding towards flows with shorter paths. Fig. 6 demonstrates this bias. Additionally, `NG-SF`, `NG-CF`, and Chameleon exhibit longer maximum path lengths.

Worst-Case Flow Latency: Each controller provides the worst-case latency for all flows based on the path they are embedded on. Fig. 7 displays the latency in all scenarios. The red line denotes the traffic specification's deadline of 20 ms. No worst-case latency can exceed the deadline. The greedy strategies `G`, `G-SF`, and `G-CF` demonstrate the expected low latencies. Conversely, the non-greedy strategies `NG`, `NG-SF`, and `NG-CF` exhibit higher latencies. Here, the distribution reveals the different priority queue levels because the non-greedy strategies embed flows on the lowest priority queue possible. Furthermore, Chameleon tends to have low latencies as well. Chameleon's maximum latency is closer to the deadline than other strategies, showing that Chameleon benefits from the higher path diversity it offers.

Discussion: Comparing different controllers and different topologies in a fair manner is challenging. To provide fairness, the evaluation uses a single traffic specification for all flows. Furthermore, the evaluation shows that the path lengths are similar for LCDN's strategies and Chameleon, making the number of flows a valid metric to compare controller performance there. However, this metric alone cannot present a holistic view of controller performance and fails to fairly compare RAP. Overall, LCDN demonstrates competitive performance relative to RAP and Chameleon, especially with non-greedy strategies. While Chameleon retains the highest performance due to higher path diversity, LCDN comes very close and offers a tunable trade-off between embedded flows, worst-case delays, and embedding speed, making it a viable alternative for small to medium-sized topologies.

To gain deeper insights into LCDN's behavior, this section

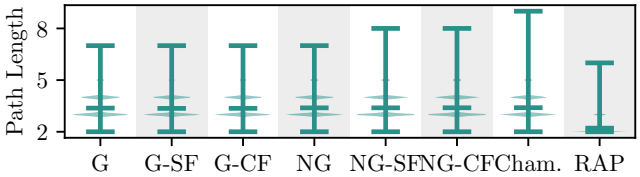


Fig. 6: Path length distribution across all flows

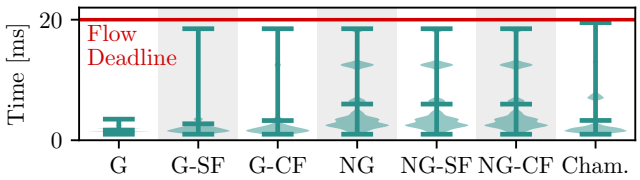


Fig. 7: Worst case flow delays across all flows

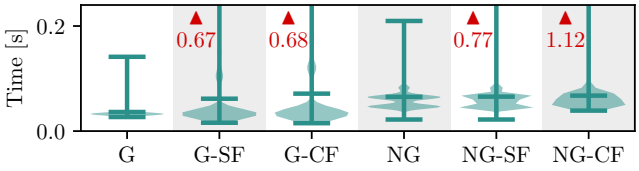


Fig. 8: Embedding times for a single flow

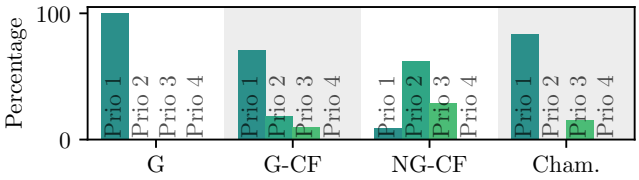


Fig. 9: Queue distribution of embedded flows

evaluates LCDN on the topology *Layer 42* [3]. This evaluation highlights the four key aspects: (1) Embedding time per flow, (2) Flow distribution across priority queues, (3) Buffer utilization, and (4) Delay budget usage.

Embedding time: Fig. 8 presents the per-flow embedding times across all LCDN strategies. The greedy strategies `G`, `G-SF`, and `G-CF` outperform the non-greedy strategies on average. `NG`, `NG-SF`, and `NG-CF` have to check multiple priority queues before embedding, leading to higher per-flow processing times. Furthermore, rerouting multiple flows `CF` requires more time than rerouting single flows `SF`. As expected, having rerouting enabled leads to higher embedding times than without rerouting, since even without successful reroutes, the controller still needs to check reroute candidates. Overall, LCDN embeds the majority of flows in 100ms or less. LCDN is intended for use in research projects. Therefore, LCDN's programming does not focus on providing the highest possible speeds and is not compared to Chameleon.

Queue distribution: Greedy and non-greedy LCDN strategies favor different priority queues. The queue distribution helps to better understand the network's utilization. Fig. 9 shows the flows' distribution on the priority queues for `G`, `G-CF`, `NG-CF`, and Chameleon. As expected, `G` places all flows on the highest priority queue, which is queue 1. However, greedy with rerouting `G-CF` distributes flows to queue 2 and queue 3 as well. Similarly, Chameleon embeds most flows on the highest priority queue and some flows, or segments of their paths, on the lower priority queue 3. Finally, `NG-CF` reverses the embeddings and admits more flows to lower priority queues than to higher priority queues.

Buffer and delay usage: For *Layer 42*, `G-CF` and `NG-CF` admit a similar number of flows on average, 3324 and 3436, respectively (see Fig. 5b). However, the queue distribution is completely different (see Fig. 9), leading to different average flow delays. To illustrate the differences between both strategies, Fig. 10 shows the buffer utilization and Fig. 11 presents the delay usage. LCDN calculates the worst-case burst for every link and every priority queue. Fig. 10 displays the burst as a percentage of the available buffer. `NG-CF` utilizes a lot of buffer in priority queues 2 and 3. In comparison, `G-CF`'s buffer usage is distributed more, especially in the highest priority queue. Every priority queue has a maximum delay threshold assigned. When LCDN embeds a flow in a queue, the flow uses some of the delay budget available for this queue. Fig. 11 presents the used delay budget compared to the threshold in percent. Here, `G-CF` uses a lot of delay budget on priority queues 1 and 2. In comparison, `NG-CF` requires less delay budget in the first queue. However, `NG-CF` utilizes the budget of queues 2 and 3 more on some links. In summary, the buffer and delay usage demonstrate the difference between the greedy and the non-greedy strategy. Where the greedy strategy tends to use up the delay of high-priority queues, the non-greedy strategy favors higher buffer utilization on lower-priority queues. The strategies and thresholds can be fine-tuned for every topology and scenario to provide the best performance.

[3]https://topology-zoo.org/maps/Layer42.jpg visualizes the topology.

VII. Discussion of Limitations

LCDN is designed to work with relatively low-cost hardware. In this initial setup, LCDN works with the FS S2805S switch series, which starts at around 80€, and the Raspberry Pi 4, which is in a similar price range. Clearly, these devices have limitations. These limitations decrease LCDN's performance compared to other methods, like Chameleon or RAP. However, LCDN manages to stay quite competitive. The number of usable VLANs (4094) available on the switches, slower

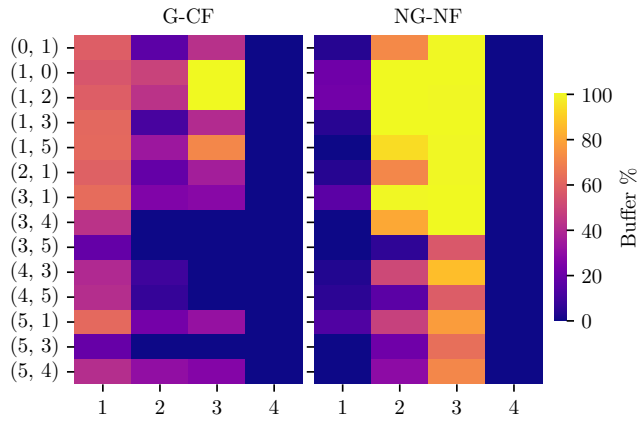


Fig. 10: Buffer usage per link and per priority queue as percentage of the available buffer

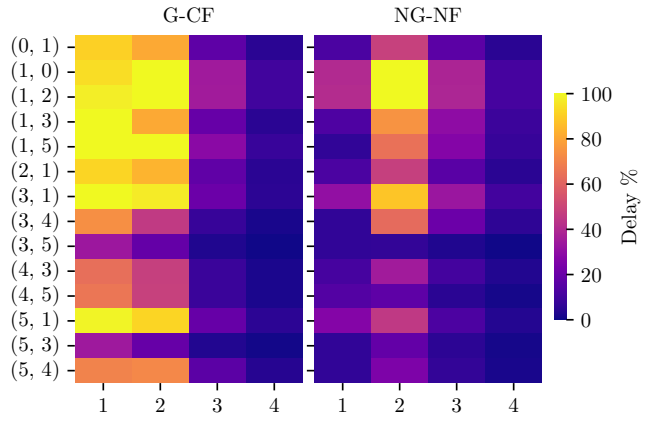


Fig. 11: Used delay per link and per priority queue as percentage of the delay threshold

communication with the switch's management interface, and the overhead of configuring multiple spanning trees with constrained hardware all present challenges for using LCDN in small networks or testbeds. Furthermore, LCDN does not take reliability into account. LCDN only protects flows from delays in queuing and forwarding, not from link failures. LCDN could be extended to work with ideas from TSN, like FRER, to improve the resilience against link failures. Moreover, the lack of advanced schedulers like TAS also restricts its use to applications where a few time-sensitive flows must co-exist with other background traffic.

VIII. Conclusion

This paper presents LCDN, a novel system that presents network determinism with affordable hardware. It uses a centralized controller that monitors the whole state of the network. This allows LCDN to adapt a Network Calculus framework from an already existing controller to suit its use case with low-cost devices. Using low-cost and, therefore, low-power devices has implications for the system, especially for the routing over L2 switches. LCDN solves the routing problem with multiple spanning trees and source-routed flows. LCDN addresses other integration shortcomings by having additional software components in the data plane that keep track of potential violations. In that way, LCDN can intervene and preserve network determinism. This paper provides the parameters for the strict service curve of the FS S2805S and shows that the Raspberry Pi is a suitable end-device, achieving the goal of low-cost deterministic networking. Furthermore, LCDN can be tuned to fit specific use cases and topologies, while performing similarly to RAP and Chameleon.

References

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, no. 1, 2017.

[2] D. N. Chorafas and H. Steinmann, *Implementing networks in banking and financial services*. Springer, 2016.

[3] A. Pruski, M. A. Ojewale, V. Gavrilut, P. M. Yomsi, M. S. Berger, and L. Almeida, "Implementation cost comparison of tsn traffic control mechanisms," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021.

[4] "IEEE 802.1Q: Bridges and bridged networks AMENDMENT 3: Enhancements for scheduled traffic," IEEE, Standard, 2018.

[5] L. Thomas and J.-Y. Le Boudec, "On time synchronization issues in time-sensitive networks with regulators and nonideal clocks," *Proc. ACM Meas. Anal. Comput. Syst.*, jun 2020.

[6] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (tsn)," *IEEE Access*, 2023.

[7] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.

[8] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation," *IEEE Communications Surveys Tutorials*, no. 1, 2018.

[9] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Performance comparison of ieee 802.1 tsn time aware shaper (tas) and asynchronous traffic shaper (ats)," *IEEE Access*, 2019.

[10] A. Van Bemten, N. Deric, A. Varasteh, S. Schmid, C. Mas-Machuca, A. Blenk, and W. Kellerer, "Chameleon: predictable latency and high utilization with queue-aware and adaptive source routing," in *Proceedings of CoNEXT*. Barcelona Spain: ACM, Nov. 2020.

[11] A. Van Bemten, N. Deric, J. Zerwas, A. Blenk, S. Schmid, and W. Kellerer, "Loko: Predictable latency in small networks," in *Proceedings of CoNEXT*. New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 355–369.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, mar 2008. [Online]. Available: https://doi-org.eaccess.tum.edu/10.1145/1355734.1355746

[13] D. Raunecker, S. Geissler, A. Grigorjew, P. Diederich, W. Kellerer, and T. Hossfeld, "Centralized vs. decentralized: A hybrid performance model of the tsn resource allocation protocol," in *2024 20th International Conference on Network and Service Management (CNSM)*, 2024, pp. 1–9.

[14] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," *SIGCOMM Comput. Commun. Rev.*, aug 2015.

[15] A. van Bemten and W. Kellerer, "Network Calculus: A Comprehensive Guide," Technical University Munich, Munich, Technical Report 201603, Aug. 2016.

[16] J.-Y. Le Boudec, P. Thiran, G. Goos, J. Hartmanis, and J. Van Leeuwen, Eds., *Network Calculus*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, vol. 2050.

[17] S. M. Tabatabaee, J.-Y. Le Boudec, and M. Boyer, "Interleaved weighted round-robin: A network calculus analysis," in *2020 32nd International Teletraffic Congress (ITC 32)*, 2020.

[18] S. Bradner and J. McQuaid, "Rfc2544: Benchmarking methodology for network interconnect devices," 1999.

[19] M. Besta, J. Domke, M. Schneider, M. Konieczny, S. D. Girolamo, T. Schneider, A. Singla, and T. Hoefler, "High-performance routing with multipathing and path diversity in ethernet and hpc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 943–959, 2021.

[20] T. Otsuka, M. Koibuchi, T. Kudoh, and H. Amano, "Switch-tagged vlan routing methodology for pc clusters with ethernet," in *2006 International Conference on Parallel Processing (ICPP'06)*, 2006, pp. 479–486.

[21] G. N. Kumar, K. Katsalis, and P. Papadimitriou, "Coupling source routing with time-sensitive networking," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 797–802.

[22] H. N. Gabow and E. W. Myers, "Finding all spanning trees of directed and undirected graphs," *SIAM Journal on Computing*, no. 3, 1978.

[23] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, no. 11, 1971.

[24] A. Cayley, "A Theorem on Trees," in *The Quarterly Journal of Mathematics*, vol. 23, 1889, pp. 376–378.

[25] A. van Bemten, N. Deric, A. Varasteh, A. Blenk, S. Schmid, and W. Kellerer, "Empirical Predictability Study of SDN Switches," in *2019 ACM/IEEE Symposium ANCS*, Sep. 2019, pp. 1–13.

[26] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.