Dissertation / Doctoral Thesis

Titel der Dissertation / Title of the Doctoral Thesis

Algorithms for Traffic Control Challenges in Softwarized Networks

verfasst von / submitted by

Mahmoud Parham

angestrebter akademischer Grad / in partial fulfillment of requirements for the degree of

Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreuerin / Supervisor:	Univ.-Prof. Dr. Stefan Schmid

Abstract

Data centers face growing scalability and operational challenges due to the rapid expansion of bandwidth-intensive applications such as remote collaboration and video streaming services. Scalable networks must react to unanticipated events (e.g., failures) promptly and adapt to traffic pattern or policy changes with minimal human intervention. Such qualities are achievable via virtualized and software-defined networking. A virtualized node (VN) is a network resource such as a firewall and load balancer that is emulated in software and logically decoupled from the underlying hardware. On the other hand, the software-defined networking (SDN) decouples routing computations from the underlying packet forwarding mechanism. SDN's controller has a global view of the network, executes computational tasks, and installs routes remotely. This thesis is primarily motivated by the flexibility and potentials of virtualized and software-defined networks. We leverage them to influence traffic flow without altering the underlying network structure; we refer to this paradigm as traffic control. This thesis focuses on several traffic control challenges:

- Traffic Engineering. To achieve a balanced bandwidth utilization and low congestion, we introduce a new optimization approach that combines two existing traffic engineering techniques. We prove its merits and provide polynomial-time algorithms.
- Fast Traffic Rerouting. Once the traffic encounters a faulty link, the adjacent router node reroutes the traffic along a pre-installed backup route, ideally oblivious to non-incident faulty links (locally and "fast"). We formulate the resiliency model, present backup route schemes, and prove their resiliency for special networks.
- Traffic Partitioning. Communication between VNs can be significantly wasteful of the inter-server bandwidth. An appropriate partitioning of these nodes can potentially reduce the inter-server traffic. Approximating the optimal partitioning is computationally challenging as the underlying communication pattern is not known a priori. We present online algorithms that proactively transform costly inter-server traffic into (inexpensive) local traffic based on the recent communication history.
- Traffic Waypoint Routing. Service chaining is a technique for composing custom network services using primitive functions. For instance, a security service may be composed of several functions: a firewall, DPI, and virus scanner. Computing shortest routes passing through a given set of nodes is an NP-hard problem. We present hardness bounds and algorithms for a few special networks that render the problem tractable.

Zusammenfassung

Rechenzentren stehen vor wachsenden Herausforderungen hinsichtlich Skalierbarkeit und Betrieb aufgrund der rasanten Zunahme bandbreitenintensiver Anwendungen wie Remote-Zusammenarbeit und Video-Streaming-Dienste. Skalierbare Netzwerke müssen auf unvorhergesehene Ereignisse (z. B. Ausfälle) sofort reagieren und sich an Verkehrsmuster oder Richtlinienänderungen mit minimalem menschlichem Eingriff anpassen. Diese Eigenschaften lassen sich mit virtualisierten und softwaredefinierten Netzwerken erreichen. Ein virtueller Netzwerkfunktionsknoten (VNF) ist eine Netzwerkressource wie eine Firewall und ein Load Balancer, die in Software emuliert werden und logisch von der zugrunde liegenden Hardware entkoppelt sind. Auf einer anderen Ebene entkoppelt das softwaredefinierte Netzwerk (SDN) die Routing-Berechnungen von dem zugrunde liegenden Paketweiterleitungsmechanismus. Der SDN-Controller hat einen globalen Überblick über des Netzes, führt Berechnungsaufgaben aus und installiert die Routen aus der Ferne. Die Motivation für diese Arbeit liegt in erster Linie in der Flexibilität und den Möglichkeiten von virtualisierten und softwaredefinierten Netzwerken. Wir nutzen sie zur Verkehrsfluss zu beeinflussen, ohne die zugrundeliegende Netzwerkstruktur zu verändern; wir bezeichnen dieses Paradigma als Verkehrskontrolle. Diese Arbeit konzentriert sich auf mehrere Herausforderungen der Verkehrskontrolle:

- Verkehrstechnik. Um eine ausgewogene Bandbreitennutzung und eine geringe Überlastung zu erreichen, stellen wir einen neuen Optimierungsansatz vor, der zwei bestehende Verkehrstechniken kombiniert. Wir beweisen seine Vorzüge und liefern Algorithmen mit polynomialer Zeit.
- Schnelles Verkehrs-Rerouting. Sobald der Verkehr auf eine fehlerhafte Verbindung stößt, leitet der benachbarte Routerknoten den Verkehr über eine vorinstallierte Backup-Route um, im Idealfall unabhängig von nicht fehlerhaften Verbindungen (lokal und “schnell”). Wir formulieren das Ausfallsicherheitsmodell, stellen Backup-Routenschemata vor und beweisen ihre Ausfallsicherheit für spezielle Netzwerke.
- Verkehrspartitionierung. Die Kommunikation zwischen virtualisierten Knoten kann eine erhebliche Verschwendung der Bandbreite zwischen Servern darstellen. Eine geeignete Partitionierung dieser Knoten kann den Datenverkehr zwischen den Servern reduzieren. Die Annäherung an die optimale Partitionierung ist eine rechnerische Herausforderung, da das zugrunde liegende Kommunikationsmuster nicht a priori bekannt ist. Wir stellen Online-Algorithmen vor, die proaktiv den kostspieligen Inter-Server-Verkehr

in (kostengünstigen) lokalen Verkehr umwandeln, der auf der jüngsten Kommunikationshistorie basiert.

- Verkehrsteuerung durch Wegpunkte. Service Verkettung ist eine Technik zur Zusammenstellung benutzerdefinierter Netzwerkdienste unter Verwendung primitiver Funktionen. Ein Sicherheitsdienst kann beispielsweise aus mehreren Knoten bestehen: einer Firewall, einem DPI und einem Virenscanner. Die Berechnung kürzester Routen, die durch eine gegebene Menge von Knoten führen, ist ein NP-schweres Problem. Wir stellen Härtegrade und Algorithmen für einige spezielle Netzwerke vor, die das Problem überschaubar machen.

به یاد مادربزرگ مهربان و دلسوزم

In Memory of My Loving and Selfless Grandma

Bibliographic Note

Part of this thesis is already published in conference papers and the chapters of this thesis are based on the following papers:

Contents

Chapter 1

Introduction

Internet service providers and data centers are facing scalability and flexibility challenges due to the rapid expansion of internet applications. The addition of billions of internet-enabled devices, the increasing demand for high quality live and real-time communication services, and the advent of high-capacity mobile generations (5G and 6G) are few reasons that indicate facing such challenges is inevitable. Thanks to rapid advancements in processing technologies, communication networks have evolved considerably more flexible than the rigid and hardware-centric ones a decade ago, which do not scale easily for today's massive traffic and quality standards. Hence, it is imperative to exploit software-based flexibilities towards developing more agile and cost-efficient networks. In particular, we intend to facilitate the realization of networks that can react to unanticipated events and local changes (e.g., failures) autonomously, can be adapted to new traffic demands or new policies quickly, and can be scaled with minimal physical rewiring and resource provision. The existing network technologies already offer flexibilities that are promising for the development of such networks. In particular, flexibilities that are offered by software-based and programmable networks are not fully exploited yet, which is partly the task of algorithm developers and the interest of this thesis.

One source of flexibility is the *virtualization* process, which refers to the replication of hardware resources in functionally equivalent software units. Network resources such as firewalls, load balancers, caches, optimizers etc., also known as *network functions*, are traditionally delivered in the form of propriety hardware units called *middleboxes* [39]. A *Virtual Network Function (VNF)* [59] is a software that emulates a specific middlebox, and is logically decoupled from the underlying hardware. Therefore, a pool of VNFs may share the same physical server, implying flexibility in the management and scaling of networks. For instance, deploying, upgrading, and migration of VNFs can be simply a matter of reconfiguring a management software; tasks that otherwise are done manually, possibly inefficiently and error-prone.

Another source of flexibility is the paradigm *software-defined networking (SDN)*

[125]. The SDN's architecture decouples routing decisions from the underlying packet forwarding mechanism which merely executes the routing decisions. Under SDN, routing decisions are made by a logically centralized *controller* software that has a global view of the network. SDN and its programmable interface bring more flexibility in routing especially for traffic engineering and failure recovery. In addition, we extensively leverage routing flexibilities made possible by the standardized technique *segment routing (SR)* [72, 185]. Under SR, the journey of a packet is logically subdivided into consecutive segments and the packet is forced to reach the end of each segment as an intermediate stop, before eventually arriving at its final destination. SR enables a finer control on the final route taken by a packet without modifying the underlying routing mechanism.

This Thesis in a Nutshell. We are given a network that supports SDN, VNF, and is equipped with specialized software components for managing the placement of VNFs on physical servers. We refer to this abstract combination as a *softwarized network* [102, 118]. This thesis focuses on algorithmic challenges motivated by softwarized networking sharing a common goal: *traffic control.* Namely, we focus on practically relevant combinatorial problems concerning the steering of network traffic for specific performance objectives.

1.1 Objective and Methodology

Objective. Large scale networks are expected to support predictable traffic congestion and fault-tolerance under natural resource limitations (e.g. bandwidth), under user-driven traffic demands, or under unforeseen failure scenarios. Achieving this goal necessitates the development of simple and practical algorithms with provable guarantees. In particular, we are interested in algorithms that operate *locally* during runtime, i.e., within the scope of a network node (router), while restricted to a limited or possibly no global knowledge of the network. However, the algorithm may entail a pre-computation (or pre-runtime) phase during which it has access to the global view of the network. Often the pre-computation phase alone poses a significant challenge, and it is the domain of this thesis.

Methodology. Most algorithms developed in this area are heuristic-based, and they are often backed by a purely empirical analysis based on experiments on specific (synthetic or benchmark) datasets. Despite many successful heuristics, still confident understanding of algorithms through rigorous formal analysis is crucial for guaranteed performance and gaining insights into the theory behind empirical observations. This work insists on rigorous analysis of algorithms and occasionally complements it with empirical results to demonstrate their applicability. We abstract and model a network problem as a combinatorial problem, often a graph problem. Then we explore algorithms that can provably address the problem on a predefined infinite range of inputs, such as a class of graphs or an infinite sequence of requests.

1.2 Motivation: Softwarized Traffic Control

Virtualized Networking and SDN technologies are relatively recent, and the majority of the research in these areas have been focused on the design and implementation challenges. Flexibilities implied by softwarized networking have also motivated plenty of computational challenges, especially on the design of practical algorithms with provable guarantees. This thesis touches on a couple of traffic control challenges motivated by softwarized networking summarized as follows.

1. *Traffic Engineering.* A large flow of traffic may over-utilize (or congest) parts of a network, while the rest is under-utilized. Network operators tune link weights to influence the shortest path routing for balanced traffic distribution. However, shortest path routing cannot distinguish between flows to different destination nodes. A remedy to this shortcoming is the segment routing (SR) technique. SR on top of shortest path routing enables per-flow traffic steering. We introduce a new optimization approach that combines two traffic engineering techniques, link-weight, and segment optimization, and we prove its competence in Chapter 2.

2. *Fast Traffic Rerouting.* Link failures if not handled, can lead to packet losses and, therefore, an unreliable network. A mechanism known as *Fast ReRouting (FRR)* is widely used for failure recovery with minimal runtime computation. Once the traffic hits a faulty link, FRR is activated immediately, and packets are redirected to a secondary route. The rerouting occurs locally within the scope of the affected router, i.e., without any orchestration or centralized intervention and with negligible processing overhead. SDN facilitates updating routers with new rerouting instructions, and we exploit this flexibility to reroute failure-hit traffic in a provably reliable manner. We formally present a special resiliency model that supports a maximal number of link failures, and we provide resilient rerouting schemes under this model for certain classes of network topologies in Chapter 3.

3. *Traffic Steering Through Clouds.* Cloud service providers maintain a variety of network functions encapsulated in virtual machines placed on multiple physical machines running them. The traffic flow must visit at least two nodes: its source and destination, and possibly pre-specified intermediate middle-point nodes to satisfy specific policies (e.g., security). The amount of resources consumed by the traffic (e.g., bandwidth and time) can be controlled by optimizing the placement and the order of visiting endpoint and middle-point nodes. In this regard, we present two traffic control challenges as follows.

   I) *Traffic Partitioning:* Communications between nodes consume possibly a significant bandwidth between servers, or *inter-server bandwidth*, which may diminish after a carefully calculated placement of nodes. The optimal placement is not trivial due to the varying communication pattern. We aim to transform costly inter-server traffic into local (inexpensive) traffic within servers proactively in Chapter 4.

II) *Traffic Waypoint Routing:* A virtual network function (VNF) provides a specific service, e.g., firewall, proxy, cache, etc. An abstract service can be realized by chaining VNFs. For example, a security service may be composed of a firewall, deep packet inspection, and virus scanner VNF nodes. SDN streamlines the realization of such services by providing programmable interfaces for steering the traffic through these nodes flexibly. However, choosing a non-arbitrary (shortest and capacity respecting) route through given nodes is computationally challenging. We address this problem for bi-directed networks in Chapter 5.

1.3 Preliminaries

This thesis extensively relies on the *source routing* concept, where the source of a route dictates the entire route to all intermediate nodes of that route. In particular, we employ the *segment routing (SR)* [72] technology as the primary mechanism for enforcing a given route. With SR, the source of a route inserts an ordered list of "labels" (a label stack) into the packet header. An SR-label encodes an action (as to what to do with the packet), or it points to a node and link. In the latter case, labels instruct (router) nodes where to forward the packet next. We consider a simplified abstraction of SR for the sake of this thesis, where each label is only a node/link identifier. We assume that each node has a static global view of the network (not necessarily consistent with the actual topology) and they all possess a list of node labels, so that they can insert labels and act as the source of a segmented route. A node label serves as a *waypoint*, i.e., it is treated as an intermediate destination that is removed from the packet header (label stack) once it is reached. Link labels instruct the incident router to forward the packet via the specific link regardless of other header fields. We seldom use link labels as they are not always globally unique in practice. Once a packet carrying a label stack arrives, the router pops the topmost label if the label belongs to that router. Otherwise, the router forwards the packet to the destination indicated by the topmost label without modifying the stack, as in the normal (default) routing. SR offers the same functionalities of the more traditional path-enforcing technology MPLS, but in a simpler and more scalable way. The main advantage of SR over MPLS is that enforcing a new route does not require maintaining per-path information along the new route, as only the source node of the route needs to be informed about the route. Put differently, intermediate nodes of a route do not need to distinguish between different flows, and they forward a packet only by looking at its (next) waypoint (top of stack). The key constraint in SR is the number of labels encoded in the SR header, a small constant, and a limited resource that we must spend sparingly. Technical details of SR implementations are beyond the scope of this work, and we refer readers to [185] for a recent survey on segment routing.

1.4 Traffic Engineering

It is crucial for large-scale network service providers to maintain a predictable quality of service for static traffic patterns. A congested link is often the cause of unexpected delays and packet losses and thereby an impaired quality of service. Hence, it is necessary to engineer routes such that no link is overloaded with more traffic it can handle. We assume the predominant routing scheme that routes each packet along a shortest path to its destination. Network operators may influence these shortest paths by tuning link weights with the goal of distributing the traffic as evenly as possible. Segment routing (SR) [72, 75] is a recent technology and another means of influencing the shortest path routing. Using SR, individual traffic flows can be steered away from their (default) shortest-path route by introducing intermediate destinations or *waypoints*. We combine these two traffic engineering techniques into a *joint optimization* problem that improves link utilization by setting appropriate link weights and waypoints jointly. We first scrutinize the effectiveness of this optimization technique to answer the following research question.

Question 1. *To what extent a network benefits from applying the joint optimization in place of separate link weight and waypoint optimizations?*

We formulate the "benefit" as a ratio between optimal values of the joint problem and the individual optimizations, and we refer to it as the *optimality gap*. A large gap emphasizes the advantage of applying the joint optimization. We answer the question by synthesizing a class of networks for which the gap between the two optimization problems is a factor super-linear in the number of nodes.

1.4.1 Related Works

Traffic engineering by link-weight optimization was proved NP-hard by Fortz et al. [83, 84]. They also showed a gap linear in the number of nodes between the worst link utilization of the flow under ECMP and the optimal weight setting and the utilization under the optimal (multi-commodity) flow. Their gap analysis implies that splitting evenly can lead to a utilization worse than that of the optimal flow by at least a linear factor. We show that the same gap holds even when we replace the optimal flow with the flow solution of the joint weight and waypoint problem (Section 2.1). A similar gap analysis was presented in [154] comparing the maximum even-splitting flow that does not have to correspond to a weight setting and the optimal flow. We improve the gap by including a multiplicative logarithmic factor that holds also for the case of Fortz et. al. The other TE dimension is waypoint optimization. There have been several works, including a doctoral thesis on this front [16, 18]. They focus on the computational challenges of waypoint optimization rather than its effectiveness, which is the interest of this thesis. Section 2.1.3 elaborates on the related works on both of these dimensions.

1.4.2 Contributions

Chapter 2 presents the joint optimization problem and its gap analysis. Major contributions are as follows.

- Formal formulation of the gap ratio (Definition 2.3.1)
- A lower bound for the gap in $\Omega(n \log n)$ (Section 2.3)
- An upper bound for the gap matching the lower bound (Section 2.4)
- An approximation algorithm for the weight optimization problem and single source-target demands (Section 2.5)

1.5 Fast Traffic Rerouting

Network link components such as cables or network interface cards may fail to transfer packets for maintenance or unanticipated reasons such as cable cuts, hardware malfunction, human errors, and so on. An obvious way of dealing with failures is to update the entire network with the new (post-failure) topology information by running a slow distributed process known as *convergence*. "The convergence process increases router load, introduces outages and transient loops, and slows reaction to failures." [127]. A complementary convergence-free technique is *fast rerouting (FRR)* [171]: a process that temporarily takes control of the forwarding logic of a router upon an incident link failure until the convergence is fully executed. For practical reasons, FRR is required to execute within a twentieth of a second ($< 50ms$). Thus, it can depend only on the information available locally (the failed link, packet destination, incoming port etc.), and it must function without the global knowledge of all failures and preferably without any path (re-)computation. Under such constraints, all rerouting decisions must be (pre-)computed in advance and encoded in the forwarding tables of routers, or in short, a *static rerouting* scheme [1] must be installed.

Challenges of Static Resiliency

Once a packet hits a faulty link, we do not forward the packet to a functioning link, oblivious to potential failures that the packet might encounter later. Under an incorrect scheme, a second failure encountered later may cause a rerouting back to the first failure and thus an indefinite rerouting back and forth between the two faulty links; we refer to this undesirable condition as a *routing loop*. Hence, under arbitrary rerouting schemes, a failure scenario (i.e., a subset of failures) can create

[1]Here, "scheme" is any assignment of a unique backup route/next-hop to each link.

a permanent routing loop over two or more failures [104], which leads to waste of bandwidth and eventually packet drop.

First Challenge: Guaranteed Resiliency

We often assume the given network is k-connected. In the absence of the global view of failures, we resort to schemes that are provably resilient against the failure of any subset of up to k links (k-resilient). The feasibility and complexity of computing such schemes depend on the information available locally (at the affected router node), which depends on the specific resiliency model.

With and Without Header Rewriting. FRR mechanisms with a provable resiliency (without packet duplication) exist under two categories: with and without custom packet-header rewriting.
1) Header-rewriting techniques use additional header bits for encoding information on previously encountered failures, e.g., by explicitly writing a list of failed links into the packet header as in [127]. This information is used later upon encountering further failures down the route to the destination to avoid repeating the same failure and thus infinite routing loops. Although header rewriting allows for the *perfect resiliency* [109] (i.e., delivering packets as long as there is a path), it introduces extra complexity and is not always favorable in practice.
2) Without writing data into packet header and under the basic FRR model that uses only packet destination (to decide the next-hop), protecting against even single failure scenarios is not possible [126]. The remedy is to incorporate the incoming port (i.e., the link from which the packet arrives) into the rerouting mechanism. Applying the combination of packet destination and incoming port allows to protect against any number of failures that do not disconnect the underlying graph in all scenarios [41], and hence it relates resiliency to edge-connectivity. We refer to such guarantee of resiliency as *maximal resiliency*, which is weaker than perfect resiliency as it may not cover all failure scenarios where there still exists a path to the destination.

With and Without Carrying Failures. The FRR models used in this thesis fall under the first category (i.e. using header rewriting) in two different ways. In the first part of chapter 3, in Section 3.2, we consider a rerouting model that stores failure information in the packet header. Using failure-carrying packets, loop-free rerouting becomes almost trivial, as each node receiving such packets is able compute a path free from those failures. This convenience allows us to focus on the non-trivial challenge of realizing a given rerouting scheme, or more specifically, on segment routing. In particular, we show that the number of required segments grows linearly with the number of failures. Then in Section 3.3, we shift our attention to an FRR model that does not store failures in the packet header and assumes segment routing as the underlying rerouting mechanism. In contrast to the former section, here computing loop-free backup routes is challenging; when the packet hits a second failure, the router is not aware of the first failure; is it still possible to reroute without hitting either failures? Note that even though we do not use failure-carrying packets in this part, the mechanism still requires header rewriting because of segment routing specifications.

In the remainder, we sketch the resiliency model used in the second part of Chapter 3 in more detail. First, we clarify the distinction between two models of backup routes used in this chapter.

Backup Route Models. In Section 3.2, we use backup routes in its basic definition: a route that delivers packets from a node incident to a failure to their destination avoiding all previously encountered failures. In Section 3.3, a backup route is a detour around the incident failed link, and we refer to it as a backup path (a.k.a replacement path). Specifically, a *backup path (BP)* is an alternative route that delivers packets to the opposite endpoint of the (incident) faulty link. In other models, often the packet is rerouted without preserving its original node traversal. In contrast, the backup path model preserves original node visits, which is advantageous in service chaining applications (see Section 1.7).

Resiliency via Backup Paths. Providing backup paths for single link failure scenarios is trivial, e.g., by assigning to each link the shortest path connecting its endpoints without using the link. However, when multiple links are expected to fail simultaneously, is it not fully known how to compute backup paths that guarantee delivery under any failure scenario especially without any global information on failures. To see the challenge, consider an arbitrary BP assignment in which two links mutually contain each other on their backup paths: if both links fail then packets keep bouncing back and forth between the two affected endpoints. Such schemes do no guarantee resiliency but may perform well in certain practical settings, and we evaluate one of these schemes empirically in Section 3.2.5. Furthermore, resiliency in this model is limited to edge-connectivity. That is, in a k-(edge-)connected network, up to $k-1$ (simultaneous) failures can be tolerated. If a BP scheme is resilient to all $k-1$ link-failure scenarios then we say it is *maximally resilient.* This brings us to the following traffic control challenge.

Question 2. *How to assign a backup path to each link such that packets arrive to their destinations under any number of simultaneous link failures up to the network's edge-connectivity?*

We address this question in Section 3.3 for special structures, including hypercubes, and we prove our schemes are maximally resilient.

Second Challenge: Realizing Backup Routes

At this point, we assume a backup route scheme is given. Now the challenge is to ensure that all nodes execute the rerouting scheme correctly, even when they are not incident to a failed link. Recall that when FRR is activated, we require nodes to forward packets along the backup route and not along the default route (when the two routes do not share the same next-hop). In order to enforce rerouting along a backup route, an FRR model must provide a mechanism for influencing the forwarding operation at every intermediate node. A naive approach is to let each node compute a backup route using the header information; this inflicts a large computational overhead to routers. An alternative is to perform the backup route computation only on the router incident to a failed link, and insert this route entirely in the packet header for guiding the subsequent nodes (as in [127]). The latter technique ensures a correct routing at each hop at the expense of a large spacial overhead.

Question 3. *Given a backup route, how to ensure that each of its nodes forwards the packet to the next node on this route?*

The forwarding rules installed on routers dictate the default (primary) routing in a distributed manner. Deviating from the default routing is not trivial and requires auxiliary mechanisms such as MPLS or more conveniently, the segment routing which is used in the real-world FRR implementation TI-LFA [85] and our main point of interest. Realizing backup paths in a network is feasible via existing routing technologies such as MPLS and more conveniently, via segment routing

Segment Routing for Resiliency. Our FRR models assume a network that supports segment routing (SR). We first (in Section 3.2) introduce a mechanism using failure-carrying packets that extends the existing SR-based FRR mechanism TI-LFA [85]. Then in Section 3.3, we switch to a rather minimalistic model that does not insert any failure information into the packet header and relies on SR for realizing backup paths.

1.5.1 Related Works

Static resiliency against two or more link failures is impossible if we only match the destination. This is not the case with matching the input port [126] (the last functioning link traversed before hitting a failure). Moreover, matching only the input port is not sufficient for resiliency against 2-link failures [44]. To overcome these impossibility bounds, rerouting schemes match both the input port and the destination. Alternatively, writing limited extra data in the packet header (a.k.a. header rewriting) is also sufficient for a higher resiliency. Prior works under the maximal and perfect resiliency models are as follows.

Maximal Resiliency. Chiesa et al. [41, 44, 45] presented rerouting schemes that require packet-header rewriting and proved that using only three bits is sufficient for maximal resiliency on any graph. They also presented maximally resilient schemes for the special class of planar graphs, hypercubes, grids, and complete graphs, without header rewriting and by matching input port and destination. In their scheme, a failure-hit packet is rerouted along an *arborescence*: a directed spanning tree rooted at the packet's destination. A collection of arc-disjoint arborescences and a specific ordering of them are computed and encoded in the forwarding tables. Upon hitting a (subsequent) failure, the packet is rerouted along the next arborescence in the given ordering. In contrast to our model, theirs is not committed to delivering the packet to the opposite endpoint of the faulty link. As a result, their schemes do not guarantee to visit all nodes of the original route, which is a disadvantage for service chaining applications.

Perfect resiliency. The goal in this stronger model of resiliency is to compute a local rerouting scheme that delivers packets to their destinations as long as they are reachable. Notice that in contrast, a maximally resilient scheme may fail to deliver a packet when the number of failed links is larger than the edge-connectivity, even when failures do not disconnect the destination. See [44, 67, 68, 77] for extensive details. This thesis does not address perfect resiliency and focuses only on maximal resiliency.

1.5.2 Contributions

The following list summarizes the contributions of Chapter 3.

- Section 3.2 sketches resiliency via segment routing and formally describes the TI-MFA protocol, its analysis and empirical evaluation for multiple link failure scenarios.
- Section 3.3.3 presents the formal description of maximal resiliency and its ILP formulation.
- Section 3.3.4 presents maximally resilient Backup path schemes for the class of Hypercubes, Complete graphs, torus, and grids, including the proof of resiliency for each case.

1.6 Traffic Partitioning

A network service may be composed of multiple VNFs each running in a dedicated virtual machine (VM), and they may frequently communicate with each other in order to perform a task. Consider a data center with several physical server machines, each having a fixed capacity for hosting guest VM nodes. Pairs of communicating nodes that are deployed on different servers incur an inter-server communication cost that is significantly larger than the cost when they are collocated on the same server, i.e., when they communicate locally. On way of reducing the inter-server traffic is to replicate VM nodes that are frequently requested on different servers. This method is limited by the processing capacity of servers. An alternative and capacity-respecting strategy (without replication) is to migrate frequently communicating nodes to the same server while respecting server capacities. Migrating nodes is a highly costly operation and must be executed conservatively. Moreover, by realistic assumptions, the sequence of communication requests may not be known to algorithms in advance. Specifically, we assume only past requests are fully known and future requests arrive one at a time. Maintaining a trade-off between the cost of inter-server traffic and the migration cost is challenging, not knowing the future requests. Migrating nodes too early may turn out wasteful (of bandwidth) on the hindsight, while postponing migrations may lead to a huge waste of inter-server bandwidth. We restate the general challenge in the following question.

Question 4. *When to collocate two nodes that communicate frequently, and on which server should we place them in a way that the cost of migration and communications is minimized? To what extend can we minimize these costs without knowing future requests?*

We address these questions in special settings that simplify the problem, e.g., when it is assumed that a prefect partitioning exists on which no inter-server communication ever occurs or when the capacity of each server is exactly two. The latter variant is also known as the online rematching problem in the literature [29] as it is related to the classic perfect matching problem.

1.6.1 Related Works

The Balanced Graph Partitioning problem [10] is the offline version of our problem where the entire communication graph (or pattern) is given, and the task is to find a minimum multi-cut that partitions the nodes into equal-size clusters. The offline problem is NP-hard and is not a subject in this thesis. Online algorithms are not privileged with the entire communication pattern beforehand, and therefore they resort to any information available from the past communication history. An online algorithm may migrate nodes "dynamically" in order to adapt to a communication pattern which is not necessarily a static pattern. These challenges are formulated in an online problem known in the literature as the *online balanced repartitioning (OBP)* [19, 22].

Avin et al. [19] presented online algorithms and complexity bounds for the OBP variant where servers are augmented with limited additional space and the communication pattern is unconstrained. For the variant that assumes a specially constrained communication pattern called the *perfect partitioning*, Henzinger et al. [105, 106] provide several results with server augmentation. For the most part of this thesis, we assume variants without augmentation. Section 4.1.3 elaborates on related works in more detail.

1.6.2 Contributions

Table 4.1 summarizes the results in this chapter. The contributions are as follows.

- Theorem 4.3.1 provides a lower bound for the competitive ratio of any online algorithm for the learning variant without capacity augmentation. Theorem 4.4.1 generalizes this bound to the general problem.

- Section 4.3.3 presents an asymptotically optimal algorithm for the learning variant matching the lower bound.

- Section 4.4.4 presents a (strictly) 6-competitive algorithm for the case of $k = 2$ (proven in Theorem 4.4.6), which improves the (non-strict) 7-competitive algorithm of Avin et al. [19].

1.7 Traffic Waypoint Routing

Network service providers often require the traffic to pass through a sequence of network functions (virtual or physical) for security or performance reasons. This is a practice widely known as *service chaining* [143] and can be realized by software-defined networking [32, 192]. A graph-theoretic abstraction of service chaining brings us to the known waypoint routing problem that we revisit in Chapter 5.

In the general *waypoint routing problem (WRP)* [7–9], we are given a network and its link capacities, a pair of source-target nodes with a unit-size demand between them, and a set of k special nodes or *waypoints*. The goal is to find a shortest route connecting the source and the target that visits all waypoint nodes respecting link capacities.

We attend a special case where all link capacities are 1.

That is, the walk may not use a link more than once. We consider two variants of this problem: ordered and unordered. In the *ordered* variant, the first visit to waypoints must occur in a specific given order (as a permutation). The order of visits is arbitrary in the *unordered* case. The complexity of this problem has been partially uncovered is several recent works. We provide further insights into its complexity by tackling the following question.

Question 5. *For what graph families the (ordered or unordered) waypoint routing problem admits polynomial-time exact or approximation algorithms?*

This thesis focuses on *bidirected graphs*: directed graphs in which a link (u, v) exists if and only if the link (v, u) exists.

1.7.1 Related Works

The waypoint routing problem has been explored in several recent works. The ordered variant is closely related to the family of Edge-disjoint Paths problems, and the unordered variant is related to TSP problems. We described known results for each variant separately.

Arbitrary Ordering. The unordered WRP is NP-hard on general graphs and for an arbitrary number of waypoints, since it generalizes the metric Traveling Salesman Problem (TSP) as follows. Given an instance of the metric TSP, set all link capacities to 1, set all nodes as waypoints, obtain the WRP walk using any node as its start and endpoints and shortcut all repeated node occurrences. The resulting walk is the shortest tour visiting every node exactly once and hence an optimal solution for the metric TSP. An alternative NP-hardness proof is presented by Amiri et al. [9]. They reduce from the Hamiltonian Cycle problem on special graphs of degree 3, and they show that the walk computed by WRP never repeats a node, and hence it is a Hamiltonian tour. They also present polynomial-time algorithms that compute the shortest feasible (capacity-respecting) route through an arbitrary number of waypoints on graphs of constant treewidth. For general graphs, they infer a polynomial-time randomized algorithm from an existing result on the shortest cycle problem for a logarithmic number of waypoints, as well as a deterministic algorithm for a super-constant number of waypoints.

Fixed Ordering. In the ordered variant, routes between consecutive waypoints must be link-disjoint; hence it is closely related to the classic edge-disjoint paths [158] and k-cycle [30] problems. In contrast to our problem, the disjoint path problem does not allow node repetition and does not require consecutive paths, and the k-cycle requires a closed walk. The feasibility problem (i.e. without optimality) is NP-complete for an arbitrary number of waypoints and both directed and undirected graphs. The feasibility problem is NP-complete in directed graphs even with a single waypoint [4]. However, with constant number of waypoints and undirected graphs, it becomes tractable by a reduction to the edge-disjoint path problem [7] and using the seminal result of Robertson and Seymour [158]. The optimal (shortest) route can be computed in polynomial time on trees (trivially), and on undirected graphs of constant treewidth when the number of waypoints is constant [4]. For an arbitrary number of waypoints, they show the feasibility problem is NP-complete for a family of undirected graphs of treewidth at most 3 (series-parallel graphs), with the exception of outerplanar graphs (treewidth at most 2). The complexity of the ordered variant is still unknown for other undirected graphs, i.e., special graphs of super-constant treewidth.

1.7.2 Contributions

We contribute to Question 5 with several results. We show that the unordered WRP on bidirected graphs is closely related to the metric Traveling Salesman Problem (TSP) and also to its generalization, the subset TSP [122]. We summarize the results for each of the order and unordered variants separately. We derive several complexity results from these relations summarized as follows.

Unordered Bidirected WRP

- Constant-factor approximation algorithms via reductions to the Steiner Tree problem (Cor. 5.2.5) and metric TSP (Cor. 5.2.8).
- PTAS infeasibility and approximation lower bound via reduction from metric TSP (Cor. 5.2.7).
- Unordered bidirected WRP is always feasible (Cor. 5.2.4).
- Exact algorithm for a sub-logarithmic number of waypoints (Thm. 5.2.3).
- Exact algorithm for a logarithmic number of waypoints via reduction to subset TSP (Cor. 5.2.9).

Ordered Bidirected WRP.

- Polynomial-time algorithm via reduction to the edge-disjoint paths on bidirected graphs (Thm. 5.3.1).
- NP-hardness via reduction from EDP on bidirected graphs (Thm. 5.3.2).
- Polynomial-time algorithm for cactus graphs (Thm. 5.3.5).

1.8 Overview of this Thesis

We begin with a traffic engineering (TE) challenge in Chapter 2, and we refer it as the joint optimization. Section 2.3 shows the benefits of the joint optimization over two classic TE strategies. Sections 2.4 to 2.6 provide algorithms and theoretical upper bounds for the joint optimization.

Chapter 3 touches on a few network resiliency challenges in two major parts. The first part (Section 3.2) employs segment routing for the restoration model where packets can "remember" previously encountered failures. This resiliency model is already established in the industry but is limited to single failure scenarios due to its complexity overheads. This section extends it to arbitrary failure scenarios and observes its overhead both theoretically and empirically. In the second part (Section 3.3), we consider the model where packets cannot remember past failures. We provide resiliency solutions that are provably effective even under such a challenging model.

Chapter 4 takes on a different approach to traffic control. Traffic partitioning concerns relocating the endpoints that generate significant traffic closer to each other to save bandwidth. Section 4.3 considers a restricted model, where the traffic adheres to a fixed pattern and provides lower and upper bounds for this model. Section 4.4 considers the problem for arbitrary traffic patterns and constant cluster sizes. Particularly, Section 4.4.4 presents an online algorithm for a special case known as the online rematching problem.

Chapter 5 visits the waypoint routing problem, provides hardness results, and mainly focuses on a few practical cases where the problem admits simple polynomial-time algorithms.

Chapter 2

Traffic Engineering

Most ISPs use sophisticated traffic engineering strategies based on link weight optimizations to efficiently provision their backbone network and to serve intra-domain traffic. Traffic is traditionally split among the shortest weighted paths using ECMP. An additional dimension for optimization arose recently in the context of segment routing: traffic can be steered away from congested shortest paths by inserting intermediate destinations, so-called *waypoints*. This chapter investigates the benefits of jointly optimizing link weights and waypoints for traffic engineering analytically and empirically. In particular, we formulate the joint optimization problem and formally quantify its advantages over link-weight and waypoint optimizations separately using a rigorous analysis. We also present an efficient joint optimization algorithm and evaluate its performance in realistic and synthetic scenarios.

2.1 Introduction

Traffic engineering (TE) is a fundamental task in communication networks. To optimally use their infrastructure and avoid congestion, Internet Service Providers (ISPs) employ sophisticated algorithms to steer intra-domain traffic through their network. Many innovations in networking over the last years were at least partially motivated by the desire to improve traffic engineering [64].

Traditionally, traffic routes can be influenced only fairly indirectly by adapting *link weights*: Routing is based on the Equal-Cost-MultiPath (ECMP) protocol, in which flows are split at nodes where several outgoing links are on shortest paths to the destination, using per-flow static hashing. Thus, by changing link weights, shortest paths can be adjusted accordingly. While several clever algorithms are known today to optimize such link weights [84], such strategies provide relatively limited control over the paths taken by flows.

Segment routing (SR) [72, 74, 75, 185] has recently introduced a powerful opportunity to optimize traffic engineering along an additional dimension: by specifying one or multiple waypoints in the packet header, traffic can be routed around potentially congested links. In particular, given a waypoint w, traffic from a source s to a destination d, is not anymore restricted to a shortest st-path, and it is routed along a shortest path from s to w (the first segment) and then from w to t (the second segment). Segment routing hence provides two knobs for traffic engineering: the link weights in the network and the sequence of waypoints to be visited along the way.

The benefits of segment routing have been demonstrated empirically in many scenarios and have received significant attention for traffic engineering [16, 25, 100, 169, 185]. [1] However, relatively little is known today about the fundamental algorithmic problems arising from optimizing link weights and segments jointly. This chapter studies the benefits of traffic engineering mechanisms for jointly optimizing link weights and waypoints. In particular, we aim at an analytical understanding of the improvements possible by joint optimization compared to optimizing link weights and waypoints independently. For example, we will show that the usefulness of waypoints critically depends on the given weight setting and inappropriate weight settings can render waypoint optimization ineffective. Accordingly, we study algorithms for joint optimization.

2.1.1 Background

Traffic engineering objectives typically revolve around network utilization, which is also the focus of this chapter. In particular, we are interested in the maximum link utilization (MLU): the ratio obtained by dividing the load of a link by its capacity is called *link utilization*, and MLU is simply the largest such ratio over all links.

When flows are constrained to shortest paths (OSPF) and even-splits (ECMP), the maximum link utilization MLU can be significantly larger than the optimal utilization feasible without these constraints by a factor linear in the number of nodes [83].

Traffic Engineering (TE) under OSPF and ECMP. The open shortest path protocol (OSPF) [110] is a routing protocol for the IP layer, widely used within a single domain (e.g., an ISP, an enterprise, or a data center). Under OSPF, a packet is always routed along the shortest path to its destination. Hence, links weights (a.k.a. link costs) determine the actual route taken by a packet, and one can influence the distribution of traffic's load across a network by tuning these weights. Under OSPF, there might be multiple (shortest path) next-hops available from a node, which is an opportunity to split the traffic over multiple paths and prevent congestion. OSPF is often installed together with Equal-Cost-MultiPath (ECMP) routing [37], the local strategy of splitting the traffic evenly between all shortest paths available at a node. We assume a fine-grained splitting model, where flows split at the packet level [53].

[1]See [185] for a recent survey on segment routing.

TE via Link-Weight Setting. Tuning link weights for congestion control is a common TE technique, where the objective is often to optimize (a function of) network resources. Link weights (as positive real numbers) determine whether a link is on the shortest path to a node or not, and they are computed offline by network operators. Link weights are often chosen to induced shortest paths that do not congest any link beyond a tolerable factor of its capacity. This practice is known as *link weight optimization (LWO)*, which is generally NP-hard even for constant-factor approximation and the case of a single source-destination pair [43, 83]. Henceforth, in practice, link weights are computed using heuristics. One such heuristic (recommended by Cisco [49]) is to assign a weight to each link proportional to the inverse of its capacity.

TE via Waypoint Setting. Given a set of demands, setting link weights alone is not always sufficient to achieve a balanced load over all links, as will also show in this chapter (§2.3). An attractive solution enabled by segment routing is to insert intermediate destinations and force traffic flow to reach them in a specific order before arriving at its final destination. We refer to this technique as *waypoint routing*, and we refer to the problem of finding appropriate waypoints as *waypoint optimization (WPO)*. Informally, given a demand matrix, for each demand, WPO decides whether to insert waypoints for that demand and if so, it determines which of the nodes to use as waypoints for this particular demand.

TE via Joint Weights and Waypoint Setting. This chapter is interested in the optimization problem obtained by combining the two previous dimensions, link weight and waypoint optimizations. The motivation behind this approach is to alleviate shortcomings with LWO and WPO by joining their benefits; 1) LWO cannot optimize link weights for each demand independently of the other demands, while waypoints can be applied to each demand separately. 2) The effectiveness of WPO depends on the given link weights.

As we will show, WPO may perform poorly when link weights are set arbitrarily or even when they are given by standard settings commonly used in practice. We consider three such weight settings: uniform weights, the inverse of capacities, and optimal weights.

2.1.2 Contributions

We present analytical and empirical insights into the algorithmic opportunities of jointly optimizing link weights and waypoints for traffic engineering. We make the following contributions:

The Optimality Gap. We show that the joint optimization of weights and waypoints is provably competitive against the separate optimizations (§2.3). We formally define a notion of competitiveness between the two TE strategies, referring to it as the *optimality gap*. We advocate the effectiveness of the joint optimization by comparing its network utilization to the utilization under link weight and waypoint optimizations separately. We show that their joint optimization can improve network utilization by a factor in $\Omega(n \log n)$, where n is the number of nodes (Theorem 2.3.15).

We provide an upper bound for the gap in $O(n \log n)$ (§2.4), which implies our LB is tight on general networks (i.e., general graphs and capacities) and single source-target demands. We prove that the gap does not exist under uniform capacities and single source-target demands (Theorem 2.4.2). Additionally, the gap does grow by n, if the source and target are sparsely connected, e.g., when they are connected by a constant number of paths with disjoint capacities (Theorem 2.4.3).

Approximation Algorithm for Weight Optimization. As part of our gap analysis, we present an algorithm (§2.5) that computes a link weight setting that minimizes the maximum link utilization approximately. To the best of our knowledge, this is the first polynomial-time approximation algorithm for this problem on general directed networks (although single source-target demands), with a provable factor that depends on the number of nodes (in contrast to [154] which restricts capacities to a finite set of integers and the factor depends on the size of this set). Our approximation factor also serves as an upper bound for the extend of MLU improvement achievable in the joint optimization when compared to separate single optimizations (Corollary 2.4.4). Moreover, it shows that our example captures the worst case, i.e., the gap is tight.

Heuristic Algorithm. Since the general problem is NP-hard, in §2.6, we present a heuristic that extends the local search algorithm introduced in [83] by combining it with a greedy waypoint setting algorithm. We evaluate the quality of our algorithm on a variety of real-world topologies.

Empirical Gap Observation. We provide a mixed-integer linear program (MILP) formulation of the joint optimization problem available in [69]. We use the formulation to demonstrate the TE gap on small examples. For large networks, we run our heuristic and compare the resulting MLU to that of standard weight settings and computed by local search [83].

Artifacts. To contribute to the research community, ensure reproducibility, and support future research in this area, we release all our experimental artifacts and implementations as open source together with this chapter [69].

2.1.3 Related Work

Traffic engineering is an evergreen topic in networking. Besides adjusting link weights in IP networks, traffic engineering can also be performed using MPLS [190] or centralized controllers as in software-defined networking [2]. We employ segment routing [72], which is a relatively recent TE approach.

Fortz and Thorup in [83] showed that OSPF with ECMP can result in an MLU that is larger than the optimal feasible MLU under arbitrary flows by a factor in $\Omega(n)$ where n is the number of nodes. They also prove the NP-hardness of LWO and present a "local search" heuristic. We first show a linear gap between LWO and Joint (§2.3) using a similar network construction, and then we present a stronger construction (§2.3.5) that improves the gap by a logarithmic factor. Generally, it is known that weight setting is NP-hard to approximate within any constant factor even with single source-target demands [43, 154]. We complement this inapproximability result with a polynomial-time approximation algorithm and a provable approximation guarantee.

Chiesa et al. [43] provided impossibilities for several weight optimization problems. In particular, they showed that LWO is NP-hard on general topologies and even on the special class of hypercubes. They presented one positive result on special topologies known as "folded Clos networks": an ad-hoc algorithm producing weights and an optimal congestion (as in OPT). When capacities are restricted to a finite set of integers, Pióro et al. [154] show a constant factor approximation for the maximum LWO, while our approximation works with arbitrary real capacities.

Segment routing [72] for traffic engineering has been considered in, e.g., [16, 27, 100, 142], see [185] for a recent survey on SR. Moreno et al. [142] using linear programming and heuristics showed that a very limited number of stacked labels suffice to exploit the benefits of segment routing successfully. Using a worst-case construction, we demonstrate that SR cannot benefit TE under inappropriate weight settings, even with an arbitrarily large number of segments. Aubry et al. [16, 18] lay the algorithmic foundations of segment routing, considering different and related applications, however, primarily focusing on the waypoint optimization. However, we are not aware of any analytical quantification of the benefits of joint waypoint and weight optimization with segment routing.

2.2 Model and Problem Definition

We next define our key terminology and notations formally.

Network Instance. We model a network as a tuple $\mathcal{N} = (V, E, c)$, where V is a set of $n := |V|$ vertices (nodes or routers), E is the set of directed links (communication links) connecting nodes, and the mapping $c : E \mapsto \mathbb{R}^+$ assigns to each link $\ell \in E$ a capacity $c_\ell > 0$.

A *flow* is an assignment $f : E \mapsto \mathbb{R}^+$ that respects flow conservation constraints. The *load* on a link ℓ denoted by $f_\ell \geq 0$ is the amount of the (total) flow assigned to this link. Whenever the source and target of a flow f is clear from the context, $|f|$ denotes the total size of the flow emitting (entering) the source (the target). A *demand list* is a multiset denoted by $\mathcal{D}$ and consists of tuples $(s, t, d) \in \mathcal{D}$, where $s, t \in V$ are the *source* and *target* (destination) nodes of the demand, and $d \in \mathbb{R}^+$ is its size (i.e., required bandwidth). A *weight setting* $w : E \mapsto \mathbb{R}^+$ assigns a positive real $w_\ell > 0$ to each link $\ell \in E$. A *waypoint* is a node assigned to a demand, and serves as an intermediate destination for the flow of that demand. That is, the flow of that demand must reach the waypoint prior to reaching its final destination. A *waypoint setting* denoted by $\pi : V^W \mapsto \mathcal{D}$ assigns up to W waypoint nodes to each demand, where W is a given parameter.

TE Instance. The input to all our TE problems is a *traffic engineering instance (TE-instance)*, denoted by the tuple $\mathcal{I} = (\mathcal{N}, \mathcal{D}, \omega)$, where $\mathcal{N}$ is the given network, $\mathcal{D}$ is the given demand list with total demand size $D := \sum_{(s,t,d)\in\mathcal{D}} d$, and ω is the given weight setting.

Using these notations, we formally describe the TE objective and optimization problems considered in this chapter.

Maximum Link Utilization (MLU). Given a network $\mathcal{N}$ and a flow assignment f, the *utilization* of any link ℓ under the flow f is the ratio f_ℓ / c_ℓ. The *network utilization* is the maximum link utilization under f, that is, $MLU(\mathcal{N}, f) := \max_{\ell\in E} f_\ell / c_\ell$.

Even-Split Flow (ES-Flow). The aggregate flow that enters a node v and is destined to a node t may *split* over (a subset of) outgoing links of v. An *even-split* flow (ES-flow) either does not split at v or it splits evenly at this node, i.e., it may not split arbitrarily as in OPT.

ECMP Flow. If an even-split flow exiting a node v is constrained to traverse only the outgoing links of v that are on a shortest path to t, then we refer to it as an *ECMP-flow.*

Problem Definition

The minimal input common to all our optimization problems consists of a network $\mathcal{N}$ with n nodes and a demand set $\mathcal{D}$. We refer to a demand list where all demands share the same source-target pair (s, t) as a *single source-target* demand list. Next, we formally define each problem and its additional inputs separately.

Link-Weight Optimization (LWO). Given $(\mathcal{N}, \mathcal{D})$ as an input, the LWO problem computes a link weight setting such that the induced ECMP-flow minimizes MLU.

Waypoint Optimization (WPO). The input is $(\mathcal{N}, \mathcal{D}, w)$, where w is a weight setting. WPO selects an ordered set of up to W waypoints for each demand in $\mathcal{D}$. The ECMP-flow must reach each waypoint in the given order before finishing at t. WPO chooses these waypoints so that the MLU under the total flow is minimized.

Joint Link-Weight and Waypoint Optimization (Joint). Given the input $(\mathcal{N}, \mathcal{D})$, Joint computes a link-weight setting and a sequence of up to W waypoints for each demand such that MLU is minimized when flows are routed through shortest paths between consecutive waypoints.

The MLU for an instance $\mathcal{I}$ is denoted by LWO($\mathcal{I}$), WPO($\mathcal{I}$), and Joint($\mathcal{I}$), respectively, under an optimal weight setting, an optimal waypoint setting, and an optimal joint weight and waypoint setting. We omit $\mathcal{I}$ wherever it is clear from the context.

Note that in comparison to OPT, in the three latter problems, a flow must follow shortest path links, and it must split equally over all outgoing links that belong to a shortest path. Hence, LWO is equivalent to Joint when $W = 0$. By definition, if Joint and WPO are constrained to at most W waypoints (per demand), then

$$\text{OPT} \leq \text{Joint} \leq \min\{\text{LWO}, \text{WPO}\}. \tag{1}$$

Clarifying terminology on Joint We emphasize that Joint is not equivalent to applying LWO and WPO separately (e.g., sequentially), and it generalizes both of these optimizations, and hence, it is stronger. Joint is specifically the optimization problem of minimizing MLU over the Cartesian product of all link-weight settings and all waypoint settings. However, our heuristic approximates Joint using a sequential optimization.

The Optimal Flow (OPT). OPT is the multi-commodity flow problem with the objective of minimizing the MLU subject to link capacities (formulated in [69]). Note that OPT has no routing restrictions; it may assign a positive flow to any link, and it may split a flow arbitrarily at any node. We denote the optimal MLU of a TE-instance $\mathcal{I}$ under the optimal flow by OPT($\mathcal{I}$). Let f^* be a maximum (s,t)-flow. Observe that OPT $\geq D/|f^*|$.

Acyclic Maximum Flow. An *acyclic maximum* (s,t)*-flow* from a source s to a target t always exists: 1) take any maximum flow, 2) find a cycle and a link with the smallest flow value on this cycle, 3) subtract this value from the flow of every link of the cycle. 4) repeat from step 2 until the flow is acyclic. It is easy to see that the new flow has the same size and the algorithm terminates in polynomial time.

2.3 Optimality Gaps

In this section, we investigate the question of how competitive Joint is compared to LWO and WPO? To this end, we present several network instances where Joint yields a value for MLU noticeably smaller than the optimal values obtained from LWO or WPO separately. This will immediately imply that applying Joint is necessary in order to utilize the best of weights and waypoints. We assume single source-target demands throughout this section. We will study gaps (as ratios) between the optimal MLU feasible in Joint and the optimal values from LWO and WPO separately.

Definition 2.3.1. *Given a network instance $\mathcal{I}$, the* optimality gap *between Joint and each of the two problems is defined as ratios:*

$$R_{LWO}(\mathcal{I}) := \frac{LWO(\mathcal{I})}{\textit{Joint}(\mathcal{I})}, \text{ and } \qquad R_{WPO}(\mathcal{I}) := \frac{WPO(\mathcal{I})}{\textit{Joint}(\mathcal{I})}.$$

The ratio R_{WPO} depends on the given link-weight setting. We restrict the given weight setting to the special cases that are often used in practice. LWO does not take any weight setting as input.

Definition 2.3.2. *We refer to any of the following weight settings as a* standard weight setting.

- Unit weights*: the weight 1 set for every link.*
- Inverse of capacities*: the weight of each link equals the reciprocal of its capacity (*capacity^{-1}*).*
- *Optimal weights: weight settings optimal for LWO.*

We refer to general weight settings as arbitrary.

Definition 2.3.3. *We define the* TE gap *as the worst-case ratio between the best MLU obtainable from individual optimizations and the one from Joint. Formally,*

$$R^* := \max_{\mathcal{I}} \min\{R_{LWO}(\mathcal{I}), R_{WPO}(\mathcal{I})\}.$$

We may drop the instance $\mathcal{I}$ where it is irrelevant or clear from the context. Note that (1) implies $R_{\text{LWO}}, R_{\text{WPO}}, R^* \geq 1$.

Intuitively, a lower bound for the gap R^* indicates how far Joint can lower the MLU over the best of the other two optimizations. That is, a larger lower bound (i.e., a larger gap) emphasizes the necessity of applying Joint over separate optimizations.

Table 2.1 summarizes our findings for the gap in two major cases: i) when Joint is restricted to $W = 1$ waypoint per demand (Theorem 2.3.4), and ii) when Joint is restricted to 2 waypoints per demand (Corollary 2.3.16). Note that WPO is granted the extra privilege of using more waypoints as long as W is a constant.

The remainder of this section is devoted to showing the following statement.

Theorem 2.3.4. *There exist network instances with n nodes and demand lists that admits a TE gap $R^* \in \Omega(n)$ (Definition 2.3.3), when Joint and WPO are restricted to at most $W = 1$ waypoint per demand.*

Table 2.1: TE gaps for single source-target demands

Weights	**Capacities**	**TE-Gaps**	
Lower Bounds (Cor. 2.3.16)		$2 \leq W \in O(1)$	$W = 1$
arbitrary	arbitrary	$\Omega(n \log n)$	$\Omega(n)$
uniform	arbitrary	$\Omega(n \log n)$	$\Omega(n)$
capacity^{-1}	arbitrary	$\Omega(n \log n)$	$\Omega(n)$
optimal	arbitrary	$\Omega(n \log n)$	$\Omega(n)$
Upper bounds		$R^* \leq R_{\text{LWO}}$	
optimal	uniform	1, Theorem 2.4.2	
Thm. 2.4.3	arbitrary	$\lvert E \rvert$, Theorem 2.4.3	
optimal	arbitrary	$n \log n$, Corollary 2.4.4	

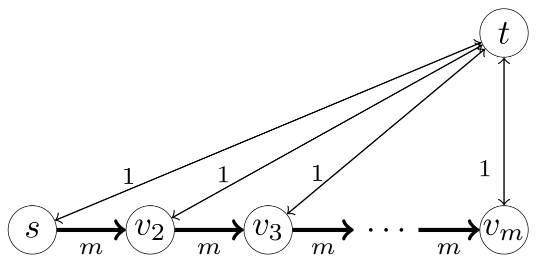


Figure 2.1: See TE-Instance 1. There are $m = n - 1$ unit-size demands from s to t. The optimal flow and JOINT are able to split one unit size flow away from the thick path at each node v_i. This is not possible under shortest path routing which causes a large gap in $\Omega(n)$ between JOINT and LWO.

We present a network instance that we use to prove Theorem 2.3.4. Similar examples are presented in [154] and [84] to show the gap between the optimal flow and the LWO for single demand cases. Here we adopt a similar example as in [154] and [83] and show that the linear bound holds for our gap ratio as well.

TE-Instance 1 (Figure 2.1). *Consider the network in Figure 2.1. It consists of n nodes $\{v_i, 1 \leq i \leq m\} \cup \{s, t\}$ where $s = v_1$, arcs $(v_i, v_{i+1}), 1 \leq i < m$ each with capacity m, and arcs $(v_i, t), 1 \leq i \leq m$ each with capacity 1. There are $m = n - 1$ unit-size demands from s to t. The optimal flow routes each demand via one of the paths with capacity 1, e.g., it may route the ith demand through s, v_i, t, which yields the optimal utilization $OPT = 1$.*

Next, we observe that in this instance the optimal MLU is feasible in JOINT using only one waypoint per demand.

Lemma 2.3.5. *Instance 1 admits* J<span style="font-variant:small-caps">OINT</span> *= OPT = 1 using up to one waypoint for each demand.*

Proof. We observe that the optimal flow is feasible also for Joint using the following waypoint and weight setting.

i) Set the node v_i as a waypoint for the ith demand. ii) Set the weight m for every link $(v_i, t), (t, v_i), 1 \leq i \leq m$, and iii) set the weight 1 for the other (horizontal) links. The flow of the ith demand runs through the unique shortest path to v_i, that is $s, v_2, \dots, v_i$, which has the cost i. The (unique) shortest path from v_i to t has the cost m and it consists of the link (v_i, t). Therefore, Joint assigns the (unit-size) flow of the ith demand to the unit-capacity path $s, v_2, \dots, v_i, t$. This flow assignment is identical to that of optimal flow and therefore Joint = OPT = 1. □

2.3.1 Optimizing with Link Weights

Optimizing (only) weights may lead to an MLU significantly larger than the optimal feasible MLU in Joint. We show that applying only LWO may result in an MLU that is worse than the optimal from Joint by a factor in $\Omega(n)$.

Consider Instance 1 (Figure 2.1). Any optimal even-split flow splits evenly at a node $v_i, i < m$. Assume w.l.o.g. that the optimal flow splits evenly at $v_1 = s$. As a result, half of the flow runs through the link (s, t) having the capacity 1, that is, the load $m/2$ on this link. Observe that splitting at the latter nodes $v_i, i \geq 2$ does not improve the MLU. We set the weight 2 for the link (s, t) and the weight 1 for every other link. This weight setting realizes the optimal even-splitting flow that splits only over the two shortest paths s, t and s, v_2, t. We conclude this case by comparing the MLU in LWO and Joint using Lemma 2.3.5 in the following statement.

Lemma 2.3.6. *The optimal link weight setting for TE-Instance* 1 *yield a maximum link utilization $LWO \geq (n-1)/2$.*

Proof. We observe that the size maximum feasible even split (s, t)-flow is 2 units which implies the claim for the total demand size $D = m = n - 1$. From v_m to t, only 1 unit of ES-flow is feasible. We reason that the size of the maximum ES-flow feasible via each $v_i, i < m$, is 2. From v_{m-1} to t, 2 units of ES-flow splits into two unit size parts, one part traverses the path v_{m-1}, t and the other part reaches t through the node v_m. In general, at $v_i, i < m$, only 2 units can be delivered via the neighbor v_{i+1}, i.e., if the flow does not split at v_i. If the flow splits at v_i then each of the two paths delivers 1 units. Therefore, the size of the maximum the ES-flow at $v_i, i < m$ is 2, whether it splits into two or not. □

2.3.2 Optimizing with Waypoints

Waypoint optimization on top of arbitrary or standard weight settings is not always competitive to Joint. That is, applying only WPO may result in an MLU worse than the optimal of Joint by a factor in $\Omega(n)$.

Lemma 2.3.7. *Given an arbitrary or a standard weight setting (Definition 2.3.2), there exists a network instance of n nodes on which the optimal waypoint setting from WPO subject to $W = 1$ waypoint (per demand) yields a maximum link utilization WPO $\geq (n-1)/3$.*

Proof. We show the claim in several cases of the given weight setting using TE-Instance 1 (Figure 2.1) denoted by $\mathcal{I}_1$.

Arbitrary Weights. Consider the weight setting for $\mathcal{I}_1$ that assigns the weight $\epsilon = 1/3$ for all links connected to t, i.e. to links $(v_i, t), (t, v_i)$ for every $1 \leq i \leq m$, and it assigns the weight 1 to every other link. Under this weight setting, the shortest path from s to t is the link (s,t). The shortest path from s to any node v_i is s, t, v_i. Therefore, with any choice of waypoints, the flow of every demand uses the link (s,t), which leads to WPO $= m = n - 1$.

Uniform Weights. Assume every link has the weight 1 in $\mathcal{I}_1$. Consider any assignment of zero or one waypoint to each demand. The flow that has v_3 as waypoint must split evenly over the two shortest paths s, v_2, v_3 and s, t, v_3 with equal costs of 2. All flows without a waypoint use the path s, t. Each flow using the node $v_i, i \geq 4$ as waypoint takes the shortest path s, t, v_i. Thus, an optimal waypoint setting may distributes the total load into at most three parts which assigns at least 1/3 of the load to the link (s,t) and therefore WPO $\geq m/3 = (n-1)/3$.

Inverse of Capacities. In this case, the weight of each link equals the reciprocal of its capacity. We transform the network instance into a new instance $\mathcal{I}_1'$ with $n = 2m + 1$ nodes (same demands). We replace each of the links (s, v_2) and (v_2, v_3) with m unit-capacity paths of two links as follows:

1. Add $2m$ new nodes $u^1, \ldots, u^m$ and $z^1, \ldots, z^m$.
2. Replace (s, v_2) with m paths where the jth path consists of unit-capacity links $\{(s, u^j), (u^j, z^j), (z^j, v_3)\}$ for $1 \leq j \leq m$.
3. Set capacity of every new link to 1.

Consider the weight setting for $\mathcal{I}_1'$ that sets the weight of each link to the inverse of its capacity. Every link (v_i, v_{i+1}), $i \in \{3, \ldots, m\}$, has the weight $1/m$, and every other links has the weight 1. For $i \geq 2$, any path $s, u^j, z^j, v_2, \ldots, v_i$ has a cost at least 3. Then the shortest path from s to any node $v_i, i \geq 2$ is s, t, v_i of cost 2. Therefore, with any waypoint setting in WPO, the link (s,t) having capacity 1 receives the entire load and WPO$(\mathcal{I}_1') = m = (n-1)/2$.

Optimal LWO Weights. We show even when the given weight setting is optimal for LWO, the MLU obtained from WPO can be significantly larger than that of JOINT.

By Lemma 2.3.7, the MLU under an optimal weight setting is is LWO $\geq m/2$, as it splits the flow into two parts at a node v_i. Consider the weight setting that assigns the weight 2 to the link (s,t) and the weight 1 to every other link. Under this weight setting, the flow of size m splits evenly at $v_1 = s$ into two parts of size $m/2$, overloading the link (s,t) by the factor $m/2$. Hence, LWO $= m/2$ and the weight setting is optimal.

We show that using a single waypoint per demand under the optimal weight setting does not improve the MLU beyond an additive factor. Assume w.l.o.g. that the optimal waypoint setting assigns v_i as the waypoint for the ith demand. The shortest paths to t and v_2 are respectively s,t and s,v_2, which are used by the first two demands without splitting. The third demand is destined to its waypoint v_3 and splits at s, since both paths s,t,v_3 and s,v_2,v_3 have the same cost of 3. The shortest path to v_i for every $i \geq 4$ is uniquely s,t,v_i. Thus, the link (s,t) receives a flow larger than $m-4$ and hence WPO $> m-4 = n-5$ in the optimal weight setting. □

2.3.3 Bounding the TE Gap

We can now show the TE gap claimed in Theorem 2.3.4.

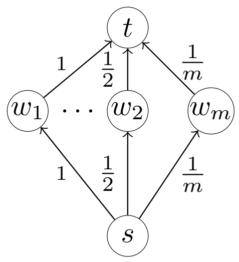
Proof of Theorem 2.3.4. We show the claim for TE-Instance 1 illustrated in Figure 2.1. First, we derive the individual gap ratios of Definition 2.3.1. From Lemma 2.3.5, we have JOINT $= 1$. Lemma 2.3.6 implies $R_{\text{LWO}} = \text{LWO}/1 \geq (n-1)/2$. Due to Lemma 2.3.7, under arbitrary and standard weight settings (Definition 2.3.2) the instance admits a gap $R_{\text{WPO}} = \text{WPO}/\text{JOINT} = \text{WPO}/1 \geq (n-1)/3$, when both JOINT and WPO are constrained with $W = 1$. Hence, by Definition 2.3.3, we conclude the claimed TE gap $R^* \geq (n-1)/3 \in \Omega(n)$. □

2.3.4 The Special Case of Uniform Capacities

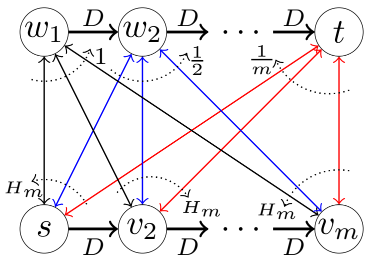
So far we considered instances with arbitrary (i.e., non-uniform) capacities. We will show later (in Theorem 2.4.2) that gaps larger than 1 do not exist in the special case of uniform capacities and single source-target demands. In particular, we show when all link capacities are equal and all demands share the same source-target pair then LWO = JOINT = OPT, which implies $R_{\text{LWO}} = 1$. The gap R_{LWO} can be larger than 1 when there are demands with arbitrary source-target pairs.

Theorem 2.3.8. *Under uniform capacities and arbitrary source-target pairs, there exists an instance with n nodes where the MLU is in $\Omega(n)$ under either of LWO or WPO, when the latter is restricted to one waypoint and standard/arbitrary weight settings.*

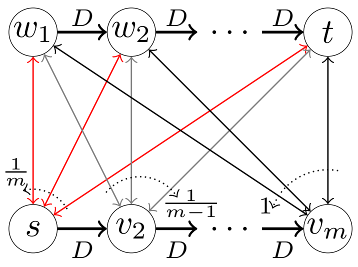
Proof. Consider the TE-Instance 1 (Figure 2.1) and denote it by $\mathcal{I} := ((G,c),\mathcal{D})$. We construct a new instance $\mathcal{I}' := ((G,c'),\mathcal{D}')$, where c' is the uniform capacity assignment that assigns the capacity m to every link, and $\mathcal{D}'$ is a demand list generated as



(a) See TE-Instance 2. The gap R_{LWO} is in $\Omega(\ln n)$ given $m = n - 2$ demands from s to t with harmonic sizes.



(b) See TE-Instance 3. The gap R_{LWO} is in $\Omega(n \log n)$ under m^2 demands with sizes that constitute m harmonic sets H_m, $m = n/2$, with total demand size D.



(c) See TE-Instance 4. Demands are identical to 2.2b. The gap R_{WPO} is in $\Omega(n \log n)$ which holds for any constant number of waypoints in WPO.

Figure 2.2: Each instance has n nodes. Arc labels represent link capacities. Bidirected arcs represent two directed links in the opposite directions with equal capacities. All demands are between $s = v_1$ and t. The parameter m is in $\Omega(n)$. In each instance, JOINT = OPT = 1 using up to two waypoints per demand. In 2.2b, OPT and JOINT separate one harmonic subset of demands away from the thick path at each node v_i causing a large gap between LWO and JOINT, while in 2.2c, they separate a subset of equal-size demands away from the thick path at each v_i causing large gap between WPO and JOINT. In short, OPT and JOINT are capable of fine-grained control over individual paths unlike LWO and WPO.

follows. i) Initialize $\mathcal{D}' = \mathcal{D}$. ii) $\forall \ell = (u, v) \in G, c(\ell) < m : \mathcal{D}' = \mathcal{D}' \cup \{(u, v, m - c(\ell))\}$. That is, $\mathcal{D}'$ includes all demands in $\mathcal{D}$ and an additional demand between the endpoints of each link in $\mathcal{I}$ with capacity less than m. It is not difficult to see that under all link-weight settings that we use in our lower bound analysis, the (unique) shortest path between the endpoints of a link is the link itself. Therefore, under both LWO and WPO, each demand in $\mathcal{D}' \setminus \mathcal{D}$ is routed along the link that connects its endpoints. Hence, once we deduce the capacities occupied by these additional demands, the residual capacities are exactly those assigned by c (as in Figure 2.1). Thus, our lower bounds hold for $\mathcal{I}'$ via an analysis similar to that of Theorem 2.3.4. □

2.3.5 Amplifying the TE Gap

In this section, we introduce an instance that admits a TE gap $R^* = \Omega(n \log n)$ (Theorem 2.3.15). First, we introduce and analyze an instance that offers a logarithmic gap, and later, we combine it with TE-Instance 1 to obtain a new instance that admits larger gap.

TE-Instance 2 (Figure 2.2a)**.** *In Figure 2.2a, there are $m = n - 2$ demands from s to t, where demand sizes constitute the harmonic series $H_m := 1, 1/2, \dots, 1/m$. The size of the maximum (s,t)-flow through this network is $\sum_{1 \le k \le m} 1/k \approx \ln(m)$.*

Next, we analyze the optimal even-split flow for Instance 2.

Lemma 2.3.9. *Let f be any maximum even-split (s,t)-flow on Instance 2. The flow splits evenly over a subset of (s,t)-paths with capacities that form a prefix of H_m.*

Proof. Let f be the maximum ES-flow and P be the subset of paths s, w_j, t used by f. Equivalently to the statement of the claim, P consists of paths with consecutive capacities (in H_m) at least $1/j^*$ for some $1 \le j^* \le m$. Then, in a maximum ES-flow, the path $p^* \in P$ with the smallest capacity $1/j^*$ is saturated, the load on each of the paths in P is $1/j^*$, and the size of the ES-flow is $|f| = |P| \cdot 1/j^*$. Assume for contradiction that this is not the case, that is, for some $k < j^*$, P does not contain a path p with a larger capacity $1/k > 1/j^*$. Consider the set $P' = \{p\} \cup P \setminus \{p^*\}$. Let f' denote the ES-flow that splits evenly over all paths in P'. The smallest capacity in P' is at least $1/(j^*-1)$. Since $|P'| = |P|$, we obtain $|f'| = |P'| \cdot 1/(j^*-1) = |P| \cdot 1/(j^*-1) > |f|$, which contradicts f being a maximum ES-flow. □

Lemma 2.3.10. *In the network Instance 2, the size of the maximum even-split (s,t)-flow is 1.*

Proof. By Lemma 2.3.9, for some $1 \le j^* \le m$, the maximum ES-flow f splits over a subset of paths s, w_j, t, $1 \le j \le j^*$ with capacities that constitute the first j^* numbers in H_m. The smallest capacity of these paths is $1/j^*$ which yields $|f| = j^* \cdot 1/j^* = 1$. □

Next, we introduce a network instance that has Instance 2 as a substructure, and admits a gap $R_{\mathrm{LWO}} \in \Omega(n \log n)$.

TE-Instance 3 (Figure 2.2b)**.** *Consider the network in Figure 2.2b with n nodes and the parameter $m = n/2$. Capacities of links $(v_i, w_1), \dots, (v_i, w_m)$ form the harmonic series $H_m = \{1, 1/2, \dots, 1/m\}$. Equivalently, every link connected to w_j, $1 \le j \le m$, has the same capacity $1/j$. All links (v_i, v_{i+1}) and (w_i, w_{i+1}) have the capacity D. There are m^2 demands from $s = v_1$ to $t = w_m$. Demand sizes can be partitioned into m subsets, where each subset constitutes the Harmonic series H_m. Hence, the total demand size is $D = m \cdot \sum_{1 \le k \le m} 1/k \approx m \cdot \ln(m)$.*

Next, we show that for TE-Instance 3, two waypoints are sufficient to obtain a joint waypoint and weight setting that admits an ES-flow with $MLU = 1$.

Lemma 2.3.11. *In TE-Instance 3 (Figure 2.2b), using only two waypoints per demand, JOINT achieves a utilization JOINT $= 1$.*

Lemma 2.3.12. *TE-Instance 3 admits a gap $R_{LWO} \in \Omega(n \log n)$ when JOINT is constrained with $W \ge 2$ waypoints per demand.*

The proof of lemmas 2.3.11 and 2.3.12 can be found in Appendix 2.9.

We now present another instance similar to TE-Instance 3 that differs only in link capacities. We will use it to lower bound the gap R_{WPO} (Definition 2.3.1).

TE-Instance 4 (Figure 2.2c). *Consider the network in Figure 2.2c with n nodes and the parameter $m = n/2$. Each link (v_i, w_j), $1 \leq i, j \leq m$, has the capacity $1/(m - i + 1)$. All links $(v_i, v_{i+1}), (w_i, w_{i+1})$ have the same capacity $D \geq m \ln m$, where D is the total demand size. The demand list identical to that of Instance 3.*

Similarly to Lemma 2.3.11, we show that for TE-Instance 4, two waypoints are sufficient to obtain a joint waypoint and weight setting that admits an ES-flow with $MLU = 1$.

Lemma 2.3.13. *In TE-Instance 4 (Figure 2.2c), using two waypoints (per demand), JOINT achieves the utilization JOINT $= 1$.*

The proof of Lemma 2.3.13 can be found in Appendix 2.9.

In the following lemma, we lower-bound the gap R_{WPO} when the input weight setting is arbitrary or the standard weight setting from Definition 2.3.2.

Lemma 2.3.14. *Let $\mathcal{I}_3$ denote TE-Instance 3 (Figure 2.2b), and $\mathcal{I}_4$ denote TE-Instance 4 (Figure 2.2c). Assume the number of waypoints per demand for JOINT is constrained to $W \geq 2$, and for WPO, it is constrained to $W = \lceil c \cdot n \rceil$, $0 < c < 1/3$.*

(i) If the given weight setting is arbitrary, uniform, or the inverse of link capacities, then the instance $\mathcal{I}_4$ admits a gap

$$R_{WPO}(\mathcal{I}_4) \in \Omega((n \log n)/W).$$

(ii) If the given weight setting is the optimal from LWO then $\mathcal{I}_3$ admits a gap $R_{WPO}(\mathcal{I}_3) \in \Omega((n \log n)/W)$.

The proof of Lemma 2.3.14 can be found in Appendix 2.9.

TE-Instance 5. *Let $\mathcal{N}_3 = (G_3, c_3)$ denote the network instance 3 and (s_3, t_3) denote its source-target pair. Let $\mathcal{N}_4 = (G_4, c_4)$ denote instance 4 and (s_4, t_4) denote its source-target pair. We define the TE instance $(\mathcal{N}_5, \mathcal{D})$ where*

$$\mathcal{N}_5 = (G_3 \cup G_4 \cup \{(t_3, s_4)\}, c_3 \cup c_4 \cup \{(t_3, s_4) \mapsto D\})$$

is the concatenation of the two networks $\mathcal{N}_3$ and $\mathcal{N}_4$ with $n = 4m + 2$ nodes. The source node in $\mathcal{N}_5$ is s_3 and the target node is t_4. The link (t_3, s_4) with capacity D connects the target of $\mathcal{N}_3$ to the source of $\mathcal{N}_4$, where D is the total demand size. The demand list is identical to that of instances 3 and 4. That is, $\mathcal{D}$ consists of m^2 demands with sizes that can be partitioned into m identical harmonic sets H_m.

We aggregate all our optimality gaps into the (joint) TE gap R^* from Definition 2.3.3.

Theorem 2.3.15. *TE-Instance 5 admits a gap $R^* \in \Omega(n \log n/W)$ if*

1. *WPO is constrained to $W \leq c \cdot n$ waypoints for $c \leq 1/3$,*
2. *Joint is constrained to 2 waypoints, and*
3. *the given weight setting input to WPO is arbitrary, or it is the standard setting from Definition 2.3.2.*

The proof is in the Appendix 2.9. We unify the lower bounds from Theorems 2.3.4 and 2.3.15 (restricting $W \leq 2$) as follows:

Corollary 2.3.16. *When WPO is constrained to a constant number of waypoints (per demand) at least W, and Joint is constrained to at most W waypoints, the TE gap (Definition 2.3.3) satisfies $R^* \in \Omega(n)$ for $W = 1$, and $R^* \in \Omega(n \log n)$ for $W = 2$.*

2.4 Upper Bounding the Gap

In this section, we compare optimal values in Joint and LWO by studying the worst case of the ratio LWO/Joint and its upper bounds. This ratio is at least 1 since any feasible solution to LWO is feasible also in Joint for $W = 0$ (no waypoints). We consider only single source-target demands in this section.

First, we restate a helper lemma from [43]. Given a directed acyclic graph (DAG) G, it ensures a weight setting on G such that the ECMP-flow under this weight setting assigns positive flows to all links of G, i.e., every link is on a shortest path to the target node.

Lemma 2.4.1 (Lemma 2.5 in [43])**.** *Assume a given DAG G containing nodes s and t without any incoming link to s and outgoing link from t. There exists a weight setting for G such that the DAG of the induced ECMP-flow is identical to G.*

2.4.1 Uniform Capacities

We show that with uniform link capacities, applying LWO is sufficient to obtain the optimal MLU (Theorem 2.4.2). This immediately concludes the TE gap $R^* =$ LWO/Joint = 1.

Theorem 2.4.2. *Given a network and a list of demands $\mathcal{D}$ between the same source and target nodes, if the capacity of all links are equal (i.e. uniform) then LWO = OPT.*

Proof. Assume all links have the capacity C. We denote the capacity of a minimum cut between s and t by $cut(s, t)$. Thus, OPT $\geq \mathcal{D}/cut(s, t)$. By Menger's theorem [93], there exists a set of link-disjoint paths (corresponding to an acyclic maximum flow) denoted by $\mathcal{P}$ from s to t such that $C \cdot |\mathcal{P}| = cut(s, t)$. Therefore, OPT $\geq D/(C \cdot |\mathcal{P}|)$. We refer to paths in $\mathcal{P}$ as *basic* paths.

We construct a ES-flow that is feasible for LWO, then we show that the MLU under this flow is optimal, that is, LWO = OPT. Consider the DAG $G := \cup_{P \in \mathcal{P}} P$

obtained by taking the union of all basic paths in $\mathcal{P}$. By applying Lemma 2.4.1 to G, we obtain link weights such that every path in $\mathcal{P}$ is a shortest path to t. We set the weight of every other link $\mathcal{N} \setminus G$ to a number sufficiently large in order to ensure only links of basic paths are on a shortest path to t. Let $\mathcal{F}$ be the ES-(s,t)-flow in G (splitting evenly over links of basic paths).

Next, we show that $\mathcal{F}$ assigns the same flow size $D/|\mathcal{P}|$ to every link that belongs to any shortest path to t. The aggregate ES-flow at s splits evenly over all the outgoing links of s that are on a basic path (i.e. links in G). Hence, the flow on each of these outgoing links is of size $\mathcal{D}/|\mathcal{P}|$. We show by induction that the flow on every other link is $D/|\mathcal{P}|$ as well. Consider the topologically sorted ordering of nodes $v_1 = s, \dots, v_n = t$ in G. Assume the load on each outgoing links of every node $v_j, j < i < n$ is $\mathcal{D}/|\mathcal{P}|$. That is, the load on each link incoming to v_i is $D/|\mathcal{P}|$. We show the same flow size is assigned the outgoing links of v_i. The number of incoming links equals the number of outgoing links at v, as otherwise some basic paths share the same (incoming/outgoing) link of v_i, which contradicts the assumption they are link-disjoint. Therefore, each outgoing link carries a flow of size $D/|\mathcal{P}|$ concluding our claim.

Thus, every link that belongs to a basic path (i.e. to G) carries a portion of the aggregate flow of the size $D/|\mathcal{P}|$, which implies the maximum utilization in $\mathcal{F}$ is $(D/|\mathcal{P}|)/C = D/(C \cdot |\mathcal{P}|) = \text{OPT}$. □

Consider any maximum (s,t)-flow f^* and its decomposition into set of paths $\mathcal{P}$ s.t. $\sum_{P \in \mathcal{P}} c(p) = |f^*|$ (by *flow decomposition* theorem in [3]). We provide upper bounds for the TE gap in the number of paths in a flow decomposition and also in the number of links.

Theorem 2.4.3. *Given the flow decomposition $\mathcal{P}$ of any maximum (s,t)-flow, the optimality gap for an optimal weight setting satisfies $R_{LWO} \le |\mathcal{P}| \le |E|$.*

Proof. Let $p_{max} \in \mathcal{P}$ be the path with the largest capacity in the flow decomposition $\mathcal{P}$ and let $c(p_{max})$ be its capacity, i.e., the capacity of the weakest link on p_{max}. Set the weight of every link in p_{max} to 1 and the weight of every other link to n. Therefore, p_{max} is the unique shortest (s,t)-path in $\mathcal{N}$. The MLU under this weight setting is $D/c(p_{max})$ and

$$\begin{aligned}\text{LWO} &\le \frac{D}{c(p_{max})} = \frac{|\mathcal{P}| \cdot D}{|\mathcal{P}| \cdot c(p_{max})} \le \frac{|\mathcal{P}| \cdot c(p_{max})}{\sum_{P \in \mathcal{P}} c(p)} \\ &= \frac{|\mathcal{P}| \cdot D}{|f^*|} \le |\mathcal{P}| \cdot \text{OPT}(\mathcal{P}).\end{aligned}$$

It is known from the flow decomposition theorem that $|\mathcal{P}| \le |E|$. Thus, we conclude

$$R_{\text{LWO}} = \frac{\text{LWO}}{\text{Joint}} \le \frac{\text{LWO}}{\text{OPT}} \le |\mathcal{P}| \le |E|. \quad \square$$

Note that Theorem 2.4.3 refines Theorem 2.3.15 for sparse networks, as it eliminates the logarithmic factor when $|E| \in O(n)$.

2.4.2 General Networks

For networks on arbitrary graphs, arbitrary capacities, and single source-target demands, we show that the gap ratio is bounded above by a factor in $O(n \log n)$ (Corollary 2.4.4). This immediately implies our lower bound in Theorem 2.3.15 is asymptotically tight.

In Section 2.5, we introduce the algorithm LWO-APX that approximates the optimal weight setting within a factor in $O(n \log n)$ of the optimal setting (Theorem 2.5.4). This bound implies

$$R_{\text{LWO}} = \frac{\text{LWO}}{\textsc{Joint}} \leq \frac{\text{LWO}}{\text{OPT}} \leq \frac{\text{LWO-APX}}{\text{OPT}} \in O(n \log n).$$

Hence, the following corollary follows immediately.

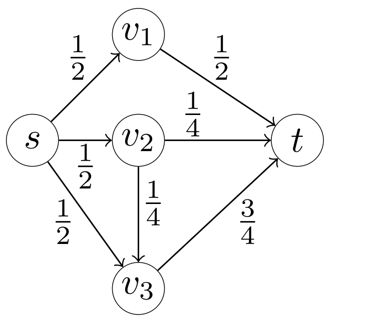
Corollary 2.4.4. *If all demands share the same source-target pair and* $\textsc{Joint}$ *can assign an arbitrary number of waypoints to any demand (i.e.,* $W = n$*), then* $R^* \in O(n \log n)$.

2.5 Approximating Optimal Weights

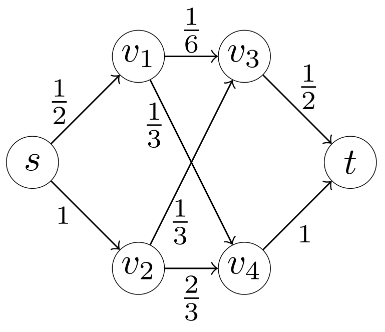
We introduce an approximation algorithm for LWO (Algorithm 2.1) that computes a weight setting for instances where all demands are defined between the same source-target pair (s, t). We fix an acyclic maximum (s, t)-flow denoted by f^*. The capacity of a link may be arbitrarily larger than $|f^*|$ (the size of the flow). To obtain minimal capacities without decreasing the flow size, we redefine the capacity of each link ℓ as $c^*(\ell) = f^*(\ell)$, i.e., the fraction of the flow through ℓ, and we refer to it as *usable capacity.* Next, we refine the notion of usable capacity and define a notion of capacity for nodes and links that is minimal for maximal ES-(s, t)-flows.

Effective Capacity. Informally, the *effective capacity (e-capacity)* of a node v with respect to the target node t, denoted by $ec_t(v)$, is the size of the maximal ES-flow from v to t subject to c^*. Since the flow splits evenly at v, an outgoing link of v having the smallest e-capacity determines the size of the maximal ES-flow feasible from v. The e-capacity of a node v is equal to the smallest e-capacity of links in $out_G(v) = \{\ell \mid \ell = (v, *)\}$ times the *out-degree* of v denoted by $\delta_G(v) = |out_G(v)|$. The e-capacity of a link $\ell = (*, u)$ is bounded by the e-capacity of u and $c^*(\ell)$, i.e., the size of the maximum (s, t)-flow through ℓ subject to the original capacities c. We define the notation ec_t formally as follows.

Definition 2.5.1. *Assume a network* $N = (G, c)$*, source-target pair* (s, t)*, and an acyclic maximum* (s, t)*-flow* f^* *are given. Let* G^* *be the acyclic subgraph of* G *consisting of links with a positive flow under* f^**. Let* $c^*(\ell) = f^*(\ell)$ *be the usable capacity for each link* ℓ*. We define the effective capacity of all nodes and links w.r.t* t *inductively. We define* $ec_t(t) = \infty$*. Assume the effective capacity of all outgoing links of a node* $v \neq t$ *is already determined. Then*



(a)
$ec(t) = \infty$
$ec(v_1) = \frac{1}{2}$
$ec(v_2) = 2 \times \frac{1}{4}$
$ec(v_3) = \frac{3}{4}$
$ec((s, v_1)) = ec(v_1) = \frac{1}{2}$
$ec((s, v_2)) = ec(v_2) = \frac{1}{2}$
$ec((s, v_3)) = ec(v_3) = \frac{3}{4}$
$ec(s) = 3 \times ec((s, v_1)) = \frac{3}{2}$
$|f^*| = \frac{3}{2} = ec(s)$



(b)
$ec(t) = \infty$
$ec(v_3) = \frac{1}{2}$
$ec(v_4) = 1$
$ec(v_1) = 2 \times \frac{1}{6} = \frac{1}{3}$
$ec(v_2) = 2 \times \frac{1}{3} = \frac{2}{3}$
$ec((s, v_1)) = ec(v_1) = \frac{1}{3}$
$ec((s, v_2)) = ec(v_2) = \frac{2}{3}$
$ec(s) = 2 \times ec((s, v_1)) = \frac{2}{3}$
$|f^*| = \frac{3}{2} = 2.25 \times ec_t(s)$

Figure 2.3: Two examples illustrating the effective capacity of nodes and links from Definition 2.5.1. The label of each link represents its capacity which for simplicity is chosen to be also its usable capacity, i.e., the value assigned by a maximum (s, t)-flow f^*. In (2.3a), the effective capacity of the node s equals its usable capacity. In (2.3b), the effective capacity of s is less than half of its usable capacity.

- $ec_t(v) = \delta_{G^*}(v) \times \min_{\ell'=(v,*)} ec_t(\ell')$,
- $\forall \ell = (*, v) \, : \, ec_t(\ell) = \min\{c^*(\ell), ec_t(v)\}$.

We use $ec(.)$ in place of $ec_t(.)$ when t is clear from the context. Definition 2.5.1 is also an algorithm that computes the size of a ES-flow from every node to the target node t. This algorithm is embedded in Algorithm 2.1, which computes an approximately maximum ES-flow. Figure 2.3 illustrates the distinction between the usable capacity of nodes and links, i.e., the size of the maximum (s, t)-flow feasible through them, and their e-capacity. In both examples, the e-capacity of each link connected to t equals its usable capacity. The e-capacity of the remaining links and nodes are obtained in a backward traversal starting from t. Observe that in Figure 2.3b, there are two choices at v_1: splitting the flow evenly over its two outgoing links yields the same e-capacity as not splitting and using only the link with larger e-capacity. Therefore in such cases, we break ties arbitrarily, e.g., by always splitting. In contrast, if we split the flow evenly at v_2 then the size of the maximum ES-flow is limited to $2 \times 1/4 = 1/2$, while not splitting allows for a larger ES-flow of size $2/3$ from v_2.

Algorithm 2.1: Algorithm LWO-APX

```
Input  : directed network $\mathcal{N}$, nodes $s$ and $t$
Output: weight setting $E \mapsto \mathbb{R}^+$
// initialization
1  Let $G^*$ denote the subgraph of $G$ consisting of links $\ell$ with positive
     flow $c^*(\ell) = f^*(\ell) > 0$ in an acyclic maximum $(s,t)$-flow $f^*$.
2  Let $v_1, \dots, v_n$, where $v_1 = t, v_n = s$, be nodes of $G^*$ sorted in the reverse topological
     ordering, and let $\delta_i = \delta_{G^*}(v_i)$ denote the number of $v_i$'s outgoing links in $G^*$.
3  Initialize $ec : V \mapsto \mathbb{R}^+$ for assigning effective capacities.
// maximizing the e-capacity of $s$
4  Set $ec(t) = \infty$                                  // the e-capacity of $t$
5  for $i = 2$ to $i = n$ do                              // for each node $v_i \neq t$
6      Let $\ell_1, \dots, \ell_{\delta_i}; \ell_j \in out(v_i)$ be the outgoing links of $v_i$ sorted
         s.t. $ec(\ell_1) \geq \cdots \geq ec(\ell_{\delta_i})$.
7      Let $j^* = \arg\max_{1 \leq j \leq \delta_i}\{j \cdot ec(\ell_j)\}$.
8      $ec(v_i) = j^* \cdot ec(\ell_{j^*})$                       // e-capacity of $v_i$
9      $\forall \ell \in in(v_i) : ec(\ell) = \min\{c^*(\ell), ec(v_i)\}$
       //removing links of low e-capacity
10     $G^* = G^* \setminus \{\ell_{j^*+1}, \dots, \ell_{\delta_i}\}$
11 return the weight setting for $G^*$ from Lemma 2.4.1.
```

Algorithm 2.1 selects a subset of outgoing links at every node such that splitting the flow evenly over these links yields an approximately optimal ES-flow.

Now we describe Algorithm 2.1 (LWO-APX), which employs Definition 2.5.1 for computing an approximately optimal weight setting. The algorithm works in two stages. The first stage (Lines 5–10) computes an ES-flow with a size at least $\approx |f^*|/n \log n$. It begins with computing an acyclic maximum (s,t)-flow and its corresponding DAG G^*. Then each iteration i removes certain outgoing links of v_i from G^* to optimize the ES-flow from v_i. Specifically, for each node v_i in the reverse topological ordering of nodes of G^*, the algorithm at Line 7 selects a subset of outgoing neighbors such that the effective capacity of v_i is (locally) maximized. Then it removes the remaining outgoing links of v_i from G^*. Note that at each iteration, the effective capacity of all outgoing links has been determined in previous iterations. After all these iterations, G^* reduces to a possibly sparser DAG where the size of the ES-flow on this DAG is within our approximation guarantee. The second stage (Line 11) produces a weight setting that realizes the ES-flow computed in the first stage.

Analysis

We first show that at the end of the ith iteration, the e-capacity of v_i may be smaller than the sum of children's e-capacity by at most a logarithmic factor. Equivalently, the ith iteration reduces the size of the maximal ES-flow from v_i to t by a logarithmic factor of its out-degree $\delta_{G^*}(v_i)$. Intuitively, the algorithm takes the optimal flow (with optimal flow-splits) and in $|V|$ iterations transforms it into a flow that splits evenly everywhere. Each iteration yields an intermediate flow with one more even-splitting node than the previous flow; this is the reason behind the capacity reduction. The worst-case of this reduction occurs when e-capacities of the outgoing links of v_i constitute a harmonic series. The following lemma and its proof formalize the idea.

Lemma 2.5.2. *Consider the DAG G^* of the maximum (s,t)-flow (Line 2.1.1). The total effective capacity of v_i's outgoing links in G^* is at most a logarithmic factor larger than v_i's effective capacity. That is, $\sum_{\ell=(v_i,*)} ec(\ell) \leq \lceil \ln(\delta_{G^*}(v_i)) \rceil \cdot ec(v_i)$.*

Proof. Consider the outgoing links of v_i in the non-decreasing order of their effective capacities at line 2.1.6. The algorithm at line 2.1.7 selects the first j^* values in this ordering s.t. $ec(v_i) = j^* \cdot ec(\ell_{j^*})$ is maximized. Thus, for each outgoing link at ℓ_j, we have $j \cdot ec(\ell_j) \leq j^* \cdot ec(\ell_{j^*})$ and

$$ec(\ell_j) \leq (1/j) \cdot j^* \cdot ec(\ell_{j^*}) = (1/j) \cdot ec(v_i). \tag{2}$$

The total effective capacity of all outgoing links of v_i is

$$\sum_{1 \leq j \leq \delta_{G^*}(v_i)} ec(\ell_j) \leq ec(v_i) \cdot \sum_{1 \leq j \leq \delta_{G^*}(v_i)} (1/j) \leq ec(v_i) \cdot \lceil \ln(\delta_{G^*}(v_i)) \rceil,$$

which follows from (2) and the fact $\sum_{1 \leq y \leq z} \approx \ln(z)$. □

The effective capacity of the source node determines the MLU under the weight setting computed by Algorithm 2.1. We show that the effective capacity of s may be smaller than its usable capacity, i.e. the size of the maximum (s,t)-flow, by at most the factor $n \log n$. That is, we show $c(s) \leq n \log n \cdot ec(s)$, which implies the approximation factor $n \log n$. Recall that $ec(u)$, the effective capacity at the node u, is the size of the maximum ES-flow from u to t. Next, we show that $ec(s)$ may be smaller than the usable capacity from s to t at most by the factor that depends on the number of nodes n.

Lemma 2.5.3. *Let Δ^* denote the largest splitting factor of the maximum (s,t)-flow f^* in $\mathcal{N}$. Then,*

$$|f^*| \leq n \cdot \lceil \ln(\Delta^*) \rceil \cdot ec_t(s).$$

Proof Idea. Recall that each iteration (Line 5) determines the e-capacity of a node v_i, which by Lemma 2.5.2 may be smaller than the ec of v_i's children by a logarithmic factor of their number. We interpret each iteration (Line 5) as a process that reduces the usable capacity at s. Consider the reverse of this process where we undo the ith iteration by switching from even-split to optimal split at v_i. Before the reverse process, v_i splits its flow evenly its usable capacity is $ec(v_i)$. Afterward, v_i splits optimally and its usable capacity is $\sum_{u \in out(v_i)} ec(u) \geq ec(v_i)$. We upper bound the impact of each reversed iteration on s (for each v_i) in a formal proof in Appendix 2.9. We conclude the approximation factor of LWO-APX as follows.

Algorithm 2.2: JOINT-HEUR

```
Input  : network $\mathcal{N}$, demand list $\mathcal{D}$
Output: weight and waypoint settings
1 Run HeurOSPF [83] and let $\omega$ be the weight setting result.
2 Run GreedyWPO using $\omega$ to obtain a waypoint setting $\pi$.
3 Replace each demand $\psi = (s, t, d) \in \mathcal{D}$ with two new demands: $(s, \pi_\psi, d)$ and
  $(\pi_\psi, t, d)$, and let $\mathcal{D}'$ be the new demand list.
4 Run HeurOSPF on $\mathcal{D}'$ to obtain a new weight setting $\omega'$.
5 return $\omega'$ and $\pi$.
```

Theorem 2.5.4. *The weight setting from Algorithm 2.1 induces a maximum ECMP-flow smaller than the maximum (s, t)-flow (i.e. OPT) by at most a factor in $O(n \log n)$.*

Proof. Assume a given TE-instance $\mathcal{I} = (\mathcal{N}, \mathcal{D}, w^*)$ comprising a network $\mathcal{N}$, a single source-target demand list $\mathcal{D}$, and a weight setting w^* from Algorithm 2.1.

Let Δ^* be the largest out-degree in $\mathcal{N}$. Let LWO-APX($\mathcal{I}$) denote the MLU under the ECMP-flow induced by w^*. Recall that $\mathrm{OPT}(\mathcal{I}) = D/|f^*|$ is the MLU under the optimal (arbitrary split) flow, where $|f^*|$ is the size of the maximum (s, t)-flow.

By the definition of effective capacity, the MLU under w^* is LWO-APX($\mathcal{I}$) $= D/ec(s)$. Then from Lemma 2.5.3 and the fact that $\Delta^* < n$, we obtain LWO-APX($\mathcal{I}$) $= D/ec(s) \leq$

$$\frac{D}{|f^*|/(n\lceil \ln(\Delta^*)\rceil)} = \frac{D \cdot n\lceil \ln(n)\rceil}{|f^*|} = n \cdot \lceil \ln(n)\rceil \cdot \mathrm{OPT}(\mathcal{I}),$$

which concludes the claim. □

2.6 Algorithm for Joint

As the Joint problem is NP-hard, we propose a polynomial-time algorithm that employs an adaptation of the local search algorithm of Fortz and Thorup [83] as a subroutine (referred as HeurOSPF). Given a general demand list (arbitrary source-target pairs) and a general network instance $\mathcal{I} = (\mathcal{N}, \mathcal{D})$, Algorithm 2.2 computes a weight setting and a waypoint setting by running the two optimizations separately and iteratively. While it is not exactly a joint optimization, it is a first attempt to approximate Joint. Our experiments show that even such a simplistic heuristic can improve the utilization beyond what is feasible with weight optimization.

2.7 Empirical Gap Observations

In §2.3, we provided bounds on worst cases of the gap (ratio) between Joint and LWO. In this section, we observe the gap on a collection of real network topologies with real and synthetic traffic demands [69].

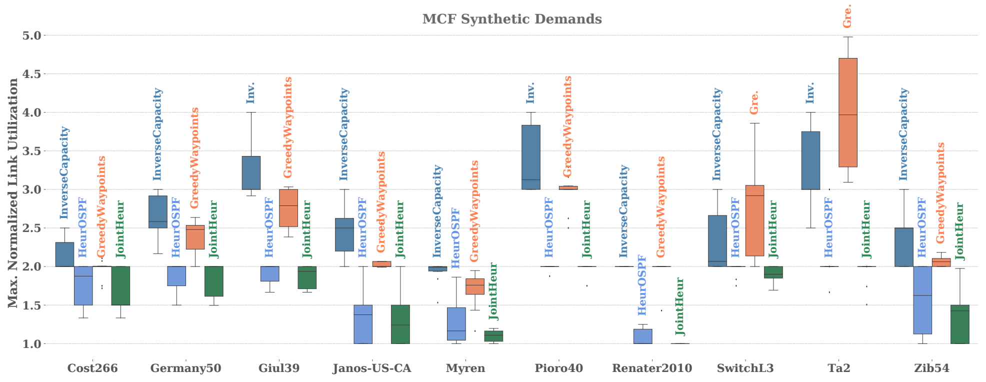


Figure 2.4: MLU statistics from different algorithms on the 10 largest capacitated non-tree topologies from TopologyZoo and SNDLib. Demands are generated using the MCF Synthetic method.

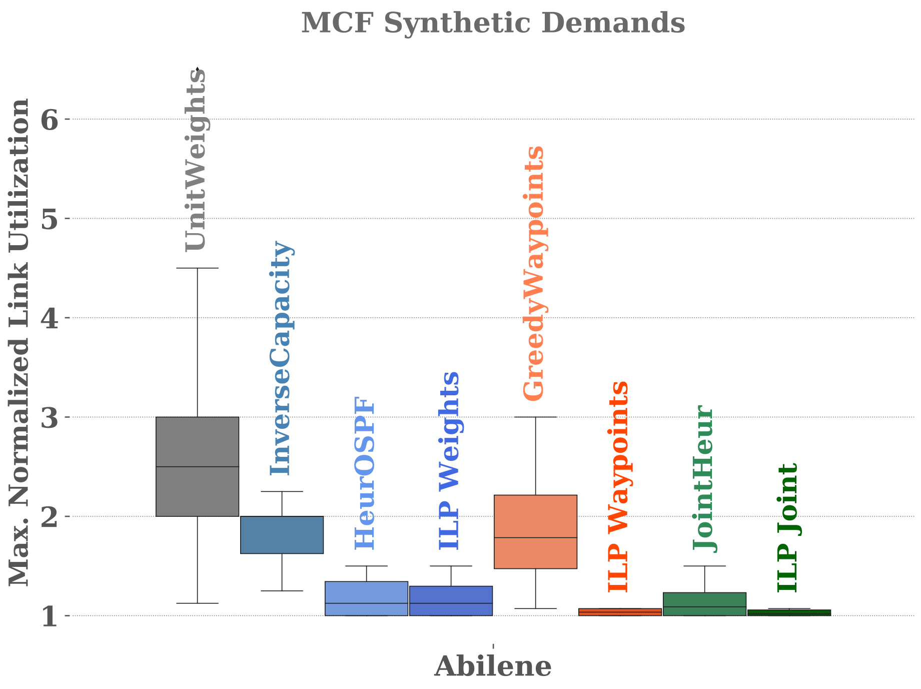


Figure 2.5: Comparing MILP Results to Heuristics.

Algorithm 2.3: Waypoint Selection (GreedyWPO)

```
input : network instance $(\mathcal{N}, \mathcal{D})$, weight setting $\omega$
output: waypoint setting $\pi : \mathcal{D} \mapsto V$
1 Let $U_{min}$ be the MLU of the ECMP-flow for $\mathcal{D}$ induced by $\omega$.
2 for each demand $\psi = (s,t,d) \in \mathcal{D}$ in the descending order of demand sizes do
3     for each node $w \in V$ do                    // improving MLU greedily
4         Let $\mathcal{D}' := \mathcal{D} \setminus \{\psi\} \cup \{(s,w,d),(w,t,d)\}$.
5         Let $U$ be the MLU under $\mathcal{D}'$ and induced by $\omega$.
6         If $U < U_{\min}$ then set $\pi_\psi = w$ and $U_{\min} = U$.
7 return $\pi$.
```

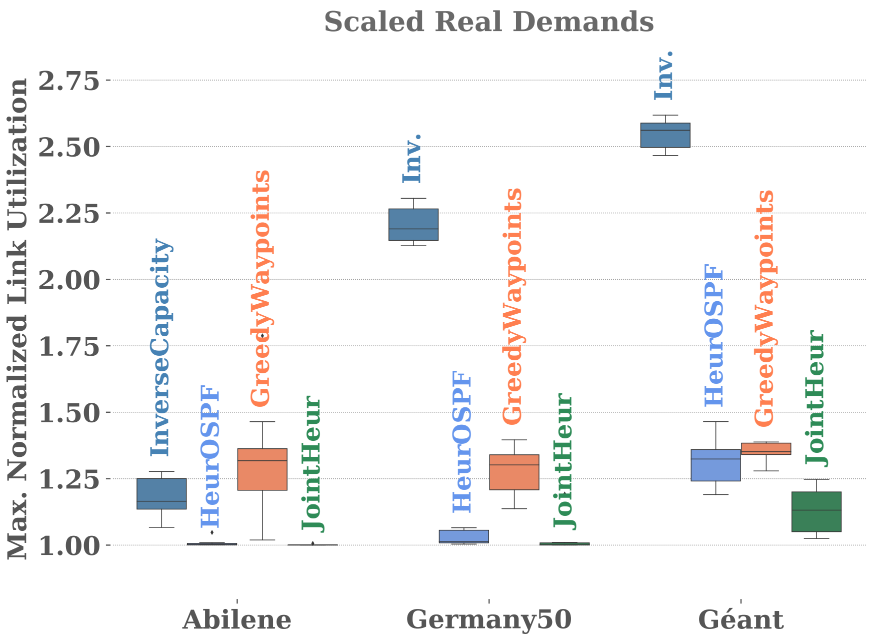


Figure 2.6: MLU under real demands and SNDLib topologies.

Test Environment. All simulations were executed on an HP DL380 G9 with 2x Intel Xeons E5-2697V3 SR1XF with 2.6 GHz, 14 cores each, and a total of 128 GB DDR4 RAM. The host machine was running Ubuntu 18.04.4 LTS. We implemented the proposed algorithms in Python (3.7.10) [155] leveraging the libraries NetworkX (2.5.1) [147], NetworKit (8.1) [146] Numpy (1.20.3) [149], and SciPy (1.6.3) [167]. To solve the MIPs/LPs we used Gurobi (9.1.2) [96].

Data Sources. For our simulations, we used real-world traffic data and topology from SNDLib [**SNDLib10**, **orlowski2010SNDLib**, **SNDLib**], which is provided in a standardized XML file format. SNDLib provides a large set of data for the Abilene, Géant and Germany50 topologies consisting of traffic matrices in various granularity levels. Our second real-world source is TopologyZoo [123, 180] which provides only topology data in GraphML format. We selected topologies with capacity information and used the latest version of a topology if multiple ones exist. We synthesize traffic demands using the maximal concurrent multi-commodity flow (MCF) [69] as follows.

Demand generation. Due to the proprietary nature of ISPs and backbones we could not procure real traffic data. We extracted the traffic data from research repositories and synthesized traffic for the evaluation of arbitrary topologies.

MCF Synthetic Demands. In all our experiments with non-real demands, we generated demands using the maximal multi-commodity flow (MCF) formulated in [69]. We randomly select 20% of connection pairs and scale the demand between those as such that the objective MLU computed by the MCF routing is 1. The resulting demands are further split into smaller sub-demands so that each source-target pair has multiple equal-size flows. The number of flows per pair scales with the topology size, or precisely, by $|E|/4$.

2.7.1 Comparing the different Algorithms

We now present our empirical results on the previously described real-world and synthetic data. Each algorithm is evaluated with 10 sets of demands on each of the selected topologies.

Small Networks. The MILP formulation of JOINT is presented in [69]. Due to its complexity, we computed the optimal solutions using the MILP solver only on Abilene with 12 nodes and 30 links. We derive formulations for LWO and LWO from the JOINT's MILP separately: for LWO, we simply set $W = 0$; for WPO, given a weight setting ω', we add one constraint for each link ℓ: $\omega_\ell = \omega'(\ell)$. We obtained optimal solutions from each MILP on small examples and results are depicted in Figure 2.5. The average MLU for WPO, LWO, and JOINT respectively is 1.17, 1.04, and 1.03.

Large Networks In Figure 2.4, we evaluated the heuristic algorithms on all topologies from TopologyZoo and SNDLib, which provide complete link capacity information. The average MLU of the naive algorithm InverseCapacity is 2.74 which improves to 1.65 using HEUROSPF (from [83]) and further to 1.58 using our JOINT-HEUR. The running time of our JOINT-HEUR is longer than that of HEUROSPF by up up 3min in the largest topology with $|V| = 65$ nodes. The improvements from applying a second weight optimization (steps 3 and 4) is negligible. The first two stages of JOINT-HEUR (steps 1 and 2) are already sufficient for most of evaluated instances, and the plots are obtained only from the first two steps.

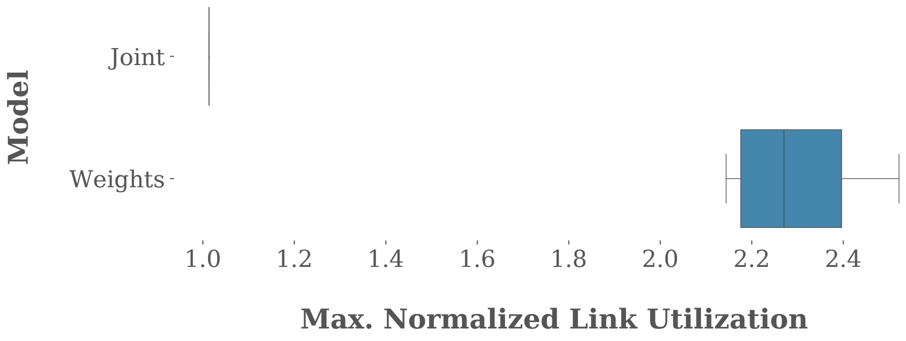


Figure 2.7: Nanonet Experiments: Joint vs. LWO

Real Demands. In Figure 2.6, we evaluated the gap reduction on real demands from SNDLib. Note that for real traffic demands, all connection pairs are active, though a huge skew can be observed. The average MLU of 1.11 using the best LWO heuristic, i.e. HeurOSPF, improves to 1.05 using our joint optimization.

Discussion. In all of the experiments, the MCF is $MLU = 1$ since we scale all demand sizes. Figure 2.5 shows that the Joint MILP achieves an average MLU of 1.03, but its complexity prevents employing it on larger network instances. Our JOINT-Heur improves the average MLU on nearly all evaluated topologies compared to the LWO heuristic HeurOSPF. Particularly, the instances using real demands (Figure 2.6) indicate the potential of optimization utilizing waypoints on top of conventional LWO techniques. Here we obtained an average MLU of 1.05. The overhead execution time due to waypoint setting is negligible compared to the execution time of HeurOSPF.

2.7.2 Nanonet Implementation

To further verify our model, we performed a small investigation in Nanonet [128], a virtualized network environment conceptually based off Mininet. Nanonet simulates network nodes by creating network namespaces in the Linux kernel and (virtual) links between them. Shortest path routes are calculated directly by Nanonet, which also supports ECMP. To achieve better splitting, Layer-4 hash was used by setting `net.ipv6.fib_multipath_hash_policy=1` on all (virtual) nodes. In our experiments, we use the TE-Instance 1 with optimal solutions for LWO and Joint and measure the link utilization to obtain the MLU. For the Joint experiment, additional routes, possibly including a segment, are added to the sources. As perfect splitting is not the case due to the hash functions even with an L4 hash, we add four additional (pseudo) nodes, each for one flow. The flows all start at the same time and run for 300 seconds. For the throughput evaluation we use `nuttcp` 6.1.2, with 32 streams per source and limit the bandwidth to the total demand size. Thus, for

Weights, the rate is set to 40 MBit/s with 32 parallel streams, and in case of Joint the rate is set to 4 times 10 MBit/s with 32 parallel streams each. The results of 10 executions of each test are depicted in Figure 2.7. The results at Joint are relatively constant and match the expected normalized value of 1 closely, with some deviation due to, e.g., Neighbor Discovery Protocol packets. However, Weight has a wider range beyond the expected MLU of 2, caused by the hash function, where the flows are not split perfectly across the equal cost routes. More precisely, Joint in our 10 test executions has MLU results of approximately ≈ 1.0138, while Weight has a range from ≈ 2.1439 to ≈ 2.5219, with a median of ≈ 2.2704. For the latter, parallel equal cost routes have a roughly correspondingly reduced link utilization. We hence conclude that the expected results are closely matched in Nanonet, with some deviation caused by the L4 hash function.

2.8 Conclusion

Motivated by the traffic engineering flexibilities introduced by segment routing architectures, we studied analytically and empirically the benefit of jointly optimizing link weights and waypoints (mathematically modeled as an optimality gap) to improve network utilization. We showed that already in relatively simple settings, the joint optimization can help significantly compared to individual optimizations: the achievable maximum link utilization can be improved by a factor linear in the number of nodes n, respectively even $\Omega(n \log n)$ for dense networks. We gave an $O(n \log n)$ approximation algorithm for link-weight optimization in a single source-target setting, which also serves as an upper bound for our optimality gap, rendering the gap asymptotically tight.

To evaluate the gap empirically, we presented both a mixed-integer linear program formulation and an efficient heuristic for joint waypoint and weight optimization. Our heuristic is scalable and provides fair utilization benefits over prior work, evaluated on various real-world topologies. It would be interesting to see an analysis for heuristics such as ours that combine the two optimization strategies sequentially. We leave open questions: how well a sequential approach can approximate the optimal Joint solution in practical instances? How may iterations and how many waypoints would be sufficient to achieve the best outcome within a reasonable runtime? We also validate our model setting in a proof-of-concept experiment in a Mininet-like environment.

Overall, we see our work as a first step towards unifying the two TE strategies, and we believe that it opens several interesting avenues for future work. We assumed static traffic demands and focused on the worst-case analysis of the loss if the two optimizations are not performed jointly. It would be interesting to explore TE algorithms that react to shifts in the traffic demand and account for reconfiguration costs.

2.9 Deferred Proofs

Proof of Lemma 2.3.11. We set the weight m to each pair of links (v_i, w_j) and (w_j, v_i), $1 \le i, j \le m$, and the weight 1 to every other link. We assign two waypoints to each demand $(s, t, 1/j) \in \mathcal{D}$ such that its flow is routed along a link (v_i, w_j) which has the matching capacity $1/j$. Specifically, for the ith demand of size $1/j$, we set v_i, w_j as waypoints, to be visited in the same order. Under these weights, the only shortest path to v_i is $s, v_2, \dots, v_i$ with a cost at most m, as any alternative path has a weight larger than $2m$. The shortest path between each pair of waypoints v_i and w_j is the link between them. Each node w_j receives one flow of size $1/j$ via each incoming link, which is then rerouted along the only shortest path to t of capacity D. Thus, all capacities are respected and the MLU under this weight and waypoint setting is 1. □

Proof of Lemma 2.3.12. We show the maximum ES-flow from s to t is 2. Let $\mathcal{N}$ denote the network in Figure 2.2b (Instance 2) and $\mathcal{N}'$ denote the network in Figure 2.2a. Consider the sub-network $\mathcal{N}_i \subset \mathcal{N}$ induced by the node set $\{v_i\} \cup \{w_j\}_j$ in $\mathcal{N}$. Recall that each link (v_i, w_j), $1 \le j \le m$, has the capacity $1/j \in H_m$, which is also the size of the maximum ES-flow feasible through the link. Observe that from the perspective of v_i and the source node s in $\mathcal{N}'$, networks $\mathcal{N}_i$ and $\mathcal{N}'$ are indistinguishable. Then, applying Lemma 2.3.10 to $\mathcal{N}_i$ implies the size of every maximum even-split (v_i, t)-flow f_i in $\mathcal{N}_i$ is $|f_i| = 1$. Then, the size of the maximum ES-flow from v_m to t is 1. However, this is not the case for $v_{i'}$, $i' < m$, because of the additional link $(v_{i'}, v_{i'+1})$.

We show that at maximum, two units of ES-flow can be delivered from each $v_{i'}$ including $v_1 = s$. From v_{m-1}, the sub-network $\mathcal{N}_{m-1}$ can deliver 1 unit of ES-flow to t. Another unit of ES-flow can be delivered via the link (v_{m-1}, v_m). Therefore, the size of the maximum even-split (v_{m-1}, t)-flow is 2, which is also the maximum feasible quantity via the link (v_{m-2}, v_{m-1}). At the node v_{m-2}, the sub-network $\mathcal{N}_{m-2}$ can deliver 1 unit in addition to 2 units via the link (v_{m-2}, v_{m-1}). Since the flow splits evenly, the size of the maximum (v_{m-2}, t)-ES-flow is 2. Repeating a similar argument for nodes $v_{m-3}, \dots, v_1$ implies the size 2 for the maximum (s, t)-ES-flow. Hence, satisfying all demands requires an ES-flow larger than the maximum (feasible) ES-flow by the factor $(m \ln m)/2$. Applying Lemma 2.3.11 yields $R_{\text{LWO}} = \text{LWO}/1 = (m \ln m)/2 \in \Omega(n \log n)$. □

Proof of Lemma 2.3.13. We set the weight m to each pair of links (v_i, w_j) and (w_j, v_i), $1 \le i, j \le m$, and the weight 1 to every other link. We assign two waypoints to each demand, such that its flow is routed along the link (v_i, w_j) that has a matching capacity. Specifically, for the jth demand of size $1/(m - i + 1)$, we set nodes v_i, w_j as waypoints to be visited in the same order. Under these weights, the only shortest path to v_i is $s, v_2, \dots, v_i$ with a cost at most m, as any alternative path has a weight larger than $2m$. The shortest path between each pair of waypoints v_i and w_j is the link between them. Each node w_j receives m flows of harmonic sizes, which is then routed along the unique shortest path to t with capacity D, i.e., $w_j, \dots, w_m$. Thus, no link is overloaded and the MLU is 1 under this weight and waypoint settings. □

Proof of Lemma 2.3.14. In each of the given weight setting cases, we show that WPO cannot utilize a large part of the available capacity (unless $W \geq m$), while Joint can utilize all of this capacity using only two waypoints and an appropriate weight setting. The basic idea in all cases is as follows. Limited to $W = c \cdot n$ waypoints for a constant c. e.g. $c = 1/3$, WPO distributes the load D (i.e. total demand size) over $\approx n/3$ of nodes v_i (Figure 2.2b and 2.2c). Hence, WPO manages to distribute the load over a number of parts $W = n/3$, each part having a size $\approx D/W$, implying the claim.

Next, we give our formal proof based off the aforementioned idea. We observe the optimal MLU in WPO on the instance $\mathcal{I}_4$ (Figure 2.2c) and under each assumed weight setting separately.

Arbitrary Weights. For any $\epsilon < 1/2$, set the weight ϵ for links (s, w_1), (w_1, v_i) and (v_i, w_i), $1 \leq i \leq m$, and the weight 1 for every other link. Under this weight setting, the shortest path from s to every other node starts with the link (s, w_1). Therefore, in any waypoint setting, the link (s, w_1) receives the entire load D, and WPO $= D/(1/m) = D \cdot m \geq m^2 \ln m$.

Uniform Weights. Assume w.l.o.g. the weight of every link is 1. In this setting, for any node $v_i \notin \{s, v_2\}$, all links (s, w_j) are on shortest paths from s to v_i. However, by inserting waypoints, flows can be routed away from these links. The shortest path from v_i to v_{i+1} is the link (v_i, v_{i+1}). Hence, WPO may route a flow to a node $v_k, k \leq W + 1$, via the path $s, v_2, \dots, v_k$, using $k - 1 \leq W$ waypoints: $v_2, \dots, v_k$. From v_k, WPO may split flows optimally at this node using one additional waypoint which is a node w_j. The capacity of each link emanating from v_k is $1/(m - k + 1)$. Therefore the maximum (arbitrary-split) flow feasible from v_k to t is at most $m/(m - k + 1) < m/(m - W)$. We refer to the latter as the *available capacity* at v_k.

Hence, using up to $W = c \cdot n < m$ waypoints in an optimal waypoint setting for $c \leq 1/3$, WPO may distribute the set of flows (of total size $D = m \log m$) over nodes $v_1, \dots, v_W$ proportionally to their available capacities in $1, \dots, m/(m - W + 1)$, before forwarding it to t via the upper (horizontal) path. Therefore, the size of the maximum (s, t)-ES-flow via at most W waypoints (per flow) is at most $W \cdot m/(m - W + 1)$, and

$$\text{WPO} \geq \frac{m \cdot \ln m}{W \cdot m/(m - W + 1)} = \frac{(m - W + 1) \cdot \ln m}{W} \in \Omega(\frac{n \log n}{W}),$$

since $(m - W) \geq (n/2 - n/3) \in \Omega(n)$.

Inverse of Capacities. In this case, the weight of each link equals the reciprocal of its capacity. That is, the weight of every link (v_i, v_{i+1}) is $1/D$, and every other links has the weight 1. Similarly to the construction in Section 2.3.2, and replace the two links (s, v_2) and (v_2, v_3) with D (parallel) paths each comprising two new links with the capacity 1. As a result, the cost of the shortest path from s to $v_i, i \geq 3$ is larger than 2, which makes the path through w_j (of cost 2) a shortest path to nodes $v_i, i \geq 3$, overloading links (s, w_j). It is not difficult to see that an argument analogously as in the case of unit weights follows from here and that $R_{\text{WPO}} \in \Omega(n \log n/W)$.

Optimal Weights. We show Lemma 2.3.14.(ii) for TE-Instance $\mathcal{I}_3$. By Lemma 2.3.12, a maximum of 2 ES-flow units is feasible from any $v_i, i < m$ to t. We set link weights realizing such optimal ES-flow. Let $\varepsilon = 1/(2(m+1))$. Consider the weight setting:

- the weight 2ε to the link (s, w_1),
- the weight ε to the link (v_2, w_1),
- the weight ε to links $(v_i, v_{i+1}), (w_i, w_{i+1}), (w_1, v_i), 1 \leq i \leq m$,
- and the weight 1 (or larger) to every other link.

Under this setting, both of the unit-capacity paths $s, w_1, w_2, \dots, t$ and $s, v_2, w_1, w_2, \dots, t$ are shortest paths from s to t with the equal cost $(m+1) \cdot \varepsilon = 1/2$. Via these two paths, 2 units of flow can split equally at s and arrive at t without overloading any link, implying the weight setting is optimal for LWO (Lemma 2.3.12).

Observe that for any node $u \notin \{s, v_2, v_3\}$, the link (s, w_1) is always on a shortest path from s to u, and at least half of the flow from s to u traverses through this link. In general, for any node $u \notin \{v_i, v_{i+1}, v_{i+2}\}$, the link (v_i, w_1) is always on a shortest path from v_i to u, and at least half of a flow from v_i to u traverses through this link (the half flow is the case only for the flow to v_{i+3}).

Therefore, by setting nodes v_2 or v_3 as the first waypoint, WPO can distribute the entire load D equally on nodes $\{s, v_2, v_3\}$. Each additional waypoint increases extend of the reach two more v_i nodes. In general, using $k \leq W$ waypoints $v_2, v_4, \dots, v_{2k}$ for a demand, its flow reaches the node v_{2k}, and then takes the link (v_{2k}, w_1) towards t. Hence, we can distribute the total load equally over $2k$ nodes $\{s, \dots, v_{2k}\}$, to be then delivered to t via the (upward) link connected to w_1. Thus, under an optimal waypoint setting, the aggregate flow splits into $2k+1$ equal-size subset of unit-size flows, and the load on each link (v_i, w_1) is $D/(2k+1) \geq D/(2W+1)$. Since the capacity of links connected to w_1 is 1, the maximum link utilization is at least $m \ln(m)/(2(W+1)) \in \Omega(n \log n/W)$, which concludes Lemma 2.3.14.(ii).

Since $\textsc{Joint} = 1$ due to Lemma 2.3.13, in all the assumed weight settings, $R_{\text{WPO}} = \text{WPO}/1 \in \Omega(n \log n/W)$. □

Proof of Theorem 2.3.15. Let $\mathcal{I}_3$, $\mathcal{I}_4$ and $\mathcal{I}_5$ denote, respectively, TE-instances 3, 4 and 5. Let $\mathcal{N}_3$, $\mathcal{N}_4$ and $\mathcal{N}_5$ denote their respective network instances. Recall that $\mathcal{N}_5$ contains $\mathcal{N}_3$ and $\mathcal{N}_4$ as sub-networks. Any (s,t)-flow in $\mathcal{N}_5$ first traverses through the sub-network $\mathcal{N}_3$ and then through $\mathcal{N}_4$ before reaching its target t. Moreover, the flow can traverse only in one direction between the two sub-networks, that is, via the connecting link (t_3, s_4) (see Instance 5). Hence, the optimal weight setting for $\mathcal{I}_5$ consists of two separate optimal weight settings for the two sub-instances. Therefore, by Lemma 2.3.12, the gap R_{LWO} under the optimal weight setting is $R_{\text{LWO}}(\mathcal{I}_5) \geq$

$$\max\{R_{\text{LWO}}(\mathcal{I}_3), R_{\text{LWO}}(\mathcal{I}_4)\} \geq R_{\text{LWO}}(\mathcal{I}_3) \in \Omega(n \log n/W).$$

Similarly, for the gap R_{WPO} under an optimal waypoint setting, we have $R_{\text{WPO}}(\mathcal{I}_5) \geq \max\{R_{\text{WPO}}(\mathcal{I}_3), R_{\text{WPO}}(\mathcal{I}_4)\}$. If the given weight setting is arbitrary, uniform, or inverse of capacities, then by Lemma 2.3.14.(i), $R_{\text{WPO}}(\mathcal{I}_5) \geq R_{\text{WPO}}(\mathcal{I}_4) \in \Omega(n \log n/W)$. Otherwise, the given weight setting is the optimal one from LWO, in which case Lemma 2.3.14.(ii) implies $R_{\text{WPO}}(\mathcal{I}_5) \geq R_{\text{WPO}}(\mathcal{I}_3) \in \Omega(n \log n/W)$. By Definition 2.3.3, the instance $\mathcal{I}_5$ implies the TE gap

$$R^* \geq \min\{R_{\text{LWO}}(\mathcal{I}_5), R_{\text{WPO}}(\mathcal{I}_5\} \in \Omega(n \log n/W). \qquad \square$$

Proof of Lemma 2.5.3. Consider all nodes sorted in the reverse topological ordering of the DAG G^* (Line 2.1.2). We define a succession of capacity assignments $c^1, \dots, c^n$ as follows. We let $c^n = c$ (the original capacity assignment). Under the ith capacity assignment for $2 \leq i \leq n$, the capacity of each link $\ell \in out(v_j)$, $1 \leq j \leq n$, is

$$c^i(\ell) = \begin{cases} \min_{\ell' \in out(v_j)} ec(\ell') & \text{for } j > i, \\ c(\ell) & \text{for } j \leq i. \end{cases}$$

That is, if $j < i$ then the c^i capacity of every outgoing link of v_j equals the smallest e-capacity of these links. Else, $j \geq i$ and all these links have their original capacities. Let f^i be the maximum (s,t)-flow in the network (G^*, c^i). Note that f^i splits optimally at nodes $v_1, \dots, v_i$, and it splits evenly at nodes $v_{i+1}, \dots, v_n$. Hence, f^1 splits evenly at every node, which implies $|f^1| = ec(s)$. Moreover, f^n splits optimally at every node and $|f^n| = |f^*|$. Trivially, the maximum flow does not decrease when link capacities increase. Therefore, these flows satisfy

$$ec(s) = |f^1| \leq \cdots \leq |f^n| = |f^*|. \tag{3}$$

Next, we bound the increase in the maximum flow between consecutive flows f^i and f^{i-1}. Recall that every flow in $f^1, \dots, f^{i-1}$ splits evenly at v_i, and every flow in $f^i, \dots, f^n$ splits optimally at v_i. Intuitively, the increase of flow specified by $|f^i| - |f^{i-1}|$ is due to switching to the optimal splitting at v_i, i.e., due to the increase of the flow passing through v_i. For flows that split evenly at v_i, we have $f^j(v_i) \geq ec(v_i)$, $1 \leq j \leq i-1$, and for flows that split optimally at v_i, we have $f^j(v_i) \leq \sum_{\ell \in out(v_i)} ec(\ell)$, $i \leq j \leq n$. Therefore,

$$|f^i| - |f^{i-1}| \leq \sum_{\ell \in out(v_i)} ec(\ell) - ec(v_i) \leq \lceil \ln(\Delta_i) \rceil \cdot ec(v_i).$$

If $ec(v_i) \leq ec(s)$ then the increase in the maximum flow is

$$|f^i| - |f^{i-1}| \leq \lceil \ln(\Delta_i) \rceil \cdot ec(s). \tag{4}$$

Else, $ec(v_i) > ec(s)$. Observe that in f^1, the portion of the maximum (s,t)-flow that passes through v_i, that is $f^1(v_i)$, satisfies

$$f^1(v_i) = \min\{ec_{v_i}(s), ec_t(v_i)\}. \tag{5}$$

That is, the maximum (s,t)-flow passing through v_i under c^i equals the smaller of the two effective capacities, one between s and v_i, and the other between v_i and t.

By definition, $f^1(v) \leq |f^1| = ec_t(s)$, which extends (5) to

$$f^1(v_i) = \min\{ec_{v_i}(s), ec_t(v_i)\} \leq ec_t(s). \tag{6}$$

The assumption $ec_t(v_i) > ec_t(s)$ simplifies (6) to

$$f^1(v_i) = ec_{v_i}(s) \leq ec_t(s). \tag{7}$$

Since each flow $f^1, \ldots, f^i$ splits evenly at every node $s, \ldots, v_{i+1}$, they are constrained by the e-capacity between s and v_i, i.e., the portion of each such flow through v_i is at most $ec_{v_i}(s)$. That is, $f^j(v_i) \leq ec_{v_i}(s)$ for $1 \leq j \leq i$. From (3) and (7), we obtain

$$ec_{v_i}(s) = f^1(v_i) \leq \cdots \leq f^i(v_i) \leq ec_{v_i}(s),$$

which implies $f^1(v_i) = \cdots = f^i(v_i) = ec_{v_i}(s)$.

Therefore, in the case where $ec(v_i) > ec(s)$, we have

$$|f^i| - |f^{i-1}| = f^i(v_i) - f^{i-1}(v_i) = 0. \tag{8}$$

We sum up the increase between consecutive pairs of flows, and by applying (4) and (8), we obtain

$$\begin{aligned} \sum_{1<i\leq n} (|f^i| - |f^{i-1}|) = |f^n| - |f^1| &\leq \sum_{1<i\leq n} \lceil \ln(\Delta_i) \rceil \cdot ec(s) \\ &\leq n \cdot \lceil \ln(\Delta) \rceil \cdot ec(s), \end{aligned} \tag{9}$$

which implies the claim since $|f^1| = ec(s)$ and $f^n = f^*$. □

Chapter 3

Resilient Routing

Static fast failover mechanisms support a fast reaction to failures without waiting for a reconvergence. This chapter provides formal models and identifies fundamental tradeoffs on what can and cannot be achieved for specific models of static resilient routing. The contributions are mainly algorithms that provide guaranteed provable resiliency under arbitrary multi-failure scenarios.

3.1 Introduction

We provide the results under two settings: with and without access to the explicit information of link failures along a route. Sections 3.2 and 3.3 respectively focus on these two settings separately.

3.2 Resilient Routing with Carrying Failures

When packets carry the information of previously encountered failed links, resilient routing reduces to the seemingly simple task of computing a route that does not include those failed links. However, there are practical challenges that are the focus of this section.

Motivation: Segment Routing

Segment Routing (SR) [73, 76, 129] emerged to address operational issues of MPLS-based traffic engineering solutions (see [185] for a recent survey). SR provides a fine-grained flow management and traffic steering, allowing to leverage a potentially much higher path diversity than shortest path control planes such as ECMP. At the same time, SR overcomes the scalability issues of MPLS networks as it does not require any resource reservations and states on all the routers part of the virtual circuit. SR is also attractive as, by relying on existing routing protocols, it is easier to deploy than OpenFlow-based solutions.

While the benefits of segment routing in terms of traffic engineering and scalability have been articulated well in the literature, less is known today about how to provide resilience to link and node failures. In particular, we in this section are interested in *Fast Reroute (FRR)* approaches, which handle failures quickly and without invoking the control plane or having to wait for shortest path reconvergence, using statically predefined failover rules. We are especially interested in highly robust FRR schemes which even tolerate *multiple* link failures as they are more likely to arise in larger networks (e.g., data center [91], backbone [138] or enterprise [170] networks) as well as due to shared risk link groups, and for these reasons also constitute a main concern of network carriers [188].

In a nutshell, segment routing is reminiscent of MPLS, in that it also relies on *label stacks* in the packet header: the label at the top of the stack defines the next node. However, unlike MPLS:

1. Segment routing leverages a *source routing* paradigm, in which entire paths are pushed at the network edge (resp. tunnel ingress). Thus, unlike in MPLS networks, there is hence no need for lookup tables to replace (e.g., swap) labels at each traversing node.

2. The next node (the node on the top of the stack) does not have to be directly adjacent to the current node. In this case, to transport packets to the next node, segment routing relies on *shortest path routing* (a "segment").

The definition of intermediate points in the label stack allows to increase the path diversity beyond shortest paths. However, intermediate points can also be used for Fast Rerouting (FRR): if a link along the current shortest path from s to t failed, say at some node u, an alternative intermediate destination (or waypoint) w can be defined to reroute around the failure. In this case, node v receives a label stack "t" (only the destination node t is on the stack), pushes the intermediate destination w (a waypoint), and hence sends out a packet with label stack "wt". Once the packet reaches node w, this node is popped from the stack and the packet (with stack "t") routed to t. If additional failures occur, additional intermediate destinations can be defined recursively.

The Challenge

The key challenge in the design of fast rerouting schemes is that failover rules need to be proactively and statically *pre*-configured: there is no time to recompute paths or communicate information about failures up- or downstream. This also implies that FRR rules can only depend on *local* knowledge and in particular, are *oblivious* to possibly additional failures occurring downstream. Without such global knowledge about failures, however, if no provisions are made by the algorithm defining the failover rules, the resulting routes may become inconsistent (e.g., form a loop). For example, if due to a failed link e, a node v reroutes the traffic from s to t along an intermediate segment to w, it can be undesirable that an additional failure of a link e' in the segment from v to w will steer the traffic back to e again. Moreover, besides ensuring connectivity even under multiple link failures, the FRR mechanism should be *efficient*, and in particular, only require a small number of additional failover segments and a small amount of extra header space.

3.2.1 Contributions

This section initiates the systematic study of static fast failover in segment routing. We present a number of insights on what can and cannot be achieved in fast failover in SR, as well as on potential tradeoffs. In particular:

§3.2.3 We show that existing solutions for SR fast failover, based on TI-LFA [85], do not work in the presence of two or more failures. Furthermore, any TI-LFA extension for k failures must provision for $2k + 1$ stack segments.

§3.2.5 We show that at least in principle, existing fast rerouting techniques based on disjoint arborescences known from the literature considering non-SR networks [41, 42, 45], can be emulated in segment routing as well. However, this emulation comes at a high overhead.

§3.2.6 We identify an interesting and fundamental tradeoff between the efficiency and robustness of failover. In particular, we show that any failover scheme for SR, without extensions, which tolerates at least *two* failures, can be forced to use very costly routes even in the presence of a *single* failure.

§3.2.7 We present an efficient algorithm which preserves reachability even under multiple link failures. Our algorithm, called TI-MFA, is an extension of TI-LFA, and comes with provable correctness guarantees. It is also evaluated in simulations using Rocketfuel topologies; the latter demonstrate TI-MFA's benefits over existing approaches.

We furthermore study the effect of *flushing* the label stack in case of a failure, i.e., removing intermediate destinations to optimize the following failover path. Maybe surprisingly, the failure rate of TI-LFA nearly doubles in this setting in our simulations.

In the remainder of §5.1, we discuss the novelty and limitations of our contributions, along with further related work. We lastly conclude in §5.4, also outlining possibilities for future work.

Novelty

Fast rerouting in SR differs from fast rerouting in other contexts (such as MPLS and OpenFlow) [34, 42, 45, 56, 153, 177, 178] in that a segment relies on shortest path routing and is subject to traditional control plane mechanisms such as ECMP. As we will show in this section, the constraint that segment paths need to be shortest makes the underlying algorithmic problem very different. Having that said, and as we also show that in this section, one can in principle *enforce* certain failover routes and emulate mechanisms such as arborescence routing [42]. This however does not only go against the fundamental principles of SR, but also comes at a higher overhead (in terms of the number of segments that need to be pushed). Finally, to the best of our knowledge, TI-LFA [85] is the only existing proposal to provide node and link protection in SR. In particular, Francois et al.'s approach builds upon IP-FRR concepts such as *Remote LFAs (RLFA)* and remote *LFAs with Directed Forwarding (DLFA)*.

However, as we will show in this section, these approaches fail in the presence of more than link failure. We thus rely on a concept from [127], where previously hit link failures are inserted into the packet header. The conceptual idea is that then a packet cannot hit the same failed link twice, we provide more details of their work in Section 3.2.8. The concepts in [127] differ from our ideas in the sense that we employ segment routing, whereas [127] either pushes the next hops one at a time or inserts complete paths into the packet header.

Finally, we note that our model is very different from models which invoke the control plane, wait for reconvergence or allow to signal failures upstream: in this case, improved failover schemes can be computed, e.g., using a linear programming model and also accounting for the restoration of bandwidth [98]; however, such approaches come with a communication overhead and delay.

Limitations

As our section is a first exploration of the *fast* RR paradigm for segment routing beyond a single failure, we restrict our model for now to not include *later*, relatively *slow* convergence effects. This allows us take an unrestricted view on the immediate effects of link failures. In a practical setting, it can also be modeled by turning off reconvergence protocols, as our algorithms guarantee delivery if the network remains connected.

3.2.2 Related Work

There exists much interesting literature on the architectures and use cases for segment routing, and we refer the reader to the IETF documents [73, 76] as well as the works by Filsfils et al. [71] and David Lebrun [129] for a good overview. Existing algorithmic work on SR mostly revolves around flow control, traffic engineering and network utilization [26, 52, 133, 182], but other use cases are considered such as network monitoring [17]. Optimization problems typically include the minimization of the number of segments required to compute segmented paths [92]. Salsano et al. [163] propose methods to leverage SR in a network without requiring extensions to routing protocols, and Hartert et al. [101] propose a framework to express and implement network requirements in segment routing.

3.2.3 Limitation of Existing Approaches

In order to acquaint ourselves with the problem and in order to show the challenges and limitations of existing approaches, we first revisit the TI-LFA approach [85].

Here and throughout this section, we will only consider the commonly studied case of symmetric networks. In other words, if P is the shortest path from v to w, then traversing P in the reverse direction is also the shortest path from w to v. Furthermore, we assume all link weights to be positive and that subpaths of a shortest path are shortest paths as well.

This symmetry allows TI-LFA to adequately push segments when a packet hits a failed link $e = (v, w)$ at node v: Let v_e-space be the set of all nodes reachable from v via shortest paths in G, where the link e will not be used, i.e., all nodes where the current routing will not hit the failed link. We can define t_e-space analogously for the destination t.[1]

The symmetry property now yields that the v_e- and the t_e-space overlap or at least have adjacent nodes, if e does not disconnect the graph.[2] For a proof intuition, imagine some node u would be contained in neither space: 1) if the shortest paths to u from v, t traverse e in different directions, then one of both of them could take a shortcut and not use e, and 2) if both shortest paths traverse e in the same direction, then the shortest path from v to t would not use e.

Hence, v can push pre-computed segments on top of the label stack: 1) if the v_e- and the t_e-space overlap, then a joint node that minimizes the routing distance, and 2) in case of mere adjacency, a border node along with a label of a link that connects both spaces, again minimizing total distance. As such, pushing two labels on the label stack always suffices for a single failure.

3.2.4 TI-LFA loops indefinitely for two link failures

For a simple example why TI-LFA is no longer guaranteed to be correct, we refer to Figure 3.1. There are three paths to the destination t, a left one from v_ℓ via e_ℓ, a middle one from v_m via e_m, and a right one from v_r via e_r. Assume e_m fails, leading w.l.o.g. to v_m pushing the label for v_r on the stack. If now e_r fails as well, then v_r pushes the label for v_m on the stack, i.e., the packet loops indefinitely between v_m and v_r, even though there is still a path via e_ℓ: the problem is that TI-LFA does not differentiate regarding where the packet is coming from, which we will tackle in Section 3.2.5 with respect to the incoming port and in Section 3.2.7 by storing the last hit failure(s) in the packet header.

Observation 3.2.1. *Even if the network is still connected after two link failures, TI-LFA can loop indefinitely.*

[1]Usually, the terms P- and Q-space are used in the literature, but we introduce node/link-specific notation here for ease of reference.

[2]For an in-depth tutorial, we refer to the topic *Topology Independent LFA (TI-LFA)* at `http://www.segment-routing.net/tutorials/`.

In order to guarantee that the packet still reaches its destination, we need to implement a local (pre-computed) mechanism that enforces that the third path via e_ℓ will be eventually taken, via segment routing.

However, this requires an increase in the number of labels pushed, for which we provide an example construction: for the case of two failures, replace the link between v_m and v_r with the construction shown in Figure 3.2, analogously for the link between v_ℓ and v_m. Then, four segments are needed to reach t from v_r when the links e_r, e_m are failed, already in the case when v_r is aware that both these links are unavailable. This idea can be extended to k link failures, each requiring two segments, by extending the network from Figure 3.1 appropriately.

Observation 3.2.2. *Even if all k link failures in the network are known, at least $2k$ further segments must be provisioned s.t. the packet can reach the destination without further additions to the label stack on-route.*

We quickly summarize our findings so far: 1) TI-LFA can loop already with two link failures, as knowledge of prior actions is not used/available, 2) unless further segments are to be pushed on-route, a label stack proportional to the link failure number is required.

In the following, we now tackle these two observations. In Section 3.2.5 we use the incoming port of the packet (the in-port) as “prior knowledge” and perform ongoing additions to the label stack, by emulating known fast local reroute techniques. While this seems efficient at first, this approach suffers from a high overhead in the path length and essentially ignores the benefits of segment routing. Furthermore, known local fast reroute techniques cannot guarantee reachability in all cases.

As thus, in Section 3.2.7, we take the approach of storing past link failures in the packet header and push more segments at once, which leads to provable reachability of the destination, as long as the network remains connected under link failures. Our simulations show that for two link failures and flushing, we have an extra label stack size of at most 4 (+t), which matches Observation 3.2.2 for $k = 2$.

3.2.5 A Robust But Inefficient Solution

Literature on static resilient routing for non-SR networks provides powerful techniques to preserve connectivity even under many link failures [41, 42, 45]. In the following, we show that such approach can be employed also in SR networks.

The deterministic techniques of Chiesa et al. in [41, 42, 45] rely on building pre-computed routing rules that match on 1) the destination, 2) incident failures, and (unlike classical segment routing) 3) the incoming port of the packet. These works consider k-link-connected networks and aim at reaching resiliency up to $k-1$ failures[3]. But they do not guarantee reachability of the destination t if the only invariant is that the network remains connected.

[3]At least a resiliency of $\lfloor k/2 \rfloor$ is guaranteed, with various graph families also reaching $k-1$, we refer to [41, 42, 45] for details.

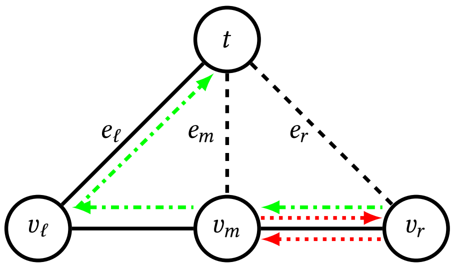


Figure 3.1: Illustration of a network where TI-LFA loops permanently with two link failures e_m, e_r, drawn dashed, when reaching v_m. Assume w.l.o.g. that TI-LFA picks the backup path via e_r: then, v_r will push segments (v_m) for the backup path via e_m, leading to a permanent loop, drawn in dotted red. Our proposed algorithm TI-MFA will pick the dash-dotted green path when reaching v_r after being rerouted from the initial node v_m, successfully reaching the destination t.

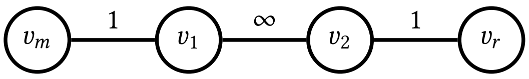


Figure 3.2: Replacing the link between v_m and v_r with a path of weights $1, \infty, 1$, the remaining network from Figure 3.1 is omitted here for ease of viewing. When segments are pushed at v_r to reach v_m, without going across the failed link e_r, one does not suffice: when being at v_2, the shortest path to v_m is via the failed link e_r (not drawn here). Hence, a second segment is needed, the link between v_2 and v_1. When an analogous construction is applied to the link between v_ℓ and v_m, two further segments are needed to circumvent e_m, with identical arguments. Hence, to reach t from v_r without pushing further segments along the way, v_r needs to push four segments.

We can emulate these techniques if we incorporate a match on the incoming port or on the top element of the stack before popping it. However, the main obstacle is that the techniques of Chiesa et al. do not route along shortest paths, but rather enforce single-hop routing mechanisms, guiding the packet to the destination one step at a time, possibly along long detours in comparison to the global shortest paths. We overcome this obstacle by putting (“forcing”) a link on top of the label stack each time the packet arrives at a node.

Now, we can install rules on the switches emulating the techniques by Chiesa et al.: 1) at most one extra label will be on the top of the destination (a link), which is popped when reaching the next node, leaving the destination on top of the label stack to match, 2) incident failures may inherently be matched upon in SR, and 3) the incoming port as described above.

However, this emulation approach comes with a high overhead and essentially ignores the main features of SR networks by forcing each link along the route individually.

3.2.6 The Price of Resiliency

We showed that robustness can be achieved using segment routing, but that our emulation approach removes many inherent efficiency aspects. Unfortunately, as we show in the following, there is an inherent price of local resiliency.

Consider the example in Figure 3.1, but assign the following link weights: 1 for e_m and e_r, ∞ for e_ℓ. Alternatively, replace e_ℓ with a path of arbitrary length. If, similar to TI-LFA, we protect against the failure of only one of the links e_ℓ, e_m, e_r towards the destination t from v_r, v_m, v_ℓ, the failover path can always be chosen to have a short length, respectively small link weights. However, as soon as we protect against two link failures, this resiliency comes at a price. As the label-pushing at the second failure only takes the 1) incident failure and 2) the destination t[4] into account, the node is unaware of the first failure – and rerouting decisions are only performed when a failure is hit. Hence, either the failover for e_m or e_r must reroute via the expensive e_ℓ, or risk looping indefinitely, similar to the example in Section 3.2.4. We note that the above example from Figure 3.1 can easily be extended to a 3-link-connected graph. This highlights the price of resiliency against two failures in local schemes: even if only one failure occurs, an inefficient failover path has to be taken.

3.2.7 Efficient Resilient Segment Routing

In order to overcome the identified tradeoff, we make the following observation: while static failover schemes are inherently oblivious to failures downstream of the path, at least we could remember the already encountered failures.

Note that in fact, methods such as TI-LFA implicitly assume global knowledge of all current failures: as the rerouting decision is performed at a node incident to the single failure, all other links are assumed to be available. Hence, if there is still a path to the destination, packets will reach it. Nonetheless, as we showed before, if the rerouting node is not aware of the second failure, TI-LFA loops indefinitely already in simple topologies. As in [127], we thus propose to store already hit failures in the packet header for two reasons:

1. The header space usage scales linearly with the number of hit failures, where a single failure uses similar space as pushing an intermediate destination on the label stack.
2. Rerouting via intermediate segments is performed locally, based off pre-computed table entries..

We note that all described computations can be performed ahead of time and stored in a static routing table, whose size scales with the number of failure combinations.

[4]As v_r will be popped from the label stack when reaching v_r. If the label v_r would also be taken into consideration, we can use a construction along the lines of Figure 3.2 to ensure that the label stack at v_r only consists of t.

More specifically, our so-called *Topology Independent Multi-Failure Alternate* (*TI-MFA*) algorithm works as follows, described from the viewpoint of the node v where the packet hits another failed link:

1. Flush the label stack except for t.
2. Based on all link failures stored in the packet header, determine the shortest path P to the destination t in the remaining network G'.
3. Add segments to the label stack of the packet as follows:
    - Index the nodes on P as $v = v_1, v_2, \dots, v_x = w$. Compute the node v_i on P with the highest index s.t. the shortest path from v is identical in G' (with failures) and G (without failures) and set it as the top of the label stack. If this node is v, push the link $(v_1, v_2 = v_i)$ as the top of the label stack. For the second item on the label stack, start over with v_i as the starting node, etc., until $v_i = w$.

We next show TI-MFA to be correct in theory and efficient in practice, only using small stack sizes under two link failures.

3.2.8 Formal Correctness Analysis

We begin by first going a step back to explain the approach of Lakshminarayanan et al. [127], which comes in two flavors called FCP (Failure-Carrying Packets) and source-routing FCP:

- In FCP, a node decides on the next hop by considering the failures the packet hit so far and the packet destination t. Delivery in a connected network is guaranteed by the fact that a packet will not hit a link failure twice: else, an intermediate node would have ignored that this link failure was inserted into the packet header. In our context, FCP thus resembles the hop-by-hop forwarding in Section 3.2.5, with the advantage that FCP can aim for short paths and stronger reachability, but on the other hand, the emulation idea in Section 3.2.5 did not add failures to the packet header.
- Source-routing FCP behaves analogously, but rather inserts the whole path to the destination into the packet header, consistent with the view based on the link failures stored in the packet header. Should the packet hit a new failure, it is added to the header, the old path is removed (as in our case), and a new path is inserted into the header.

The main difference between TI-MFA and the work of Lakshminarayanan et al. [127] is that in segment routing, only few intermediate waypoints are provided, whereas FCP implicitly provides the whole path: either on a hop-by-hop basis or by inserting the path into the packet header. We now show the correctness of TI-MFA.

Theorem 3.2.3.
Let G be a network where k links fail, s.t. the remaining graph G' is connected. TI-MFA routes successfully to the destination.

Proof. Our proof will be via nested induction arguments over the number of failures hit. Clearly, if no failures are hit, the destination is reached. Similarly, if only one failure is hit at a node v, TI-MFA behaves identical to TI-LFA: the packet is routed along a failure-free backup path. Note that due to the pushed segments, the shortest-path segment routing is identical in G (without failures) and G' (with failures) from v.

We now assume the packet already hit $x - 1$ failures and hits failure x on its path. We distinguish two cases:

- $x = k$: Then, due to construction (Item 3), the shortest-path segment routing is again identical in G' and G.
- $x > k$: Assume some link failure $x + 1$ is hit next (the first x failures can no longer be hit, due to construction), else we will reach the destination. Then, we invoke our inductive construction again: as an invariant, already hit failures can no longer be hit, i.e., eventually we will either have hit all failures or reach the destination. However, once all failures are hit, this implies that the destination will be reached via the path P.

Note that the correctness of the above arguments relies on the assumption that the network $G'$is connected. □

We would like to note that for two link failures, we can show that the segment stack does not need to be flushed s.t. only the destination remains[5]: let P_1 be the path the packet takes as a backup, after hitting the link failure e_1, if no second link failure is hit. To hit a second failure e_2, the link e_2 must be on P_1. If the label stack is then empty except for t, the destination will be reached by the above proof arguments. Else, the next label stack item v_i is *on* P_1, behind e_2. As the rerouting is aware of both link failures e_1, e_2, v_i will be reached as the network remains connected. The remaining part of P_1 beyond v_i cannot contain the failed links e_1 or e_2.

Corollary 3.2.4. *If the number of link failures is at most $k = 2$ and the remaining network is connected, TI-MFA is correct even if the label stack is not flushed on a link failure hit.*

3.3 Resilient Routing without Carrying Failures

This section studies a static resilient routing model that does not require storing any failure-related information in the packet header.

[5]We assume that if a link is on top of the label stack and this link fails, that popping this unavailable link does not count as flushing.

The need for a more reliable network performance and quickly growing traffic volumes led, starting from the late 1990s [63], to the development of more advanced approaches to control the routes along which traffic is delivered. Multipath-Label Switching (MPLS) was one of the first and most widely deployed alternatives to traditional weight and destination based routing (such as OSPF), enabling a per-flow traffic engineering. Recently, *Segment Routing (SR)* [76, 130] has emerged as a scalable alternative to MPLS networks: SR networks do not require any resource reservations nor states on all the routers part of the route (the *virtual circuit*). SR networks are also attractive for their simple deployment; in contrast to, e.g., Software-Defined Network (SDN) and OpenFlow-based solutions, they rely on existing protocols such as IPv6 [181].

We in this section investigate how to enhance SR networks with *(local) fast rerouting* algorithms, to react to failures *without the need to invoke the control plane.* The re-computation (and distribution) of routes after failures via the control plane is notoriously slow [86] and known to harm performance [135]. Also link-reversal algorithms [87] tolerating multiple failures have a quadratic convergence time [35], besides requiring dynamic routing tables. This is problematic as certain applications, e.g., in data centers, are known to require a latency of less than 100 ms [189]; voice traffic [99] and interactive services [107] already degrade after 60 ms of delay. Not surprisingly, reliability is also one of the foremost challenges for network carriers nowadays [187], and in the context of power systems (e.g., smart grids), an almost entirely lossless network is expected [173]. Accordingly, most modern communication networks (including IP, MPLS, OpenFlow networks) feature fast rerouting primitives to support networks to recover quickly from failures.

Designing a fast rerouting algorithm however is non-trivial, as reactions need to be *(statically) pre-defined* and can only depend on the *local* failures, but not on "future" failures, *downstream.* As link failures, also multiple ones, are common in networks [139], e.g., due to shared link risk groups or virtualization, it is crucial to pre-define the conditional local failover rules such that connectivity is preserved (i.e., forwarding loops and blackholes avoided) under *any* possible additional failures. In fact, in many networks, including SR networks, algorithms cannot even depend on already encountered failures *upstream*, as it requires mechanisms to carry and process such information in the packet header; such "failure-carrying packets" [78, 127] require additional and complex forwarding rules. Further challenges are introduced by policy-related constraints on the paths along which packets are rerouted in case of failures. In particular, failover paths may not be allowed to "skip" nodes, but rather should reroute around failed links individually: communication networks include an increasing number of middleboxes and network functions, so-called *waypoints* [8], which must be traversed for security and performance reasons. Without precautions, in case of a link failure, the backup path could omit these waypoints.

Ideally, a local fast rerouting algorithm preserves connectivity "whenever this is still possible", i.e., as long as the underlying network is still *physically* connected. In other words, in a k-(link-)connected network, we would like the rerouting algorithm to tolerate $k - 1$ link failures. We will refer to this strong notion of robustness as *maximal robustness* in the following.

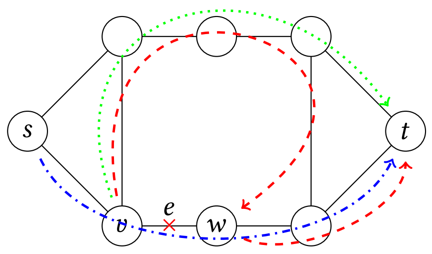


Figure 3.3: Illustration of two different fast failover mechanisms, upon hitting a link failure. The default path is depicted in dash-dotted blue. In the green dotted path, the destination t is reached, but the waypoint w is not traversed. The red dashed walk circumvents the link failure, traverses w, and then reaches t.

Example

Figure 3.3 illustrates an example for the problem considered in this section: how to efficiently circumvent multiple link failures using SR local fast failover mechanisms, such that the original route will be preserved as part of the packets' new route, hence also ensuring waypoint traversal. In this example, while the backup path in dotted green reaches the destination, the middlebox w is not visited. Ideally, we want to circumvent the failed link and then continue on the original route (as depicted in the red dashed walk).

In case of only a single link $e = (v, w)$ failing, one can exploit that both nodes v, w have a globally correct view of the network link states. For example, if a packet hits the failed link e at v, the node v can provide an alternative path to w, after which the packet resumes its original path, as shown in dashed red in Figure 3.3. To this end, each node only needs to be provisioned with one alternative path for each of its incident links.

In Segment Routing, similar to MPLS, each packet contains a label stack, consisting of nodes or links. However, these labels just represent the next waypoint to be reached, the route ("segment") which depends on the underlying routing functions (e.g., shortest path). Once the top item is reached, the corresponding label is popped and the next item on the stack is parsed. As such, the next label does not need to be in the vicinity of the current node, it can be anywhere in the network. For the case of a single link $e = (v, w)$ failure, it has been shown that pushing two items on the label stack always suffices [85], if the network is still connected and a shortest alternative path is chosen.

While SR enables waypoint traversal even after a single link failure [85], dealing with multiple link failures in SR is still not well understood. As observed in [78], the option of choosing the shortest alternative path already fails under two link failures, see Figure 3.4; when e_1 fails (and the dash-dotted blue path as well), the packet will be sent along e.g. e_2, but upon the failure of e_2, the packet is sent along e_1—a forwarding loop (shown in dotted red). In this example, we can easily fix the reachability issues: a failure of e_2 causes rerouting along e_3 (in dashed green, not along e_1), and failure of e_3 causes rerouting along e_1. In other words, e_1 depends on e_2, which depends on e_3, which in turn depends on e_1. As this circular dependency chain has a length of three, two failures of $\{e_1, e_2, e_3\}$ cannot induce a forwarding loop when routing to w. We will later formalize and extend these ideas, generating dependency chains of length $\geq k$ for k-dimensional hypercubes.

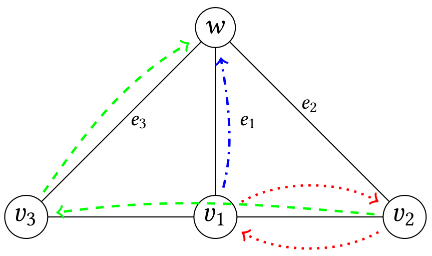


Figure 3.4: Example illustrating how local fast failover methods for a single link failure can loop under two link failures, as shown in [78]. When the dash-dotted blue default route between v_1 and w fails, v_2 can be pushed as a segment, to in turn reroute along e_2. However, when v_2 uses e_1 via v_1 as a failover for e_2, then failing both e_1 and e_2 leads to a permanent forwarding loop, as depicted in dotted red. In order to route successfully under both e_1, e_2 failing, v_2 has to push segments v_1, v_3 to route along e_3, as depicted in dashed green.

3.3.1 Contributions

We initiate the study of fast reroute algorithms for emerging Segment Routing networks which are 1) resilient to a maximum number of failures (i.e., are *maximally robust*), 2) respect the path traversal of the original route, and 3) are compatible to current technologies in that they do not require packets to carry failure information: routing tables are static and forwarding just depends on the packet's top-of-the-stack destination label and the incident link failures.

Our main result is an efficient algorithm which provably provides all these properties on hypercube networks, as they are commonly used in data centers (see e.g., [148]). Furthermore, we formulate the underlying optimization problem as an integer linear program for general graphs, and provide first exploratory insights on the practical performance of segment routing under multiple link failures.

3.3.2 Related Work

Most modern communication networks support some form of resilient routing, and the topic has already received much interest in the literature. There exists much literature on single [57, 145, 186, 191], double [46, 150], and more [56] failure scenarios, the latter being motivated by, e.g., shared risk link groups [172], attacks [179], or simply node failures which affect all incident links [15, 56, 91, 170]. The spectrum of solutions is broad as well, with some solutions providing only heuristic guarantees [46, 150], some schemes exploiting packet-header rewriting [38, 56] (which however is not always supported in existing networks) or packet-duplication [97] (which however comes with overheads). Furthermore, there is also work that aims at quickly optimizing network behavior after link failures have propagated, e.g., by pre-computing how to rescale traffic at ingress routers once these nodes are fault-aware [134]. However, such mechanisms do not provide protection for packets during convergence.

An interesting line of research studies mechanisms which do not require any additional information in the packet header, such as the works by Feigenbaum et al. [109], by Chiesa et al. [44, 45] (establishing an interesting connection to arc-disjoint graph covers), by Elhourani et al. [56], by Stephens et al. [177, 178], by Borokhovich et al. [34], by Pignolet et al. [153] (establishing an interesting connection to distributed computing problems without communication [137]), and by Foerster et al. [80]. However, these solutions do not require failover paths to traverse the nodes of the original path and do not account for the specific properties of the networks considered in this section. The former is particularly motivated by the advent of (virtualized [59]) middleboxes [39], and is also known as *local protection scheme* in MPLS terminology [166].

Our work is situated in the context of MPLS and Segment Routing (SR) networks where routing is based on stacks and more specifically, the top of the stack label [152]. While the design of resilient routing algorithms has received much attention already in the context of MPLS, see e.g., [95] and [114, 166] and references therein, existing research on SR networks mainly revolves around flow control, traffic engineering and network utilization [26, 52, 133, 182], or network monitoring [17], see the works by Filsfils et al. [71] and Lebrun et al. [54, 130–132] for a good overview. Optimization problems typically include the minimization of the number of segments required to compute segmented paths [92]. Salsano et al. [163] propose methods to leverage SR in a network without requiring extensions to routing protocols, and Hartert et al. [101] propose a framework to express and implement network requirements in SR. Only little is known today about fast rerouting in SR networks. In [78], it has been shown that existing solutions for SR fast failover, based on TI-LFA [85], do not work in the presence of two or more failures. However, [78] relies on failure-carrying packets, which is undesirable as discussed above and we overcome in the current section. Finally, we in this section considered hypercubes, which have recently been studied for local fast failover algorithms in [45, 79] as well. While for a single link failure, the general approach of François et al. [85] can be used, we are not aware of any approaches that (conceptually) employ Segment Routing for local fast failover in hypercubes for multiple failures.

Organization

The remainder of this section is organized as follows. We first introduce necessary model preliminaries in Section 5.1.1, followed by our main result in Section 3.3.4, where we provide a maximally robust SR failover scheme for k-dimensional hypercubes. We cover related work in Section 5.1.3 and conclude our study in Section 5.4, where we also provide further insights which we believe to be useful for future work, in the form of an integer linear program formulation for general graphs and a brief investigation regarding testbed experiments.

3.3.3 Model

In this section, we start by providing model and notation preliminaries. We will consider undirected graphs $G = (V, E)$, where the links may be indexed according to some (possibly arbitrary) ordering, with $\ell_i \in E$ denoting the ith link. All routing rules have to be pre-computed and may not be changed during the runtime (e.g., after failures). We will only allow routing rules that match on 1) the packet's next destination (i.e., the top of the label stack)[6], and the 2) incident link failures.[7] When a packet hits a failed link $\ell = (u, v)$ at some node u, the current node u may push a set of pre-computed labels on top of the current label stack, in order to create a so-called backup path to v (which can also be traversed in reverse from v to u).

Definition 3.3.1. *A* backup path *for a link ℓ is a simple path (not containing ℓ) that connects the endpoint of the link ℓ. Let $\mathcal{P}$ be the set of all backup paths in a graph. An injective function $BP : E \to \mathcal{P}$ that maps one backup path to each link is a* backup path scheme.

When the packet reaches the current top label, the respective label is popped and the underlying label is set as top label. As such, via backup paths, the incoming packets that normally travel through ℓ are rerouted around the link to the respective endpoint, circumventing the failure. Hence, our model preserves the intermediate visits (i.e., all possible waypoints) and their order in a subset of the traversed route, possibly introducing repeated visits [7]. In the following, we will investigate backup path schemes that guarantee packet delivery even under multiple failures. To this end, we need to ensure that the backup paths do not contain infinite forwarding loops, for their specified maximum number of failures. More formally:

Definition 3.3.2. *A backup path scheme $BP(.)$ is called f-*resilient *if and only if there does not exist a subset of links $L \subseteq E, |L| \leq f$ such that for some ordering σ :*

[6]In practice, one could also imagine matching on other header fields, such as the packet's source, and also the incoming port. However, our algorithms do not require these additional inputs.

[7]In other words, only the endpoints u, v of the failed link $(u, v) = \ell \in E$ are aware of the failure.

$\{0,\dots,|L|-1\} \to \{0,\dots,|E|-1\}$, $\forall j < |L| : \ell_{\sigma(j+1 \pmod{|L|})} \in BP(\ell_{\sigma(j)})$. *We refer to the inclusion relation ($\in$) as* dependency *from $\ell_{\sigma(j)}$ to $\ell_{\sigma(j+1 \pmod{|L|})}$. Equivalently, $BP(.)$ is f-resilient if and only if any* cycle of dependencies *is longer than f.*

In the next section, we will show how to efficiently generate a $(k-1)$-resilient backup path scheme for k-dimensional hypercubes. As k-dimensional hypercubes are k-link-connected, our scheme has ideal robustness.

3.3.4 Resilient Routing on k-Dimensional Hypercubes

This section presents a fast and maximally robust rerouting algorithm on hypercubes, one of the most important and well-studied network topologies [162, 184]. The regular structure of hypercubes makes them an ideal fit for e.g., parallel interconnection architectures [161] or data center [94].

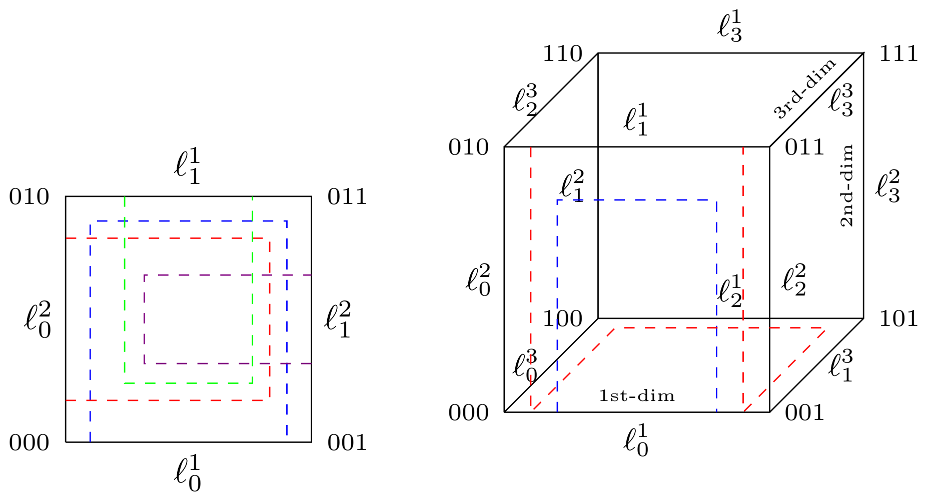
Our study on k-dimensional hypercubes is structured as follows: We first provide an intuition and overview of the $(k-1)$-resilient scheme in Section 3.3.5, providing a formal definition of all backup paths in Term 1. Next, in Section 3.3.6, we introduce some useful technical preliminaries for the correctness proof of our scheme, which is presented in Section 3.3.6.

3.3.5 Overview of the Fast Local Failover Scheme

We label the nodes in a k-dimensional hypercube (k-*cube*) with tuples $(b_k, b_{k-1}, \dots, b_1)$, $\forall i \in [k] : b_i \in \{0,1\}$, such that the *origin* node has the label $\{0\}^k$. A hypercube link is denoted by an ordered pair of binary node labels (a,b) s.t. $a, b \in \{0,1\}^k, a < b$, where the two labels differ in one bit. Additionally, a link is said to be in dimension $d, d \in [k]$, if and only if a and b differ only at their dth bit. We refer to them as d-*dim* links. For convenience, we treat a hypercube as a set of links grouped by their dimension, within each dimension sorted according to the following bitwise comparison. For $x \in \{0,1\}^k$, let $x^{>>s} := x >> s$, where $>>$ is right circular shift. Let ℓ_i^d denote the ith link in dimension d, see Figure 3.5. For $\ell_p^d = (a,b)$ and $\ell_q^d = (c,d)$, we have $p < q$ if and only if $a^{>>d} < c^{>>d}$. Lastly, we denote a k-cube by $C_k := \cup_{d,i} \ell_i^d, d \in [k], 0 \le i < 2^{k-1}$.

The idea is to allocate backup paths in k iterations, one for each subset of links in the same dimension, such that the induced dependencies over same-dimension links form cycles of length at least k. However, since there are additional dependency cycles induced by links in different dimensions, we devise a scheme that does not induce any dependency cycle shorter than k (hence $(k-1)$-resiliency follows).

Due to gray coding, starting from any link $\ell_i^d = (a,b)$, by traversing the (unique) pair of incident d'-dim links, we reach the link $N_{d'}(\ell_i^d) = (a', b')$ such that $a' = (2^{d'-1})_2 \oplus a$ and $b' = (2^{d'-1})_2 \oplus b$. Let $\left(L_0^{d'}[\ell_i^d], L_1^{d'}[\ell_i^d]\right)$ denote the (unique) pair of incident d'-dim links, i.e. $L_0^{d'}[\ell_i^d] = (a, a')$ and $L_1^{d'}[\ell_i^d] = (b, b')$. The subscripts 0 and 1 indicate the value at the dth bit position of the links in the pair. Due to symmetry, $N_{d'}(N_{d'}(\ell_i^d)) = \ell_i^d$ and $L_b^{d'}[\ell_i^d] = L_b^{d'}[N_{d'}(\ell_i^d)], b \in \{0,1\}$.



$$
\begin{array}{lll}
BP(\ell_0^1) = \{\ell_0^2, \ell_1^2, \ell_1^1\} & BP(\ell_0^2) = \{\ell_0^3, \ell_1^3, \ell_1^2\} & BP(\ell_0^3) = \{\ell_0^1, \ell_1^1, \ell_1^3\} \\
BP(\ell_1^1) = \{\ell_0^2, \ell_1^2, \ell_0^3, \ell_1^3, \ell_3^1\} & BP(\ell_1^2) = \{\ell_0^3, \ell_1^3, \ell_0^1, \ell_1^1, \ell_2^2\} & BP(\ell_1^3) = \{\ell_0^1, \ell_1^1, \ell_0^2, \ell_1^2, \ell_2^3\} \\
BP(\ell_2^1) = \{\ell_2^2, \ell_3^2, \ell_3^1\} & BP(\ell_2^2) = \{\ell_2^3, \ell_3^3, \ell_3^2\} & BP(\ell_2^3) = \{\ell_2^1, \ell_3^1, \ell_3^3\} \\
BP(\ell_3^1) = \{\ell_2^2, \ell_3^2, \ell_0^3, \ell_1^3, \ell_0^1\} & BP(\ell_3^2) = \{\ell_2^3, \ell_3^3, \ell_0^1, \ell_1^1, \ell_0^2\} & BP(\ell_3^3) = \{\ell_2^1, \ell_3^1, \ell_0^2, \ell_1^2, \ell_0^3\}
\end{array}
$$

Figure 3.5: Illustration of $BP(.)$ on 2 and 3 dimensional cubes. Backup paths are shown with dashed lines.

We formulate the backup path of a d-dim link as a set consisting of one d-dim link and pairs of links. These pairs constitute a joint path, i.e., two paths over the endpoints of detoured d-dim links. We refer to this joint path as a backup path and we always traverse it towards the included d-dim link. However in reality, a packet traverses the two paths in opposite directions, towards and away from the respective d-dim link.

For instance, the backup path of the first 1-dim link (i.e. ℓ_0^1) includes the 1-dim link reached via the incident pair of 2-dim links, and the pair itself (see Figure 3.5): $BP(\ell_0^1) = \{L_0^2[\ell_0^1], L_1^2[\ell_0^1], N_2(\ell_0^1)\} = \{\ell_0^2, \ell_1^2, \ell_1^1\}$ (see Figure 3.5). For the second 1-dim link we use the same pair, but we have to detour ℓ_0^1 in order to avoid conflict:

$$BP(\ell_1^1) = \{L_1^2[\ell_1^1], L_1^2[\ell_1^1], L_0^3[\ell_0^1], L_1^3[\ell_0^1], N_3(\ell_0^1)\} = \{\ell_0^2, \ell_1^2, \ell_0^3, \ell_1^3, \ell_3^1\}.$$

In general, the backup path of ℓ_i^d begins with the pair $\left(L_0^{d+1}[\ell_i^d], L_1^{d+1}[\ell_i^d]\right)$. If the first d-dim link, i.e. $N_{d+1}(\ell_i^d)$, is conflicting, then one continues by detouring this link via the pair of $(d+2)$-dim links and detours further d-dim links, until one reaches a d-dim link that is not conflicting, then traverses this link. Moreover, the jth detour is performed via the pair of $(d+j)$-dim links. Hence the pairs are traversed in the ascending order of consecutive dimensions. We denote the closure form of $N_d(.)$ w.r.t. this ordering as

$$N^{(j)}(\ell_i^d) := N_{d+j}(N_{d+j-1}(\dots N_{d+1}(\ell_i^d)\dots), 1 \le j < k.$$

We can now describe our backup path scheme formally, we refer to Figure 3.5 for an example listing all generated backup paths on the 3-dimensional hypercube. For each dimension $d \in [k]$ and every $0 \leq i < 2^{k-1}$, the backup path of ℓ_i^d is

$$\begin{aligned} BP(\ell_i^d) = \Big\{ & L_0^{d+1}[\ell_i^d], L_1^{d+1}[\ell_i^d], \\ & L_0^{d+2}[N_{d+1}(\ell_i^d)], L_1^{d+2}[N_{d+1}(\ell_i^d)], \\ & \dots, \\ & L_0^{d+r}[N^{(r-1)}(\ell_i^d)], L_1^{d+r}[N^{(r-1)}(\ell_i^d)], \\ & N^{(r)}(\ell_i^d)) = \ell_{i'}^d \Big\}. \end{aligned} \tag{1}$$

The path detours $r - 1$ links, where r is the number of link pairs necessary to have, in order to reach the non-conflicting link $\ell_{i'}^d$ with smallest index. Therefore the path length is $2r + 1$. We will later argue that $r \leq R := \lceil \log k \rceil$.

Alternatively to the explicit formulation in (1), $\ell_{i'}^d$ can be obtained directly using bitwise operations. Assume $\ell_i^d = (a, b)$ and $\ell_{i'}^d = (a', b')$. By comparing a' to a (b' to b), we can see that only the r bits to the left of dth bit are affected, i.e., the bits $d+1$ to $d+R \pmod{k}$. For $x \in \{0,1\}^k$ and $s := R-(k-d)$, we define the increment function that determines the successor link as $inc_{s,d}(x) := (x^{>>s} + (2^d)_2^{>>s})^{>>-s}$. Here the $+$ ignores the carry flag out of the leftmost position. Therefore, $a' = inc_{s,d}(a)$ and $b' = inc_{s,d}(b)$. It is clear that the overall computation takes polynomial time.

In the next section, we will state some necessary observations regarding our hypercube construction, which we will employ for the correctness proof of our scheme in Section 3.3.6.

3.3.6 Proof of Resiliency

According to our backup path formulation (1), the backup path of a d-dim link passes through a d-dim link reached via links in higher dimensions, which are presented in pairs in (1). The backup path possibly detours some other d-dim links along its way. The pairs and the involved d-dim links together resemble a chain-like structure which facilitates describing some properties in this section. We now describe these structures formally.

Definition 3.3.3. *Given a* sequence of dimensions $S_d := (d_i)_{i=0}, d_i \in [k] \setminus \{d\}$, *a* chain *of d-dim links, starting from $\ell_{i_0}^d$, denoted by $C(\ell_{i_0}^d)$, consists of a subset of d-dim links and pairs of d_i-dim links, $d_i \in S_d$. The pairs form two walks over the endpoints of the contained d-dim links. The two parallel walks jointly* traverse *the chain. We denote the chain by $C_{S_d}(\ell_{i_0}^d) := \{\dots, \ell_{i_j}^d, (L_0^{d_j}[\ell_{i_j}^d], L_1^{d_j}[\ell_{i_j}^d]), \ell_{i_{j+1}}^d, \dots\}, j \geq 0, \ell_{i_{j+1}}^d = N_{d_j}(\ell_{i_j}^d)$. Moreover, if $\exists \ell_{i_{j'}} \in C(\ell_{i_0}^d) : \ell_{i_{j'+1}}^d = \ell_{i_0}$, then it is a* closed *chain denoted by C_{S_d}.*

We can directly obtain the following property.

Property 1. *Starting from any link ℓ_i^d, by traversing a chain $C_{S_d}(\ell_i^d)$, assume we arrive back at the same link. Then it must be the case that S_d contains every dimension an even number of times.*

Definition 3.3.4. *A link $(a, b), a < b$ is traversed in* uphill *direction when it is from a. The opposite is a downhill direction.*

Based off this definition, we can categorize the traversal directions.

Property 2. *Consider a closed chain containing the pair $(L_0^{d'}[\ell_i^d], L_1^{d'}[\ell_i^d])$ traversed between the links ℓ_i^d and $\ell_j^d = N_{d'}(\ell_i^d)$. If $j > i$ then the direction from ℓ_i^d to ℓ_j^d is uphill, otherwise downhill.*

Property 3. *By Properties 1 and 2, in a closed chain, the number of traversals in every dimension is even, half of which is in downhill (uphill) direction.*

Intuitively, uphill and downhill traversals cancel each other which consequently turns the joint walks into joint closed walks over the endpoints.

We next study the interaction between chains. Let $S_d, S_{d'}, d' \neq d$ be two sequences of dimensions. We say the chain $C_{S'_d}$ *crosses* the chain C_{S_d} if $\exists P := (L_d^0, L_d^1) \in C_{S_{d'}} : P cut C_{S_d} \neq \emptyset$. That is, $C_{S'_d}$ traverses a pair of d-dim links, at least one of which belongs to C_{S_d}.

Definition 3.3.5. *A* mixed *chain is the concatenation of multiple chains (over several dimensions) that cross each other consecutively. In other words, a mixed chain consists of chains of links in at least two dimensions. Formally, for given dimensions $d, d', d'' \in [k]$ and sequences S_d and $S_{d'}$, assume the chain $C_{S_{d'}} = \{\dots, \ell_x^{d'}, (L_d^0[\ell_x^{d'}] = \ell_y^d, L_d^1[\ell_x^{d'}]), \dots\}$ crosses $C_{S_d} = \{\dots, \ell_y^d, (L_{d''}^0[\ell_y^d], L_{d''}^1[\ell_y^d]), \dots\}$. We concatenate these chains into a mixed chain as $\{\dots, \ell_x^{d'}, \ell_y^d, (L_{d''}^0[\ell_y^d], L_{d''}^1[\ell_y^d]), \dots\}$.*

The observations in the Properties (1), (2), and (3) hold for mixed chains as well. This is because the mentioned properties do not depend on the dimension of the links being chained, but only on dimensions that are actually traversed. However, traversing a chain of d-dim links does not always imply that dimension d is traversed. Consider three chains of d, d', and d''-dim links that cross each other consecutively. E.g., the chain $C_{S_d} = \{\dots, \ell_{j_L}^d, \dots, \ell_{j_R}^d, (L_{d''}^0 = \ell_{j_R+1}^{d''}, L_{d''}^1), \dots\}$ that is crossed by the chain $C_{S_{d'}} = \{\dots, \ell_{j_L-1}^{d'}, (L_d^0 = \ell_{j_L}^d, L_d^1), \dots\}$ at the link $\ell_{i_{j_L}}^d$. Also, C_{S_d} crosses the chain $C_{S_{d''}} = \{\dots, \ell_{i_{j_R+1}}^{d''}, \dots\}$ at the link $\ell_{i_{j_R+1}}^{d''}$. We examine whether dimension d is traversed by comparing the dth bit of the last link before the first cross to C_{S_d}, i.e. $\ell_{i_{j_L-1}}^{d'} = (a_0, b_0)$, to the dth bit of the first link after the second cross (by C_{S_d}), i.e. $\ell_{i_{j_R+1}}^{d''} = (a_1, b_1)$. Dimension d is traversed if and only if the two bits hold different values. That is, $(a_0 \wedge a_1) \wedge (2^{d-1})_2 = 0$.

A backup path $BP(\ell_i^d) = \{(L_{d+1}^0, L_{d+1}^1), \dots, N^*(\ell_i^d)\}$ can be represented as a chain $C(\ell_i^d) := \{\ell_i^d, \{L_{d+1}^0, L_{d+1}^1\}, N^{(1)}(\ell_i^d), \dots, N^{(*)}(\ell_i^d)\}$. By Definition 3.3.2, there is a dependency from ℓ_i^d to every other link $\ell_{i'}^{d'} \in BP(\ell_i^d)$. Let $MC(\ell_i^d, \ell_{i'}^{d'}) \subseteq C(\ell_i^d) \cup \{\ell_{i'}^{d'}\}$

denote the mixed chain up to and including $\ell_{i'}^{d'}$. Consider the set of backup paths of some subset of links $\{\ell_{i_0}^{d_0}, \ell_{i_1}^{d_1}, \dots, \ell_{i_{x-1}}^{d_{x-1}}\}$ that induce a cycle of dependencies. Each dependency corresponds to a mixed chain, concatenating them sequentially, yields the closed mixed chain $\mathcal{MC} := MC(\ell_{i_0}^{d_0}, \ell_{i_1}^{d_1}) \cup MC(\ell_{i_1}^{d_1}, \ell_{i_2}^{d_2}) \cup \dots \cup MC(\ell_{i_{x-1}}^{d_{x-1}}, \ell_{i_0}^{d_0})$. Recall that in $BP(.)$, pairs connecting consecutive same-dimension links are traversed in the ascending order of dimensions. Therefore, the sequence of dimensions traversed by $\mathcal{MC}$ is specified by $(\tilde{d}_i)_{i=0}$, where $\tilde{d}_0 = 0$, and either $\tilde{d}_{j+1} = \tilde{d}_j$ or $\tilde{d}_{j+1} = \tilde{d}_j + 1 \pmod k$. From now on, we assume only the closed chains restricted to the sequence of dimensions $\tilde{d}_i$.

Correctness

In the following, we address the correctness of our backup path scheme, i.e., resilience to up to $k-1$ link failures in k-dimensional hypercubes. To this end, we need one additional result:

Claim 3.3.6. *In any backup path $p := BP(\ell_i^d)$ at most one pair of links is traversed in uphill direction.*

Proof. If p does not detour any link then the only pair of links, i.e. $(L_{d+1}^0(\ell_i^d)), L_{d+1}^1(\ell_i^d)))$, is traversed either in uphill or downhill direction, which trivially satisfies the claim. If p detours some link ℓ_j^d, then $j < i$ (by construction). By Property 2, the pair of links preceding ℓ_j^d is traversed in downhill direction. Since p does not detour the last d-dim link, only the last pair (preceding the last link) is possibly traversed in uphill direction. □

We can now prove our main result:

Theorem 3.3.7. *The scheme $BP(.)$ listed in Term (1) is $(k-1)$-resilient.*

Proof. In order to show that the scheme is $(k-1)$-resilient, we argue that any cycle of dependencies consists of at least k links. We first show that for every $d \in [k]$, any cycle of dependencies over d-dim links is of length at least k. The backup path of every link ℓ_i^d uses only one d-dim link $\ell_{i'}^d, i' = i + 1 \pmod R$. Hence, the set of d-dim links are dependent sequentially. Therefore, having $R = \lceil \log k \rceil$ is sufficient to ensure any cycle of dependencies induced by d-dim links is of length $2^R \geq k$.

It remains to analyze the dependency cycles that consist of links in multiple dimensions. By Definition 3.3.5 and the construction of the $\mathcal{MC}$, such cycles correspond to mixed chains in the k-cube, each having the following properties:

1. Due to the non-descending sequence $\tilde{d}_i$ and by Property 1, $\mathcal{MC}$ traverses the sequence of dimensions $1, \dots k$ an even number of times, therefore there are at least $2k$ traversals.

2. By Property 3, at least k of the traversals are in uphill direction.

3. By Property 3.3.6, a backup path takes at most one uphill. Meaning, each dependency contributes at most one uphill traversal to the mixed chain.

Combining (1), (2), and (3), implies that there must be at least k dependencies in the assumed cycle of dependencies, which concludes our claim. □

3.3.7 Conclusion and Future Work

This section studied the design of algorithms for local fast failover in Segment Routing networks, subject to multiple link failures. Our main result is a maximally robust, $(k-1)$-resilient algorithm for k-dimensional hypercubes, which can be computed efficiently.

We see our work as a first step and believe that it opens several promising directions for future research. On the algorithmic side, it would be interesting to extend the study to algorithms for other graph classes, also providing a minimal number of segments or requiring a minimal number of forwarding rules. On the practical side, given that segment routing is ready to be deployed in IPv6 environments, it would be interesting to study experimental evaluations, which can in turn also refine our model. In the following, we provide some first directions.

3.3.8 Resilient Segment Routing on General Graphs

It will be interesting to study the complexity of fast rerouting on general graphs, and develop (approximation) algorithms accordingly. We conjecture that computing backup path schemes with maximal resiliency is NP-hard on general graphs. In non-polynomial time, a Mixed Integer Program (MIP) formulation can provide an optimal solution for general graphs. The following MIP considers the problem of generating a small number of required segments for the backup paths, and if the desired resiliency cannot be met, at least maximizes the number of protected links. We hope that our MIP formulation can aid the community in developing further backup path schemes, e.g., by using it as a baseline comparison to evaluate the quality of polynomial runtime algorithms for different graph classes beyond the hypercube.

More specifically, the MIP presented next will compute an f-resilient backup path allocation that is optimal in the number of protected links. For completeness purposes, we consider directed graphs $G = (V, E)$. As our MIP is also concerned with the number of labels for each backup path, we provide some additional preliminaries relevant to practical implementations. A backup path in general can be subdivided into path segments, each being a shortest path between its endpoints: such a path segment will only need one label on the stack, when the nodes employ shortest path routing. However, when the network utilizes link weights, some backup paths cannot be represented by node labels [85]: e.g., if a link on the backup path has infinite weight, while all other links have unit weight. For these corner cases, we need to allow single links as items on the label stack, which we denote as *tunnel* links: In the worst case, the whole backup path contains only tunnel links. Should a tunnel link physically fail, the corresponding label will be popped to prevent stuck packets (a failed link cannot be traversed), and the respective backup path will be traversed.

$$\textbf{Maximize} \sum_{\ell \in E} \mathcal{I}_\ell \tag{2}$$

$$\mathcal{SP}^z_\ell = \begin{cases} 1 & \ell \in SP(u,z) \\ 0 & else \end{cases} \qquad \forall \ell = (u,v) \in E, z \in V \tag{3}$$

$$\mathcal{D}_{\ell\ell'}, \mathcal{X}^v_{\ell\ell'}, \mathcal{T}_{\ell\ell'}, \mathcal{W}^v_\ell, \mathcal{I}_\ell, \in \{0,1\} \qquad \forall \ell, \ell' \in E, \forall v \in V \tag{4}$$

$$\mathcal{D}_{\ell\ell} = 0 \qquad \forall \ell \in E \tag{5}$$

$$\sum_{\ell_2=(v,*)} \mathcal{D}_{\ell_1\ell_2} - \sum_{\ell_2=(*,v)} \mathcal{D}_{\ell_1\ell_2} = \begin{cases} \mathcal{I}_{\ell_1} & v = s \\ -\mathcal{I}_{\ell_1} & v = t \\ 0 & \text{else} \end{cases} \qquad \forall \ell_1 = (s,t) \in E, v \in V \tag{6}$$

$$\mathcal{X}^v_{\ell_1\ell_2} \leq \mathcal{SP}^v_{\ell_2}, \sum_{\ell \in E, \ell \ni v} \mathcal{D}_{\ell_1\ell} \qquad \forall \ell_1, \ell_2 \in E, v \in V \tag{7}$$

$$\mathcal{D}_{\ell_1\ell_2} \leq \mathcal{SP}^t_{\ell_2} + \mathcal{T}_{\ell_1\ell_2} + \sum_{v \in V} \mathcal{X}^v_{\ell_1\ell_2} \qquad \forall \ell_1 = (s,t), \ell_2 \in E \tag{8}$$

$$d_{\ell_1\ell_2} \geq 0, d_{\ell_1\ell_1} = 0 \qquad \forall \ell_1, \ell_2 \in E \tag{9}$$

$$d_{\ell_1\ell_3} \leq d_{\ell_1\ell_2} + 1 + (1 - \mathcal{D}_{\ell_2\ell_3}) \times \infty \qquad \forall \ell_1, \ell_2, \ell_3 \in E \tag{10}$$

$$d_{\ell_1\ell_2} + d_{\ell_2\ell_1} \geq f + 1 \qquad \forall \ell_1, \ell_2 \in E \tag{11}$$

$$\mathcal{W}^v_{\ell_1} \geq \mathcal{X}^v_{\ell_1\ell_2} \qquad \forall \ell_1\ell_2 \in E, v \in V \tag{12}$$

$$\sum_{v \in V} \mathcal{W}^v_\ell + \sum_{\ell' \in E} \mathcal{T}_{\ell\ell'} \leq \text{LABELS} \qquad \forall \ell \in E \tag{13}$$

Armed with the above preliminaries, we can now provide a general overview of the MIP. Let $SP(u,z)$ be the shortest path between u and z.[8] For every link $\ell = (s,t)$, we pre-compute constants $\mathcal{SP}^z_\ell$, each indicating whether the shortest path from s to z includes ℓ or not. With respect to the logical flow of the formulation, the MIP first computes a backup path $\mathcal{P}_\ell = \{\ell' \in E \mid \mathcal{D}_{\ell\ell'} = 1\}$ for every link $\ell \in E$. Then, for every link $\ell' \in \mathcal{P}_\ell$ whose shortest path to t does not take the link itself (i.e. $\mathcal{SP}^t_{\ell'} = 0$), the MIP either finds an intermediate node v such that $\mathcal{SP}^v_{\ell'} = 1$, or flags the link as a tunnel link (with $\mathcal{T}_{\ell_1\ell_2}$). As a result, every link of $\mathcal{P}_\ell$ either is a tunnel link or is on the shortest path to a next intermediate node, if not t (i.e. on a segment). This is imposed by the set of constraints (7) and (8). With constraints (9) to (11), we ensure an f-resilient backup path selection. Constraint (11) forbids any cyclic dependency of length $\leq f$. At the end, the MIP restricts the number of segments to the constant LABELS.

Next, we explain each set of constraints and variables more technically.

[8]Should there be multiple options for shortest paths, we pick them in such a way that each subpath of a shortest path is again a shortest path.

- (2): maximizing the number of protected links. The failure of any subset of up to f protected links can be tolerated.
- (3): are the pre-computed shortest path trees for all nodes.
- (4): each variable $\mathcal{D}_{\ell\ell'}$ is set to 1 if ℓ' is designated to the backup path of ℓ ($\mathcal{P}_\ell$), otherwise remains 0. Each variable $\mathcal{X}^v_{\ell\ell'}$ indicates whether 1) the node v is a waypoint on $\mathcal{P}_\ell$ and 2) ℓ' is on the shortest path from the tail of ℓ' to v, hence on the backup path. Similarly, $\mathcal{T}_{\ell\ell'}$ indicates whether ℓ' is a tunnel link on $\mathcal{P}_\ell$. Variables $\mathcal{W}^v_\ell$ is set to 1 when some node v is used as a waypoint for $\mathcal{P}_\ell$. Each variable $\mathcal{I}_\ell$ indicated whether ℓ is protected.
- (6): these constraints enforce the links specified by $\mathcal{D}_{\ell_1 *}$ to form a simple path connecting the endpoints of ℓ_1, not using ℓ_1 (due to (5)).
- (7), (8): a link $\ell_2 = (x, y)$ is allowed to be on the the backup path $\mathcal{P}_{\ell_1}$, $\ell_1 = (s, t)$ only if

  1. the link ℓ_2 is on the shortest path $SP(x, t)$ i.e. $SP^t_{\ell_2} = 1$;
  2. else, a node $v \in \mathcal{P}_{\ell_1}$ exists s.t. $SP(x, v)$ begins with ℓ_2 (when $\mathcal{X}^v_{\ell_1\ell_2} = 1$ in (7)),
  3. else, the variable $\mathcal{T}_{\ell_1\ell_2}$ is set to 1, which enforces the link ℓ_2 on $\mathcal{P}_{\ell_1}$ as a tunnel link.

  Therefore at least one of the cases must apply to the pair ℓ_1, ℓ_2 in order to have $\mathcal{D}_{\ell_1\ell_2} = 1$ feasible. Cases 2 and 3 correspond to adding new segments. Note that the case 3 can trivially hold for any link which would result in unrestricted number of segments. But latter constraints avoid this in favour of having fewer segments.
- (9),(10),(11): here we formulate the all-pairs shortest path sub-problem on the dependency graph induced by $\mathcal{D}_{**}$. Given a feasible assignment, the value of each d_{xy} is at most the length of the shortest path from x to y. The length of the shortest cycle of dependencies through each dependency arc (ℓ_1, ℓ_2) is constrained by (11).
- (12),(13): the flag $\mathcal{W}^v_\ell$ is set to 1 whenever the node $v \in \mathcal{P}_\ell$ is used as a waypoint for some ℓ' on $\mathcal{P}_\ell$. We restrict the total number of labels (thus, the number of segments) using the constant LABELS.

Chapter 4

Online Balanced Partitioning

Distributed applications generate a significant amount of network traffic in datacenters. By collocating nodes (e.g., virtual machines) that communicate frequently so that they reside on the same clusters (e.g., server or rack), we can reduce the network load and improve application performance. However, the communication pattern of different applications is often unknown a priori and may change over time; hence it needs to be learned online. This paper revisits the online balanced partitioning problem that asks for an algorithm that strikes a trade-off between the benefits of collocation (i.e., reduced network traffic) and its costs (i.e., migrations). Our first contribution is a significantly improved deterministic lower bound of $\Omega(k \cdot \ell)$ on the competitive ratio, where ℓ is the number of clusters and k is the cluster size. The bound holds even for scenarios where the communication pattern can be perfectly partitioned so that all communications are internal to the clusters. We match this result with an asymptotically tight upper bound of $O(k \cdot \ell)$ for this scenario. For $k = 3$, we contribute an asymptotically tight upper bound of $\Theta(\ell)$ for the case where the communication pattern can change arbitrarily over time. We improve the result for $k = 2$ by providing a strictly 6-competitive upper bound.

4.1 Introduction

Data-centric applications, from distributed machine learning to distributed databases, generate a significant amount of network traffic in data centers [160, 174]. The performance of these distributed applications often depends on the performance of the underlying communication networks [47, 141].

The virtualization of resources in data centers introduces an intriguing opportunity to reduce network traffic and improve performance. In particular, it becomes possible to adaptively *migrate* frequently communicating virtual machines (or containers) closer to each other, in a demand-aware manner. Such adaptive migrations, however, come at a cost (e.g., resource and time overhead) and introduce a tradeoff.

This paper studies the algorithmic problem underlying such demand-aware optimizations, aiming to strike a balance between the benefits of migrations (e.g., reduced network load) and their costs. This is particularly challenging in a setting where the traffic can be bursty and change over time, in a hard to predict manner, as it is often the case in practice [21]. We are in the realm of *online algorithms and competitive analysis*, and ideally, the algorithm should perform closely to an optimal offline algorithm without requiring any information about future traffic demands.

4.1.1 Online Algorithms and Competitive Analysis

We measure the quality of the proposed solutions with competitive analysis [33], which suits well the networking problems that are online by their nature. The sequence of requests σ is revealed one-by-one, in an online fashion. Upon seeing a request, the algorithm must serve it without the knowledge of future requests.

We measure the performance of an online algorithm by comparing to the performance of an optimal offline algorithm. For a given sequence of requests σ, let $\text{ALG}(\sigma)$,be the cost incurred by a deterministic online algorithm ALG, and let $\text{OPT}(\sigma)$ be the cost incurred by an optimal offline algorithm OPT. In contrast to ALG that learns the requests one-by-one as it serves them, OPT has complete knowledge of the entire request sequence σ *ahead of time*. The goal is to design online algorithms that provide worst-case guarantees. In particular, ALG is said to be *α-competitive* if there is a constant β, such that for any input sequence σ it holds that

$$\text{ALG}(\sigma) \leq \alpha \cdot \text{OPT}(\sigma) + \beta.$$

Note that β cannot depend on input σ but can depend on other parameters of the problem, such as the number of nodes. The minimum α for which ALG is α-competitive is called the *competitive ratio* of ALG. We say that ALG is *strictly α-competitive* if additionally $\beta = 0$.

4.1.2 Model

The problem known as *online balanced graph repartitioning* was introduced by Avin et al. [19] at DISC 2016. A special variant of the general problem, called the *learning* model, was later introduced by Henzinger et al. [106] at SIGMETRICS 2019. In this paper, we obtain results for both models. We recall their formulation below.

The General Partitioning Problem We are given a set of n nodes, initially arbitrarily partitioned into $\ell \in \mathbb{N}^+$ clusters each of capacity $k = n/\ell$ nodes. The nodes interact in an online manner: we are given a sequence of pairwise communication requests $\sigma = (u_1, v_1), (u_2, v_2), (u_3, v_3), \ldots$, where a pair (u_t, v_t) indicates that nodes u_t and v_t exchange a fixed amount of data at time t. We sometimes refer to the nodes as virtual machines or processes, and we refer to the clusters as servers.

The cost of serving a request (u, v) depends on the relative positions of u and v: if they reside in the same cluster, the request costs 0, and it costs 1 otherwise. Before serving a request, an online algorithm may perform a *repartition* of nodes. That is, it may migrate any number of nodes to different clusters while ensuring the number of nodes in each cluster does not exceed k by more than ϵk for constant $\epsilon \geq 0$. We refer to the extra capacity as *resource augmentation.* For the most of this paper, we consider the problem without any augmentation, i.e., $\epsilon = 0$. The cost of migrating a single node is $\alpha \in \mathbb{N}^+$. The goal is to minimize the total cost of repartitions and to serve the requests. We use the terms "partition", "partitioning" and "configuration" interchangeably.

The Learning Variant Consider a variant of the general partitioning problem, where each pair of nodes either never communicates or they communicate indefinitely. Once a communication request (u, v) arrives, any algorithm must keep u and v collocated for the rest of the input sequence. We assume that requests of σ constitute a communication graph that admits a *perfect partitioning*: a partitioning that assigns exactly k nodes to each cluster, and no inter-cluster request ever occurs in this partitioning. Moreover, an optimal offline algorithm moves to this partitioning before serving the sequence, and it stays there permanently. The goal of the algorithm is to *learn* the communication graph (as it is revealed one edge at a time) while serving requests without performing too many node migrations. In the learning model, any two communicating nodes must be collocated, and only the migration cost is relevant; for simplicity, we may scale it down to $\alpha = 1$ (our bounds hold for any $\alpha > 1$ as well).

4.1.3 Related Work

Closest to our work are those of Avin et al. [19] at DISC 2016 (on the general partitioning model) and Henzinger et al. (on the learning model) [106] at SIGMETRICS 2019 and SODA 2021 [105]. Recently, a polynomial-time online algorithm achieving the same competitive ratio as in [19] has been proposed by Forner et al. [81]. However, the focus of these papers is primarily on models with resource augmentation: the online algorithm can use slightly larger clusters than the offline algorithm. Avin et al. showed that their lower bound $\Omega(k)$ holds even in a scenario with significant capacity augmentation, and they provided an algorithm with the competitive ratio $O(k \log k)$ using the $(2 + \epsilon)$-augmented cluster capacity. Their ratio is independent of ℓ, which is impossible without significant resource augmentation.

In contrast, we study the non-augmented setting, where the nodes need to be perfectly balanced among the clusters. This assumption is more realistic, as it utilizes all the processing capacity of a data center instead of leaving some CPUs idle. This variant is significantly more challenging, as it is related to hard problems such as integer partitioning [11]. Not much is known about the setting without augmentation. For $k = 2$, Avin et al. [19] presented a 7-competitive algorithm with a substantial $(\Omega(\ell^2))$ additive constant. For $k > 3$, a $O(k^2 \cdot \ell^2)$-competitive (phased-based) algorithm was given by [19]. Later, a more sophisticated analysis by Bienkowski et al. [28] improved the ratio of the same algorithm to $O(2^{O(k)} \cdot \ell)$, which is significant due to its linear dependency on ℓ. The best known lower bound is $\Omega(k)$ [19]. Given our lower bound of $\Omega(k \cdot \ell)$ in this paper, the quest for closing the gap remains open.

The problem has also been studied in a weaker model where the adversary can only sample requests from a fixed distribution [20] over the edges of a ring communication graph.

The *static* offline version of the partitioning problem is known as the *ℓ-balanced graph partitioning problem*, where the entire communication graph is known in advance, and the task is to partition n nodes into ℓ clusters of capacity n/ℓ each, minimizing the number of inter-cluster edges, The problem is NP-complete, and cannot even be approximated within any finite factor unless P=NP [10]. The static variant where $\ell = 2$ corresponds to the minimum bisection problem, which is already NP-hard [89], and currently the best approximation ratio is $O(\log n)$ [14, 65, 66, 124, 157, 165].

The studied problem is further related to some classic online problems. In particular, it is related to online paging [1, 51, 70, 140], sometimes also referred to as online caching, where requests for data items (nodes) arrive over time and need to be served from a cache of finite capacity, and where the number of cache misses must be minimized. Classic problem variants usually boil down to finding a smart eviction strategy, such as Least Recently Used (LRU) [51]. In our setting, requests can be served remotely (i.e., without fetching the corresponding nodes to a single physical machine). In this light, our model is more reminiscent of caching models *with bypassing* [**EpImLN15**, 58, 108]. A major difference between these problems is that in the caching problems, each request involves a single element of the universe, while in our model, *both* endpoints of a communication request are subject to optimization. In this light, we can see our model as a “symmetric" version of online paging. The general problem additionally generalizes symmetric ski rental [115].

Graph partitioning and clustering problems are fundamental in computer science and arise in multiple contexts, see [6, 176].

4.1.4 Contributions

This paper presents several new results for the online graph partitioning problem without augmentation. For both the learning model and the general model, we obtain a lower bound of $\Omega(k \cdot \ell)$ on the competitive ratio of any online deterministic online algorithm (that also holds in the general partitioning model). This improves over the best known lower bound $\Omega(k)$ [19] that holds only in the general partitioning model. We complement this result with an asymptotically optimal, $O(k \cdot \ell)$-competitive algorithm for the learning model. We further adjust the lower bounds to show that the factor of $\Omega(\ell)$ is unavoidable even with significant augmentation.

For the general partitioning model and $k = 3$, we design an asymptotically optimal $O(\ell)$-competitive algorithm matching the best known upper bound of $O(2^{O(k)}\ell)$ given by Marcin et. al. [28]. The case of $k = 3$ exhibits properties which we exploit for an alternative and simpler analysis. The generalized lower bound $\Omega(k \cdot \ell)$

for the learning model holds also in the general model. We adjust the lower bound construction to show that the factor of $\Omega(\ell)$ is unavoidable with significant augmentation also in the general model. We further present a strictly 6-competitive algorithm for $k = 2$ that improves upon the previous 7-competitive algorithm with $O(\alpha\ell^2)$ additive constant. All algorithms in this paper have a strict competitive ratio (cf. Section 4.1.1), which improves over previous results with substantial additive in terms of $(\alpha \cdot k \cdot \ell)^2$. Table 4.1 provides an overview of our contributions compared to prior work.

Variant	**Lower bound**	**Upper bound**
Learning, $k \geq 3$	$\Omega(k\ell), \epsilon = 0$ (Thm. 4.3.1)	$O(k\ell), \epsilon = 0$ (Thm. 4.3.4)
Learning, $k \geq 3$	$\Omega(\ell \log k), \epsilon \leq \frac{1}{32}$ [105]	$O(\ell \log k), \epsilon \leq 1$ [105]
Learning, $k \geq 3$	$\Omega(\ell), \epsilon < 1/3$ (Thm. 4.3.2)	$O(\log k), \epsilon > 1$ [105]
General, $k = 3$	$\Omega(\ell), \epsilon = 0$ (Thm. 4.4.1)	$O(\ell), \epsilon = 0$ (Thm. 4.4.3)
General, $k = 2$	$3, \epsilon = 0$ [19]	$6, \epsilon = 0$ (Thm. 4.4.6)
General, $k > 3$	$\Omega(k\ell), \epsilon = 0$ (Thm. 4.4.1)	$O(2^{O(k)}\ell), \epsilon = 0$ [28]

Table 4.1: Overview of our contributions and known results on the deterministic online partitioning problem.

4.2 Preliminaries

We say that an assignment of nodes to clusters is a *partitioning* of nodes, and it represents the configuration of the algorithm. The reconfiguration or migration of nodes is a *repartitioning* operation. We say that nodes are *collocated* if they reside in the same cluster in the current partitioning. An algorithm serves a communication request between two nodes either *locally* at cost 0 if they are collocated, or *remotely* at cost 1 if they are located in different clusters. We refer to these two types of requests as *internal* and *external* requests, respectively.

We often refer to the graph structure of the request sequence. Precisely, we say that a set of nodes belong to a *communicating component* if there exists a path between them in the communication graph of requests issued so far. We say that a communicating component is a *singleton* if it contains exactly one node, and we refer to this node as an *isolated* node. For each node, we say that its *origin* is the cluster that the node belonged to in the initial partition. For any component of nodes C that were collocated in the initial partition, let $I(C)$ denote the cluster where all nodes of C resided initially.

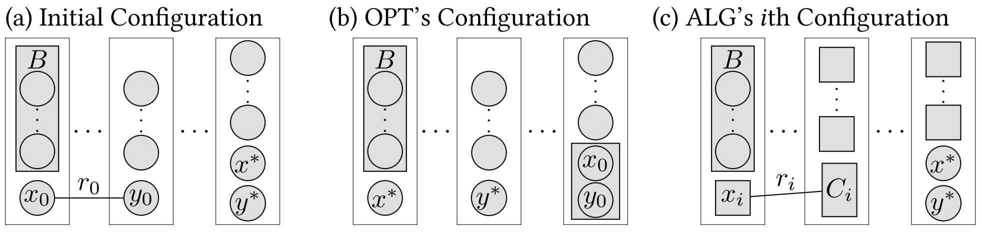


Figure 4.1: Large rectangles represent clusters. Nodes are shown in gray circles, and gray rectangles represent components. Both ALG and OPT start in the configuration (a). OPT performs only two swaps and ends up at the configuration (b). At the beginning of the ith iteration, ALG is in the configuration (c) before evicting the ith pivot node x_i from the cluster of component B.

4.3 The Learning Model

We consider the learning variant of online balanced graph partitioning problem. First, we show a surprisingly high lower bound of $\Omega(k \cdot \ell)$ against deterministic algorithms for $k \geq 3$. The lower bound holds also in the general partitioning model (see Theorem 4.4.1 for details). Second, we provide a deterministic algorithm that asymptotically matches this lower bound for the learning model.

4.3.1 Lower Bound

Overview of the construction At each time step, we issue a new request, depending on the configuration of the online algorithm. First, we issue requests between $k-1$ nodes of some arbitrarily chosen cluster, and we refer to this communicating component as B. In any partitioning, the communicating component B is collocated with exactly one isolated node, called the *pivot*. Second, we issue a request between the pivot and an arbitrarily chosen node from a different cluster. Any algorithm must collocate these nodes (recall that we consider a learning variant), and it must place them in a different cluster than B's (otherwise, its capacity would be violated). Note that after collocating them, another isolated node must the take place of the pivot. Third, we issue the request between the new pivot and an arbitrarily chosen node from the same cluster of origin as the pivot. We repeat the last step of the construction $\Theta(k \cdot \ell)$ times (exact condition to be determined), using the node isolated node collocated with B as a new pivot. The algorithm pays 1 per each such request.

We claim that this sequence is cheap to serve offline. Roughly, if an offline algorithm would reside in the initial configuration, only the requests between x_0 and y_0 would be external, and all the subsequent requests would be free. Consider an offline strategy that collocates x_0, y_0 by swapping them with some initially collocated nodes x^*, y^* that did not participate in any request. To make sure that such a pair always exists, we stop repeating the requests concerning the pivot while there are still two isolated nodes collocated on some server. We illustrate the construction in Figure 4.1.

Theorem 4.3.1. *Any deterministic online algorithm for the learning model of online balanced graph partitioning and $k \geq 3$ has the competitive ratio of at least $(k-2)(\ell-1)/2-2$.*

Proof. Fix any online algorithm ALG. Initially, all nodes are isolated, i.e., each node is in a singleton communicating component. We issue requests one-by-one, in reaction to ALG's choices.

1. **Component B.** We issue requests among $k-1$ nodes in an arbitrary cluster, and we refer to these nodes as a communicating component B. In any feasible partition, a single isolated node must be collocated with B (each cluster hosts exactly k nodes). We refer to the isolated node collocated with B at any time as the *pivot* node. Let x_0 denote the first pivot node.

2. **Request between nodes originating from different clusters.** We issue a request between x_0 and an arbitrarily chosen isolated node y_0. This leads to the eviction of x_0 (otherwise, the algorithm incurs an arbitrarily large cost, while the optimal strategy is to collocate all communicating pairs). Hence, ALG must collocate this pair in a different cluster (cannot collocate it with B). In order to maintain a feasible partitioning of nodes after collocating $\{x_0, y_0\}$, ALG must replace x_0 with another isolated node, the new pivot node.

3. **Requests between nodes originating from the same clusters.** We continue to issue requests between the current pivot node and any node with the same origin as the pivot. Consider the i-th such request, and the isolated node x_i, collocated with B. Precisely, we issue a request between x_i and some node in C_i, where C_i is the largest communicating component s.t. $I(C_i) = I(x_i), C_i \neq \{x_0, y_0\}$. Then, ALG must collocate the communicating component $\{x_i\} \cup C_i$ in one cluster, and again, the algorithm replaces x_i with some isolated node x_{i+1}. We terminate the process once the number of remaining isolated nodes is smaller than $\ell+3$. At each step i, the number of isolated nodes decreases either by one, or it decreases by two if C_i is a singleton. Therefore, once the process terminates, at least $\ell+1$ nodes remain isolated.

To assure the correctness of this input sequence, we claim that the communicating components admit a feasible partition. Once we terminate, there are at least $\ell+1$ isolated nodes left (the number of isolated nodes decreases at most by 2 at each step). Therefore, two isolated nodes x^* and y^* exist with the same cluster of origin, $I(\{x^*\}) = I(\{y^*\})$. Consider a partition P^*, obtained from the initial partition after swapping x_0 and y_0 with x^* and y^*, respectively. In P^*, the pair $\{x_0, y_0\}$ resides in the cluster $I(\{x^*, y^*\})$. After the first request $\{x_0, y_0\}$, the requests are issued only between nodes originating from the same cluster, and all of these are collocated in

P^*. This implies that no request is external in P^*, and we conclude that it is a feasible partition.

We bound the cost of ALG on the produced request sequence. At each request issued at step (3) of our construction, some communicating component grows, and ALG performs at least one swap. Let $\mathcal{S}$ be the set of all communicating components created by issuing requests at step (3) of the construction. Each component $S \in \mathcal{S}$ grew exactly $|S|-1$ times, each time joining an isolated node, and hence the number of nodes ALG swaps is at least

$$\text{ALG} \geq \sum_{S \in \mathcal{S}} (|S| - 1) = |\bigcup \mathcal{S}| - |\mathcal{S}|.$$

In total, $\bigcup \mathcal{S}$ contains all the $k \cdot \ell$ nodes excluding $k-1$ nodes of B, the 2 nodes in $\{x_0, y_0\}$ and at most $\ell + 2$ singletons, which amounts to at least $k \cdot \ell - k - \ell - 3$ nodes. The set $\mathcal{S}$ consists of at most $\ell - 1$ components, one per possible cluster of origin excluding B's cluster of origin. Hence, the total number of swaps performed by ALG is

$$\text{ALG} \geq |\bigcup \mathcal{S}| - |\mathcal{S}| = k \cdot \ell - k - 2\ell - 2 = (k-2)(\ell - 1) - 4.$$

Finally, we bound offline algorithm's cost for the constructed sequence of requests. Consider an offline algorithm OPT that moves to P^* (described earlier in this proof) by performing only two node swaps. As argued earlier, no communicating component is split in P^* and OPT pays only for the two swaps. We combine the above arguments to conclude that the competitive ratio is bounded by $\text{ALG}/\text{OPT} \geq (k-2)(\ell-1)/2 - 2$. □

We note that the lower bound requires $k \geq 3$. In contrast, for $k = 2$ the learning problem is trivial: immediate collocation of communicating pairs is 1-competitive. In contrast, the general partitioning problem for $k = 2$ is non-trivial (see Section 4.4.4), a lower bound of 3 is known [19], and we provide a 6-competitive algorithm, see Section 4.4.4.

Later in Section 4.4.1, we elaborate on how to transform this construction to a lower bound for the general partitioning problem.

4.3.2 Lower Bound for Algorithms with Resource Augmentation

The majority of work on the online balanced partitioning problem so far [19, 105, 106] focuses on the scenario with resource augmentation, where the cluster capacity of an online algorithm is larger than that of the optimal offline algorithm to which we compare the performance. We say that an online algorithm uses augmentation $\epsilon > 1$ if each of its clusters has the capacity of $\epsilon \cdot k$ nodes. The number of all nodes remains $k \cdot \ell$.

By using a variant of the lower bound construction from Theorem 4.3.1, we show a lower bound of $\Omega(\ell)$ that holds even with significant resource augmentation.

Theorem 4.3.2. *With resource augmentation strictly smaller than $k/3$ (i.e., $\epsilon < 1/3$), the competitive ratio of any deterministic online algorithm for the learning model of online balanced graph partitioning is in $\Omega(\ell)$.*

Proof. Fix k divisible by 3, and construct 3 communication components of size $k/3$ in each cluster. Consider any deterministic online algorithm with resource augmentation $\epsilon < 4/3$. Note that no more than 3 such communication components fit in one cluster. Then, apply the construction from the lower bound for $k = 3$ (Theorem 4.3.1), treating these communication components as individual nodes. The cost of any algorithm (including the optimal offline algorithm) scales up by $k/3$ due to the increased cost of moving entire components instead of individual nodes, and we conclude that the lower bound $\Omega(\ell)$ holds. □

4.3.3 Upper Bound

We present an asymptotically optimal algorithm PPL (*Perfect Partition Learner*) for the learning model. The algorithm is a modification of the algorithm DET from [19]. The difference is in the choice of partition after a component merge. In DET, the choice of the partition was arbitrary, whereas our algorithm chooses an arbitrary partition closest to the initial partition that keeps all communicating nodes collocated. The algorithm PPL is listed in Algorithm 4.1.

To maintain the feasibility of the solution, the algorithm maintains *components* of communicating nodes. Initially, all nodes are in their own component, and upon a request between two nodes from different components, we *merge* the components. We maintain an invariant that the nodes of each communicating component are collocated. We say that a partition that collocates all nodes of all communicating components is a *communicating component respecting* partition.

Perfect Partition Learner On each inter-cluster request $\{u, v\}$, PPL creates new components by merging the two components that contain nodes u and v. In order to collocate nodes of the new component, PPL moves to a communication component-respecting partition that minimizes the distance to the initial partition P_I. We measure the distance between two configurations in the number of swaps to transform one configuration to the other. The distance to the initial partition is equivalent to the number of nodes that migrated from their cluster origin. Algorithm 4.1 describes the scheme of the algorithm.

Analysis Fix a request sequence σ, the initial partition $P_I := \{I_1, \dots, I_\ell\}$ and an optimal offline strategy OPT with the final partition P_{OPT}. For each partition $P = \{C_1, \dots, C_\ell\}$ we define its *distance* from the initial partition as the number of nodes in P that do not reside in their initial cluster. Observe that at least $\Delta(P)$ node migrations are required in order to reach the partition P from P_I, and thus $\text{OPT}(\sigma) \geq \Delta(P_{\text{OPT}})$.

PPL replaces the current partition P with a perfect partition closest to P_I, thus it never moves to a partition that is more than $\Delta^* := \Delta(P_{\text{OPT}})$ migrations away

Algorithm 4.1: Perfect Partition Learner (PPL)

```
input  : Parameters $k, \ell > 0$, request sequence $\sigma$, initial partitioning $P_I$
output: Perfect partitioning $P_F$
1 for each node $v$ do
2     create a singleton component $C_v = \{v\}$ and add it to $C$.
3 for each request $\sigma_t = \{u, v\}, 1 \leq t \leq N$ do
4     Let $C_1 \ni u$ and $C_2 \ni v$ be the components containing $u$ and $v$, respectively.
5     if $C_1 \neq C_2$ then
6         Merge $C_1$ and $C_2$ into one component $C'$ and $C = (C \setminus \{C_1, C_2\}) \cup \{C'\}$.
7         if $C_1$ and $C_2$ are not collocated then
8             Move to a partitioning closest to $P_I$ and respecting all components in
              $C$.
```

from P_I. Consequently, PPL never moves to a partition beyond the distance Δ^*. We use this property to bound the cost of each repartitioning of PPL.

Property 4. *Let P be any partitioning chosen by PPL at any time. Then, $\Delta(P) \leq \Delta^*$.*

Lemma 4.3.3. *The cost of each repartitioning by PPL is at most $2 \cdot OPT(\sigma)$, where $OPT(\sigma)$ is the cost of an optimal offline algorithm for the request sequence σ.*

Proof. Let P_i denote the partition of PPL immediately after serving σ_i, the request that arrives at time t. Consider the repartitioning that transforms P_{t-1} to P_t upon the request σ_t. Let $M \subseteq V$ denote the set of nodes that migrate at t. Let M^- and M^+ denote the subset of nodes that, respectively, enter or leave their initial cluster during the repartitioning. In total, M^+ and M^- account for all migrations, $M = M^+ \cup M^-$.

Since at least $|M^-|$ nodes are not in their initial cluster before the repartitioning (i.e., in P_{t-1}), the distance from P_I before the repartitioning is $\Delta(P_{t-1}) \geq |M^-|$. Analogously, the distance after the repartitioning is $\Delta(P_t) \geq |M^+|$. Thus, $|M| \leq \Delta(P_{t-1}) + \Delta(P_t)$. By Property 4, $\Delta(P_{t-1}) \leq \Delta^*$ and $\Delta(P_t) \leq \Delta^*$. Since $\Delta^* \leq \text{OPT}(\sigma)$, we conclude that the total cost of the algorithm is $|M| \leq 2 \cdot \text{OPT}(\sigma)$. □

Theorem 4.3.4. *PPL is $(2(k-1) \cdot \ell)$-competitive.*

Proof. Let $P_F := \{F_1, \dots, F_\ell\}$ be the partition of PPL after serving the sequence σ. At each request, the algorithm enumerates all communicating component-respecting ℓ-way partitions of components that are in the same (closest) distance to P_I. That is, once it reaches a partition P at distance $\Delta^* = \Delta(P)$, it does not move to a partition P' where $\Delta(P') > \Delta^*$ before it enumerates all partitions at distance Δ^*. Hence, P_F is at distance at most $\Delta^* = \text{OPT}(\sigma)$ from the initial partition.

We claim that PPL performs at most $(k-1) \cdot \ell$ repartitions while serving σ. Each component begins as a singleton, and with each request, the size of some component increases by one. Consequently, the number of repartitions of PPL is bounded by the number of times the components grow. Consider any cluster $F_i \in P_F$. Each cluster has the capacity k, thus the total number of times a component in F_i grows is at most $\sum_{C \in F_i}(|C| - 1) \leq k - 1$. Summing this bound over all ℓ clusters gives us at most $(k - 1) \cdot \ell$ repartitions.

By Lemma 4.3.3, each repartitioning costs at most $2 \cdot \mathrm{OPT}(\sigma)$. The total cost of PPL is thus at most $2 \cdot (k - 1) \cdot \ell \cdot \mathrm{OPT}(\sigma)$, which implies the claimed competitive ratio. □

By Theorem 4.3.1, the lower bound for the competitive ratio of any deterministic algorithm is $\Omega(k \cdot \ell)$, and we conclude that PPL is asymptotically optimal.

Note on running time Since the component sizes are in $O(n)$, computing a component-respecting partition for $\ell = 2$ is feasible in polynomial time using dynamic programming [20], but is strongly NP-hard for $\ell > 2$ [90]. However, we assume unlimited computational power and focus on competitiveness instead.

4.4 General Partitioning Model

In this section, we discuss the general online model where the request sequence can be arbitrary. First, in Section 4.4.1, we show a lower bound of $\Omega(k \cdot \ell)$ by generalizing the construction for the learning model from Section 4.3.1. Second, we generalize the lower bound for resource augmentation from Theorem 4.3.2. Third, in Section 4.4.3, we show an $O(k \cdot \ell)$-competitive algorithm for $k = 3$. Finally, in Section 4.4.4, we show a strictly 6-competitive algorithm for $k = 2$.

4.4.1 Lower Bound

We generalize the lower bound construction of for the learning model in two dimensions. Recall that in the learning model, algorithms by assumption collocate any pair as soon as they communicate. Hence, we cannot apply the lower bound construction of Theorem 4.3.1 directly since algorithms may resist collocating such nodes. For the first dimension of our generalization, we impose a collocation by defining *ground sets* of nodes. Nodes that are assigned to the same ground set communicate as long as they are separated. For the second dimension of our generalization, we iterate the construction in batches. Repeating batches ensures that the input sequence with the claimed ratio can be arbitrarily long, and the algorithm cannot have a small competitive ratio with a large additive. After each batch, we ensure that the online algorithm resides in the same configuration as the optimal offline solution. A single batch resembles the construction from Theorem 4.3.1, but in place of each request, we *reveal* a ground set: once the online algorithm splits a revealed set, we issue as many requests as it takes for it to collocate the split parts.

We start by revealing a ground set B of size $k-1$ in an arbitrary cluster. Then, we reveal a cross-origin ground set with the pivot (the node collocated with B). Then, we repeatedly reveal ground sets of the current pivot and a node originating from the same cluster as the pivot. We repeat the last step of the construction $\Theta(k \cdot \ell)$ times (exact condition to be determined), using the isolated node collocated with B as a new pivot. The algorithm swaps at least one node at each such step.

We claim that this sequence is cheap to serve offline. Roughly, if an offline algorithm would reside in the initial configuration, only the requests between x_0 and y_0 would be external, and all the subsequent requests would be free. Consider an offline strategy that collocates x_0, y_0 by swapping them with some initially collocated nodes x^*, y^* that do not participate in any request. To make sure that such a pair always exists, we stop repeating the requests concerning the pivot while there are still two isolated nodes collocated on some server.

Ground sets We construct our lower bound using *ground sets*. Instead of directly constructing the request sequence, we construct ground sets of nodes that start communicating if split by the online algorithm. This is possible since the algorithm is deterministic, and the adversary knows its configuration at any time. If the algorithm insists on keeping a ground set split, we continue to issue requests between non-collocated nodes of the ground set until the algorithm collocates them. Under such a request sequence, the algorithm must maintain a perfect partition of ground sets, as otherwise, it is not competitive. The ground sets are initially unknown to the algorithm, and the perfect partition is hidden from it. In contrast, the optimal offline algorithm knows the entire sequence in advance and may move to the perfect partition at the beginning.

Theorem 4.4.1. *Any deterministic online algorithm for the general model of online balanced graph partitioning has the competitive ratio at least* $(k-2)(\ell-1)/2-2$.

Proof. Fix any deterministic online algorithm ALG and any optimal offline algorithm OPT. We construct a sequence for the general model in *batches* of requests that can be repeated arbitrarily many times.

We construct the first batch by repeating the construction from Theorem 4.3.1, where in place of a request (u, v), we merge the ground sets of u and v. This ensures that the algorithm collocates all nodes from each communicating component. If the algorithm does not collocate the nodes, it is not competitive as it pays an arbitrarily large cost for split ground sets. By the construction from Theorem 4.3.1, there exists a partitioning that collocates all the ground sets, and therefore any competitive algorithm eventually moves to such configuration.

After the batch finishes, we force the algorithm to move into the partitioning that is identical to OPT's partitioning. Let $\{C_1, C_2, \dots, C_\ell\}$ be OPT's configuration at this point. We reveal additional ground sets C_i for $i \in [1, \ell]$ (i.e., containing all nodes that OPT has collocated). Note that these requests are free for OPT, and therefore OPT does not change its configuration. The batch ends once the algorithm moves to OPT's configuration.

Once the algorithm reaches OPT's configuration, we issue the next batch by repeating the construction. We may repeat issuing batches this way an arbitrary number of times. By applying similar reasoning to the proof of Theorem 4.3.1, in each batch the algorithm performs at least $(k \cdot \ell - k - 2\ell - 2)$ swaps, each for cost α. OPT performs at most two swaps in each batch. The competitive ratio from Theorem 4.3.1 holds for each batch separately, and therefore it holds for the entire sequence. □

4.4.2 Lower Bound for Algorithms with Resource Augmentation

We generalize the lower bound for the learning model (Theorem 4.3.2) to show that the factor of $\Omega(\ell)$ is unavoidable even with significant augmentation.

Theorem 4.4.2. *Any deterministic online algorithm with resource augmentation smaller than $4/3$ for the general model of online balanced graph partitioning is $\Omega(\ell)$-competitive.*

Proof. Fix k divisible by 3, and construct 3 ground sets of size $k/3$ in each cluster. Consider any deterministic online algorithm with resource augmentation $1+1/3-\epsilon$. Note that no more than 3 such ground sets fit in one cluster. Then, apply the construction from the lower bound (Theorem 4.4.1) for $k = 3$ using these communication components treating them as individual nodes. The cost of any algorithm (including the optimal offline algorithm) scales up by $k/3$ due to increased cost of moving entire ground sets instead of individual nodes, and we conclude that the lower bound $\Omega(\ell)$ holds. □

4.4.3 Optimal Algorithm for Clusters of Size 3

For the setting with $k = 3$, Avin et al. [19] obtained a $O(\ell^2)$-competitive algorithm. Their algorithm keeps track of external communication between nodes, and upon reaching a threshold α (the cost of migrating a node), we call the edge between them *saturated*. The algorithm keeps the invariant that the endpoint nodes of each saturated edge are collocated, and to this end, the algorithm tracks connected components consisting of nodes reachable via saturated edges. To collocate the nodes, the algorithm moves to an arbitrary partition where all nodes of all connected components are collocated. The algorithm operates in *phases*, and if no partition satisfying the invariant exists, it resets the counters for all pairs of nodes.

The algorithm ALG_3 presented in this section is a modified version of this algorithm. The difference is in the choice of partition after a component merge. Their algorithm chooses the partitioning arbitrarily, and our algoritm chooses the partition closest to the current partition (a repartition of minimum cost).

Our modification of the algorithm is straightforward, and our main contribution is an improved analysis of the algorithm. A straightforward analysis of our algorithm results in the bound $O(\ell^2)$, and we improve the analysis two aspects. First, we observe that the cost of each of $O(\ell)$ reconfigurations per phase is constant. Second, we deal with pairs of nodes that do not reach the threshold α (*unsaturated edges*). In the analysis of the algorithm from [19], these caused an additive $O(\alpha \cdot k^2 \cdot \ell^2)$.

In our analysis, we observe that either the number of unsaturated edges is small, or any algorithm pays for a significant fraction of them. We bound the cost of the latter by estimating the capabilities of the optimal offline algorithm to *prepare* for an incoming sequence of requests.

Saturated components We say that the pair (u, v) is *saturated* if the counter's value is α, and *unsaturated* otherwise (saturation of a pair leads to a merge action). We say that a partition that collocates all nodes of all saturated components is a *saturated component respecting* partition.

The algorithm ALG$_3$ For each pair of nodes $\{x, y\}$, ALG$_3$ maintains a counter $C_{\{x,y\}}$ and increments it on every external request between x and y. Initially, each node is isolated (belongs to its own component). Once $C_{\{x,y\}} = \alpha$, ALG$_3$ merges the components of u and v, and moves to the closest saturated component respecting partition. If no such partitioning exists, ALG$_3$ resets all components to singleton components, resets all counters to 0, and ends the phase.

Theorem 4.4.3. *ALG$_3$ is 60ℓ-competitive for $k = 3$.*

Before bounding the competitive ratio of ALG$_3$, we upper-bound the cost of a single repartition of ALG$_3$. In our analysis, we distinguish between three types of clusters: C_1, C_2 and C_3. In a cluster of type C_i, the size of the largest component contained in this cluster is i.

Lemma 4.4.4. *In a single repartition of ALG$_3$, it swaps at most two pairs of nodes.*

Proof. If no saturated component respecting partition exists after the merge of components, then ALG$_3$ resets all components, ends the phase, and performs no repartition. Thus it suffices to show that the merged component has a size at least 4 to conclude that ALG$_3$ incurs no migration cost.

Consider a request between u and v that triggered the repartition and let U and V be their respective clusters. The request triggered the repartition, hence it was external and $U \neq V$. We consider cases based on the types of clusters U and V.

1. If either U or V is of type C_1, then this cluster can fit the merged component, and the repartition is local within U and V, for the cost of at most 2 swaps.

2. If either U or V is of type C_3, a component of size 3 participates in a merge, and we have a component of size at least 4, and ALG$_3$ ends the phase with no repartition.

3. It remains to consider the case where both U and V are of type C_2. If (u, v) both belong to components of size 2, then the merged component has size 4, and ALG$_3$ incurs no cost. Otherwise, if one of u, v belongs to a component of size 2, then it suffices to swap components of size 1 between U and V. Finally, if u and v belong to components of size 1, then we must place them in

a cluster different from U and V. Note that if C_1-type cluster does not exist, then no saturated component respecting partitioning exists. Otherwise, ALG$_3$ performs two swaps — it swaps the nodes u and v with any two nodes of any cluster of type C_1.

In each case, we showed that a saturated component respecting partition is reachable in at most two swaps. □

Next, we observe that ALG$_3$ keeps saturated edges internal, and it increases counters only upon external communication, thus we upper-bound the ALG$_3$'s counter value for each unsaturated edge in any phase.

Observation 4.4.5. *The external request counter for each unsaturated edge has a value at most $\alpha - 1$.*

Now we are ready to bound the competitive ratio of ALG$_3$.

Proof of Theorem 4.4.3. Fix a completed phase, and consider the state of ALG$_3$'s counters at the end of it (before the reset). By σ we denote the input sequence that arrived during the phase. We consider the incomplete phase later in this proof.

In our analysis, we focus on the requests that were external to ALG$_3$ at the moment of their arrival; these are the only requests that incur a cost for ALG$_3$. We denote these external requests by σ_{cost}. We partition the sequence σ_{cost} into subsequences σ_I and σ_E. The sequence σ_I (inter-component requests) denotes the requests from σ_{cost} issued to pairs that belong to the same component of ALG$_3$ at the end of the phase. The sequence σ_E (extra-component requests) denotes the requests from σ_{cost} that do not appear in σ_I. Let ALG$_3(M)$ denote the cost of migrations performed by ALG$_3$ in this phase.

First, we bound the cost of ALG$_3$ in the phase. During the phase, ALG$_3$ performs at most 2ℓ component merge operations — exceeding this number would mean that a component of size 4 exists, and the phase would have ended already. We bound the cost of each repartition after a merge by Lemma 4.4.4, obtaining ALG$_3(M) \leq 8\alpha \cdot \ell$. We bound ALG$_3(\sigma_I)$ by summing the intra-component counters of each cluster at the end of the phase. The sum of intra-component counters in a cluster of type C_3 is at most $3\alpha - 1$: two pairs of nodes from the component are saturated and its counter is α each, and the counter of the third, unsaturated pair is at most $\alpha - 1$ by Observation 4.4.5. The sum of counters inside C_1 is 0, and inside C_2 it is α. Summing over all ℓ clusters gives us ALG$_3(\sigma_I) \leq (3\alpha - 1) \cdot \ell \leq 3\alpha \cdot \ell$. Furthermore, ALG$_3$ paid for all requests from σ_E, and thus ALG$_3(\sigma_E) = |\sigma_E|$. In total, the cost of ALG$_3$ is at most ALG$_3(\sigma_I)$ + ALG$_3(\sigma_E)$ + ALG$_3(M) \leq 11\alpha \cdot \ell + |\sigma_E|$ during the phase.

Second, we lower-bound the cost of the optimal offline solution. To this end, we fix any optimal offline algorithm OPT. By OPT(σ_I) and OPT(σ_E) we denote the cost of OPT on requests from sequences σ_I and σ_E, respectively. Note that these costs are defined with respect to components of ALG$_3$ in this phase. By OPT(M) we denote the cost of migrations performed by OPT in this phase.

The cost of OPT is lower-bounded by the cost of serving σ_I and the cost of serving σ_E. While serving these requests, OPT may perform migrations, and we account for them in both parts: we separately bound OPT by OPT(σ_I) + OPT(M) and OPT(σ_E) + OPT(M). Combining those bounds and using the relation between the maximum and the average, we obtain the bound

$$\begin{aligned}\mathrm{OPT} &\geq \max\{\mathrm{OPT}(\sigma_I) + \mathrm{OPT}(M), \mathrm{OPT}(\sigma_E) + \mathrm{OPT}(M)\} \\ &\geq (\mathrm{OPT}(\sigma_I) + \mathrm{OPT}(M))/2 + (\mathrm{OPT}(\sigma_E) + \mathrm{OPT}(M))/2.\end{aligned}$$

First, we show OPT(M) + OPT(σ_I) $\geq \alpha$. Assume that OPT's partition is fixed throughout the phase (as otherwise OPT pays α for a migration). The phase ended when the components of ALG$_3$ could not be partitioned without splitting them. Hence, for every possible partition of OPT, there exists a non-collocated saturated pair, and OPT paid for α requests that saturated the pair.

Next, we bound OPT(σ_E) + OPT(M). The sequence σ_E accounts only for unsaturated edges, thus by Observation 4.4.5, there are at most $\alpha - 1$ requests to each pair in σ_E. OPT may have at most 3ℓ pairs of nodes collocated in its clusters, and thus avoid paying for $3\ell \cdot (\alpha - 1)$ requests from σ_E. Hence, at least $\chi := |\sigma_E| - 3\ell \cdot (\alpha - 1)$ requests from σ_E are external requests with respect to OPT's configuration at the beginning of the phase. Faced with these requests, OPT may serve them remotely or perform migrations to decrease its cost. By swapping a pair of nodes (u, v), OPT collocates u with two nodes u', u'', and v with two nodes v', v''. This may allow serving requests between (u, u'), (u, u''), (v, v') and (v, v'') for free afterward. Hence, by performing a single swap that costs 2α, OPT may avoid paying the remote serving costs for at most $4(\alpha - 1)$ requests from σ_E. The total cost of OPT is then at least

$$\mathrm{OPT}(\sigma_E) + \mathrm{OPT}(M) \geq \chi \cdot \frac{2\alpha}{4(\alpha - 1)} \geq \frac{|\sigma_E|}{2} - 2\alpha \cdot \ell.$$

Finally, to bound the competitive ratio, we transform the above inequality in the following way: $|\sigma_E| \leq 2(\mathrm{OPT}(\sigma_E) + \mathrm{OPT}(M)) + 4\alpha \cdot \ell$. For succinctness, let $\xi := \mathrm{OPT}(\sigma_E) + \mathrm{OPT}(M)$. Combining the bounds on the cost of ALG$_3$ and OPT during each finished phase, the competitive ratio is

$$\frac{\mathrm{ALG}_3(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{11\alpha \cdot \ell + |\sigma_E|}{\alpha/2 + \xi/2} \leq \frac{30\alpha \cdot \ell + 4 \cdot \xi}{\alpha + \xi} \leq 30\ell.$$

It remains to consider the last, unfinished phase. First, consider the case where the unfinished phase is also the first one. Then, we cannot charge OPT due to the inability to partition the components. Instead, we use the fact that ALG$_3$ and OPT started with the same initial partition. If the input finished before the first α external requests, then ALG$_3$ is 1-competitive. If at least α external requests were issued, then OPT either paid α for serving them remotely or paid α for a migration. Charging this cost to OPT serves the purpose of charging α at the end of a finished phase, and thus we can apply the reasoning as for a finished phase. Second, consider the case, where there are at least two phases. Then we split the cost α of OPT accounted in the penultimate phase into the last two phases, and we repeat the analysis of a finished phase. This way, the competitive ratio increases at most twofold in comparison to a finished phase, and the competitive ratio is $\text{ALG}_3(\sigma)/\text{OPT}(\sigma) \leq 60\ell$. □

Note on Arbitrary Capacity The presented algorithm assumes the servers have capacity 3. The challenge in generalizing beyond this fixed capacity lies in bounding the cost of finding the closest saturated component respecting partition. For the capacity $k = 3$, we bound the cost of each reconfiguration by enumerating all cluster possibilities. We argue that the reconfiguration for $k = 3$ impacts only a constant number of clusters, and its cost is bounded only in terms of k, independently of ℓ. The major challenge at the time of our work was bounding the reconfiguration cost independently of ℓ, which was later addressed by by Bienkowski et al. [28].

Distributed implementation While we have described the algorithm globally so far, we note that it allows for efficient distributed implementations. The algorithm performs two types of operations that require communication with other clusters: a component merge, and a broadcast of the end of the phase. We say that a cluster containing 3 isolated nodes is *fresh*. A merge of two components may require finding a fresh cluster (for details see the proof of Lemma 4.4.4). In the following, we show how to efficiently find a fresh cluster in a distributed manner. We organize the clusters into an arbitrary rooted balanced binary tree, and we broadcast the root to each cluster. Each cluster maintains the counter of fresh clusters in its subtree. To find a fresh cluster, we traverse an arbitrary path of non-zero counters from the root. Upon encountering a fresh cluster, we end the traversal and decrease the counters on the followed path by 1. Summarizing, ending the phase requires a single broadcast, and merging components has $O(\log \ell)$ communication complexity.

4.4.4 Improved Algorithm for Online Rematching

In this section, we present RM, an algorithm for clusters of capacity $k = 2$. We interpret a pair of nodes collocated in one cluster as a “matched” pair. Hence, the problem is an online variant of the maximal matching problem where a matched pair can separate in order to “rematch” with two other nodes. Rematching is necessary for maximizing intra-cluster communications, which is equivalent to minimizing inter-cluster communications. This is known as the *online rematching* problem, and a non-strict 7-competitive algorithm is already given by [19], in which the ratio comes with an additive factor $O(\alpha\ell^2)$. We do not only improve upon their competitive ratio, but also we show that our ratio holds *strictly* (i.e., with no additive factor).

Our algorithm is slightly simpler than the one in [19], while our analysis is simpler and more concise. In the analysis of the algorithm, we propose a novel charging scheme for edges that share a vertex.

Algorithm ReMatch The algorithm ReMatch (RM) maintains a counter $C_{\{x,y\}}$ for each pair of nodes $\{x, y\}$ and increments it on every remote request between x and y. Once $C_{\{x,y\}} = \lambda$, it resets the counter $C_{\{x,y\}} := 0$ and collocates the two nodes by swapping one of them, say x, with the node collocated with y.

Theorem 4.4.6. *For $\lambda = \alpha$, the algorithm RM is strictly 6-competitive.*

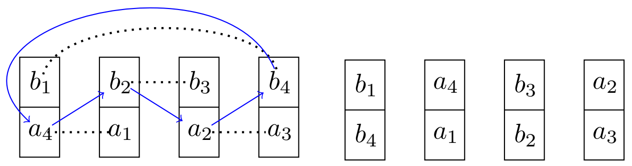


Figure 4.2: Dashed lines represent external requests. An arrow from node x to node y indicates that x replaces y. In the left configuration, OPT collocates 4 pairs by performing 4 migrations, which results in the right configuration.

The charging scheme We charge both OPT and RM whenever RM collocates a pair. RM collocates a pair always with a swap which costs 2α, while OPT may save some costs by collocating multiple pairs at once. Thus it pays the price of only one migration per pair (see Figure 4.2). Therefore, OPT possibly collocates a pair by moving one node to the cluster of the other node paying only α, in contrast to the swapping cost 2α incurred by RM.

Consider two pairs that share the same node, i.e., *intersecting pairs*, and the set of requests that cause (first) collocations of these pairs. This set contains at least one request to each pair, and OPT must pay a non-zero cost over requests in this set, as it cannot have both pairs collocated at the same time. However, we can charge this cost to OPT only the first time RM collocates a pair and not at any later time when RM collocates it a second time. Otherwise, OPT is possibly charged for the same cost repeatedly. For this reason, we charge OPT a cost inflicted by a pair if and only if OPT incurs that cost after the last time RM separates the pair.

Proof of Theorem 4.4.6. Fix an input sequence of requests $\sigma := \{\sigma_1, \dots, \sigma_m\}$. Assume that RM collocates a pair $\{u, v\}$ at time t. The value of $C_{\{u,v\}}$ at t, denoted $C^t_{\{u,v\}}$, reaches λ immediately before RM resets the counter. For any interval $[t_1, t_2]$, by $\sigma_{\{x,y\}}[t_1, t_2]$ we denote the set of all requests to a pair $\{x, y\}$ that arrive during $[t_1, t_2]$. We may use $\sigma_{\{x,y\}}$ whenever the interval $[t_1, t_2]$ is clear from the context.

If t is not the first time that RM collocates $\{u, v\}$ then let $0 < t' < t$ be the latest time when RM separates $\{u, v\}$ in order to collocate some intersecting pair $\{x, y\} \neq \{u, v\}, \{x, y\} cut \{u, v\} \neq \emptyset$, e.g., $\{x, y\} = \{u, w\}$. Else, t is the first time that

RM collocates $\{u, v\}$ and let $t' := 0$. Similarly, if $t' > 0$ is not the first time that RM collocates $\{u, w\}$ then let $0 < t'' < t'$ be the latest time before t' when RM separates $\{u, w\}$. Else, t' is the first time that RM collocates $\{u, w\}$ and we let $t'' = 0$.

First, we bound costs incurred by RM for requests that lead to the collocation of $\{u, v\}$ at time $t \in T$, where $T := \{\tau \in [1, m] \mid \exists\{x, y\} : C^{\tau}_{\{x,y\}} = \lambda\}$ is the set of times when RM performs a collocation. By definitions of t and t', the overall cost of requests in $\sigma_{\{u,v\}}$ incurred by RM, i.e., the total cost of remote serving and the moving cost is $\lambda + 2\alpha$.

Next, we bound costs incurred by RM for requests that do not lead to collocations until the end of the sequence at $t = m$. Assume $\{u, v\}$ is not collocated at $t = m$ and $0 < C^m_{\{u,v\}} < \lambda$, which means RM pays $C^m_{\{u,v\}}$ for requests in $\sigma_{\{u,v\}}(t', m]$. Then the total cost incurred by RM is

$$\mathrm{RM}(\sigma) = \sum_{t\in T}(\lambda + 2\alpha) + \sum_{\{u,v\}} C^m_{\{u,v\}}.$$

Next, we bound costs incurred by OPT for requests that lead to the collocation of $\{u, v\}$ at $t \in T$. If t is the first time that RM collocates $\{u, v\}$, then OPT pays λ for serving requests in $\sigma_{\{u,v\}}[0, t]$ (remotely), or α for collocating the pair and serving (some of) the requests with cost zero. Therefore, in this case, $\mathrm{OPT}(\sigma_{\{u,v\}}(0, t]) \geq \min\{\lambda, \alpha\}$. Otherwise, it is not the first collocation and consider times t' and t'' as defined previously, and let $R_t := \sigma_{\{u,w\}}(t'', t'] \cup \sigma_{\{u,v\}}(t', t]$. We define $R_{t'}$ for the collocation at t' analogously (see Figure 4.3). Then, $\mathrm{OPT}(R_t) = \mathrm{OPT}(\sigma_{\{u,w\}}) + \mathrm{OPT}(\sigma_{\{u,v\}})$.

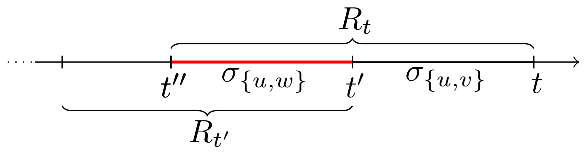


Figure 4.3: Illustration of the timeline used in the proof of Theorem 4.4.6. The set R_t consists of requests to the pairs $\{u, w\}$ and $\{u, v\}$, which arrive in the interval $(t'', t]$. Similarly, $R_{t'}$ consists of requests to $\{u, w\}$ and some other pair irrelevant to the analysis. Hence, requests to $\{u, w\}$ are contained in both sets, and they arrive in $(t'', t']$ highlighted in red thick line.

If OPT has both pairs separated during their respective intervals, then clearly it pays 2λ in those intervals. Note that (trivially) OPT cannot have both pairs collocated at the same time. Else, OPT has one of the pairs, say $\{u, v\}$, already collocated prior its respective interval, $(t', t]$, and keeps it so during this interval. Then it pays zero for requests to this pair. Hence, it pays α for collocating the other pair, in this case $\{u, w\}$, or it pays λ for serving requests in $\sigma_{\{u,w\}}$. Note that OPT may deviate from the two cases by collocating the pair after serving some of its external requests. In any case, $\mathrm{OPT}(R_t) \geq \min\{\lambda, \alpha\} = \alpha$.

It remains to bound the cost incurred by OPT due to requests to $\{u, v\}$ that do not lead to its collocation until the end of the sequence at $t = m$. We bound the cost analogously to the case where RM collocates $\{u, v\}$. If $\{u, v\}$ is not collocated in the initial matching and RM never collocates it, then $C^m_{\{u,v\}} = |\sigma_{\{u,v\}}[1, m]|$. OPT pays $\text{OPT}(\sigma_{\{u,v\}}[1, m]) \geq \min\{\alpha, C^m_{\{u,v\}}\}$, for collocating this pair, or it pays for requests in $\sigma_{\{u,v\}}[1, m]$. Else, either $\{u, v\}$ is collocated in the initial matching or RM collocates it at some point. Then there exists an intersecting pair $\{u, w\}$ that is collocated by RM at $t' < m$, separating $\{u, v\}$. We define times $t'' < t' < m$ analogously to the former case. Let $R^*_{\{u,v\}} := \sigma_{\{u,w\}}(t'', t'] \cup \sigma_{\{u,v\}}(t', m]$. Then, OPT must pay for collocating at least one pair or (and) serving requests to the other pair remotely. Thus, $\text{OPT}(R^*_{\{u,v\}}) \geq \min\{C^m_{\{u,v\}}, \alpha\}$.

Next, we sum up all costs incurred by OPT. By definitions of R_t and $R^*_{\{u,v\}}$, we have either $R_{t'} cut R_t = \sigma_{\{u,w\}}$ or $R_{t'} cut R^*_{\{u,v\}} = \sigma_{\{u,w\}}$. This means, $\text{OPT}(\sigma_{\{u,w\}})$ is counted at most twice in each of the expressions $\text{OPT}(R_{t'}) + \text{OPT}(R_t)$ and $\text{OPT}(R_{t'}) + \text{OPT}(R^*_{\{u,v\}})$. Hence, the total cost to OPT is

$$\text{OPT}(\sigma) = \frac{1}{2}\Big(\sum_{t\in T} \text{OPT}(R_t) + \sum_{\{u,v\}} \text{OPT}(R^*_{\{u,v\}})\Big) \geq \frac{1}{2}\Big(\sum_{t\in T} \alpha + \sum_{\{u,v\}} C^m_{\{u,v\}}\Big).$$

Finally, we bound the competitive ratio by aggregating the above bounds, obtaining

$$\text{RM}(\sigma)/\text{OPT}(\sigma) \leq 2\Big(\sum_{t\in T} 3\alpha + \sum_{\{u,v\}} C^m_{\{u,v\}}\Big) \Big/ \Big(\sum_{t\in T} \alpha + \sum_{\{u,v\}} C^m_{\{u,v\}}\Big) \leq 6.$$

□

4.5 Discussion and Future Work

This paper revisited the online graph partitioning problem and presented several tight bounds for the important model where capacities cannot be exceeded, both for a general partitioning model and for a special learning model.

Our algorithms allow for efficient distributed implementations. The algorithm PPL from Section 4.3.3 can be distributed similarly to the approach in [106]. The algorithm for $k = 2$ from Section 4.4.4 performs only local communication for each request: counters are kept on the clusters and updated locally, and each migration is local within two clusters that reached the collocation threshold λ. Furthermore, we proposed an efficient distributed implementation of the algorithm for $k = 3$ in Section 4.4.3.

There remain several interesting avenues for future research. A general open direction concerns the study of the power of randomization in our setting. It would generally also be interesting to study the performance of our algorithms empirically, under realistic workloads, and engineer algorithms to speed up computations. But there are also more specific open questions. The algorithm from Section 4.4.3 relies on an efficient operation of finding a saturated component respecting partition.

For $k = 3$, we proved that this operation only has a local effect by enumerating all possible cluster configurations. A major question left unresolved is if this operation is local for arbitrary k. Furthermore, our lower bound from Theorem 4.4.2 shed light on the dependency on ℓ in the competitive ratio for the setting with augmentation. The algorithm CREP from [19] requires $(2 + \epsilon)$-augmentation to guarantee the competitive ratio independent of ℓ. In contrast, our construction shows that the linear term ℓ is inevitable if the augmentation is smaller than $4/3$. This raises a question about the tradeoffs between augmentation and the dependency on ℓ for the competitive ratio. Another open question concerns the runtime. The algorithm Perfect Partition Learner runs in superpolynomial time, and the dominating term in the runtime is due to finding the communicating component-respecting partition. We hence wonder if there exists a polynomial-time algorithm achieving an (asymptotically) optimal competitive ratio.

Chapter 5

Waypoint Routing

We are given a link-capacitated network $G = (V, E)$, a source-destination pair $(s, t) \in V^2$ and a set of waypoints $\mathscr{W} \subset V$ (the VNFs). We consider the practically relevant but rarely studied case of bidirected networks. The objective is to find a (shortest) route from s to t such that all waypoints are visited. We show that this problem features interesting connections to classic combinatorial problems, present different algorithms, and derive hardness results.

5.1 Introduction

After revamping the server business, the virtualization paradigm has reached the shores of communication networks. Computer networks have broken with the "end-to-end principle" [164] a long time ago, and today, intermediate nodes called *middleboxes* serve as proxies, caches, wide-area network optimizers, network address translators, firewalls, etc. The number of middleboxes is estimated to be in the order of the number of routers [111].

The virtualization of such middleboxes is attractive not only for cost reasons, but also for the introduced flexibilities, in terms of fast deployment and innovation: in a modern and virtualized computer network, new functionality can be deployed quickly in virtual machines on cloud computing infrastructure. Moreover, the composition of multiple so-called Virtual Network Functions (VNFs) allows to implement complex network services known as *service chains* [151]: traffic between a source s and a destination t needs to traverse a set of network functions (for performance and/or security purposes).

However, the realization of such service chains poses a fundamental algorithmic problem: In order to minimize the consumed network resources, traffic should be routed along *short* paths, while respecting capacity constraints. The problem is particularly interesting as due to the need to traverse waypoints, the resulting route is not a simple path, but *a walk*.

In this paper, we are particularly interested in VNF routing on *bidirected networks*: while classic graph theoretical problems are typically concerned with undirected or directed graphs, real computer networks usually rely on links providing full-duplex communication. Figure 5.1 gives an example.

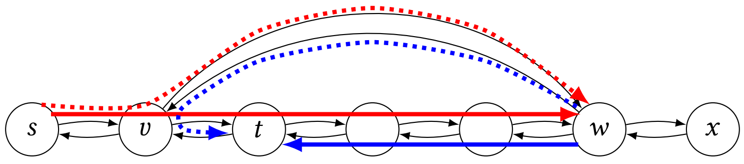


Figure 5.1: In this introductory example, the task is to route the flow of traffic from the source s to the destination t via the waypoint w. When routing via the solid red (s, w) path, followed by the solid blue (w, t) path, the combined walk length is $5 + 3 = 8$. A shorter solution exists via the dotted red and blue paths, resulting in a combined walk length of $2+2 = 4$. Observe that when the waypoint would be on the node x, no node-disjoint path can route from s to t via the waypoint. Furthermore, some combinations can violate unit capacity constraints, e.g., combining the solid red with the dotted blue path induces a double utilization of the link from v to t.

5.1.1 Model

We study full-duplex networks, modeled as connected *bidirected graphs* $G = (V, E)$ [55] with $|V| = n$ nodes (switches, middleboxes, routers) and $|E| = m$ links, where each link $e \in E$ has a capacity $c : E \to \mathbb{N}_{>0}$ and a weight $\omega : E \to \mathbb{N}_{>0}$. Bidirected graphs (also known as, e.g., Asynchronous Transfer Mode (ATM) networks [40] or symmetric digraphs [113]) are directed graphs with the property that if a link $e = (u, v)$ exists, there is also an anti-parallel link $e' = (v, u)$ with $c(e) = c(e')$ and $\omega(e) = \omega(e')$.

Given (1) a bidirected graph, (2) a source $s \in V$ and a destination $t \in V$, and (3) a set of k waypoints $\mathscr{W} \subset V$, the *bidirected waypoint routing problem* BWRP asks for a flow-route $\mathscr{R}$ (i.e., a walk) from s to t that (*i*) visits all waypoints in $\mathscr{W}$ and (*ii*) respects all link capacities. Without loss of generality, we normalize link capacities to the size of the traffic flow, removing links of insufficient capacity.

BWRP comes in two flavors. In the *unordered* version UBWRP, the waypoints $\mathscr{W}$ can be traversed in any order. In the *ordered* version, OBWRP, the waypoints depend on each other and must be traversed in a pre-determined order: every waypoint w_i may be visited at any time in the walk, and as often as desired (while respecting link capacities), but the route $\mathscr{R}$ must contain a given ordered node sequence $s, w_1, w_2, \ldots, w_k, t$. For example, in a network with stringent dependability requirements, it makes sense first route a packet through a fast firewall before performing a deeper (and more costly) packet inspection.

For both UBWRP and OBWRP, we are interested both in *feasible* solutions (respecting capacity constraints) as well as in *optimal* solutions. In the context of the latter, we aim to optimize the cost $|\mathscr{R}|$ of the route $\mathscr{R}$, i.e., we want to minimize the sum of the weights of all traversed links. As we will see, computing an optimal route

$\mathscr{R}^*$ can be hard in general, and hence, we also study $\alpha-$competitive approximation algorithms, where for any computed tour $\mathscr{R}$ holds: $|\mathscr{R}| \leq \alpha \cdot |\mathscr{R}^*|$.

5.1.2 Contributions

We initiate the study of the waypoint routing problem on bidirected networks. We put the problem into perspective with respect to classic combinatorial problems (in particular Steiner Tree problems, variants of Traveling Salesman problems, and link-disjoint paths problems), and present a comprehensive set of algorithms and hardness results.

Unordered bidirected waypoint routing UBWRP. We first show that while *any* UBWRP instance is *feasible*, as each link is traversed only once in each direction, computing *optimal* solutions is NP-hard: no polynomial-time approximation scheme (PTAS) exists unless P=NP. On the positive side, by leveraging connections to metric TSP, we show that an ≈ 1.53-approximation is possible on general graphs. But also *optimal* solutions are possible in polynomial time, namely if the number of waypoints is small, namely $k \in \mathcal{O}(\log n)$: a practically very relevant case. In fact, if the network is planar, we can solve the problem even up to $k \in \mathcal{O}(\log^{2-\varepsilon} n)$ many waypoints in polynomial time (for any constant $\varepsilon > 0$), using a connection to Subset TSP.

Ordered bidirected waypoint routing OBWRP. Due to a connection to link-disjoint paths, it holds that feasible routes can be computed in polynomial time for OBWRP if $k \in \mathcal{O}(1)$. Moreover, while finding optimal routes is NP-hard in general, we show that polynomial-time exact solutions exist for cactus graphs with constant link capacities.

5.1.3 Related Work

While routing through network functions is a standard application in the area of computer networking, we curently witness the emergence of two new trends which change the requirements on routing: (1) (virtualized) network functions are increasingly deployed not only at the edge but also in the network core [60]; (2) network functions are being composed or “chained” to provide more complex services, a.k.a. *service chains* [156],[143],[175]. Traversing these network functions may entail certain detours, i.e., routes do not necessarily follow shortest paths and may even contain loops, i.e., form a *walk* rather than a simple path [23],[61],[62],[136].

Waypoint Routing. Amiri et al. [4] recently provided first results on the ordered waypoint routing problem. In contrast to our work, Amiri et al. [4] focus on directed and undirected graphs only and do not consider approximation algorithms; however, finding a feasible solution to the ordered waypoint routing problem is NP-hard on undirected graphs, and for directed graphs already for a single waypoint. The same limitation holds for the work by Amiri et al. [5] on the unordered waypoint problem on undirected graphs. Moreover, under unit capacities, the undirected problem in [5] also has a different structure, and, e.g., it is not possible to establish the connection to Steiner Tree and Traveling Salesman problem variants as discussed in this paper. On the positive side, it is easy to see that their algorithm to compute optimal unordered solutions on bounded treewidth graphs in XP time also applies to our case; however, their hardness results do not.

Link-Disjoint Paths. Closely related to the (ordered) waypoint routing problem is the study of link-disjoint paths: Given k source-destination pairs, is it possible to find k corresponding pairwise link-disjoint paths? For an overview of results on directed and undirected graphs, we refer to the work of Amiri et al. [4].

For bidirected graphs, deciding the feasibility of link-disjoint paths is NP-hard [40]. When the number of link-disjoint paths is bounded by a constant, feasible solutions can be computed in polynomial time [113]. Extensions to multi-commodity flows have been studied in [112],[183] and parallel algorithms were presented in [119],[120]. To the best of our knowledge, the joint optimization (i.e., shortest total length) of a constant number of link-disjoint paths is still an open problem. In comparison, we can solve UBWRP for a super-constant number of waypoints optimally.

Besides simple graph classes such as trees, we are not aware of algorithms for a super-constant number of link-disjoint paths on directed graphs, which are also applicable to bidirected graphs. We refer to [144] for an annotated tableau. In contrast, we in this paper optimally solve OBWRP on cactus graphs with constant capacity.

Traveling Salesman and Steiner Tree problems. The unordered problem version is related to the *Traveling Salesman problem* (TSP): there, the task is to find a cycle through all nodes of an undirected graph, which does not permit any constant approximation ratio [50], unless P=NP: for the so-called metric version, where nodes may be visited more than once (identical to the TSP with triangle inequality), performing a DFS-tour on a minimum spanning tree (MST) gives a 2-approximation, see, e.g., [50] again. The best known approximation ratio for the metric TSP is 1.5 [48] ($3/2 + 1/34 \approx 1.53$ for $s \neq t$, called the $s - t$ *Path TSP* [168]) and no better polynomial-time approximation ratio than $123/122 \approx 1.008$ is possible [116], unless P=NP. Throughout this paper, when referring to (any variant of) TSP, we usually refer to the metric version on undirected graphs.

A related problem is the NP-hard [88] *Steiner Tree problem* (ST) on undirected graphs: given a set of terminals, construct a minimum weight tree that contains all terminals. If all nodes are terminals, this reduces to the (polynomial) MST problem. The currently best known approximation ratios for ST are $\ln 4 + \varepsilon < 1.39$ [36] (randomized) and $1 + \frac{\ln 3}{2} \approx 1.55$ [159] (deterministic).

Note that both the Steiner Tree and the Traveling Salesman problem are *oblivious* to link capacities. Notwithstanding, we can make use of approximation algorithms for both problems for UBWRP in later sections.

Prize-Collecting and Subset variants. In particular, if $\mathscr{W} \subsetneq V$, we can utilize the *prize-collecting* versions of (path) TSP and ST, called (PC-PATH2) PCTSP and PCST [12]. Here, every node is assigned a non-negative prize (penalty), s.t. if the node is not included in the tour/tree, its prize is added to the cost. For all three prize-collecting variants above, Archer et al. [12] provide approximation ratios smaller than 2: (1) PCTSP: $97/49 < 1.979592$, (2) PC-PATH2: $241/121 < 1.991736$, (3) PCST: 1.9672 (randomized) and 1.9839 (deterministic).

Contained in PC-TSP is *Subset TSP*, first proposed in [13, Sec. 6], which asks for a shortest tour through specified nodes [121]. Klein and Marx [122] give a $\left(2^{\mathcal{O}(\sqrt{k}\log k)} + W\right) \cdot n^{\mathcal{O}(1)}$ for planar graphs, where W is the largest link weight. They also point out that classic dynamic programming techniques [24],[103] have a runtime of $2^k \cdot n^{\mathcal{O}(1)}$ for general graphs. The path version is not considered, we show how to apply any optimal Subset TSP algorithm to UBWRP with $s \neq t$.

Conceptually related (to the non-metric TSP) is the K-cycle problem, which asks for a shortest cycle through K specified nodes or links, i.e., a vertex-disjoint tour. Björklund et al. [30] give a randomized algorithm with a runtime of $2^K \cdot n^{\mathcal{O}(1)}$.

5.1.4 Paper Organization

We begin with studying UBWRP in Sec. 5.2: after giving an introduction to waypoint routing in Sec. 5.2.1, we give hardness and (approximation) algorithm results via metric and subset TSP in Sec. 5.2.2. We then consider the ordered case in Sec. 5.3, tackling constant k in Sec. 5.3.1, we study hardness in Sec. 5.3.2, and finally provide a cactus graph algorithm in Sec. 5.3.3. Lastly, we conclude in Sec. 5.4 with a short summary and outlook.

5.2 The Unordered BWRP

We start our study of bidirected waypoint routing with a short *tour d' horizon* of the problem in Sec. 5.2.1, discussing the case of few waypoints and first general approximations via the Steiner Tree problem. We then follow-up in Sec. 5.2.2 by showing that UBWRP and metric TSP are polynomially equivalent, i.e., algorithmic and hardness results can be reduced. We also establish connections to the subset and prize-collecting TSP variants.

5.2.1 An Introduction to (Unordered) Waypoint Routing

First, we examine the case of a *single waypoint* w, which requires finding a shortest $s-t$ route through this waypoint. Note that for a single waypoint, the two problem variants UBWRP and OBWRP are equivalent.

One waypoint: greedy is optimal. We observe that the case of a single waypoint is easy: simply taking two *shortest paths* (SPs) $P_1 = SP(s,w)$ and $P_2 = SP(w,t)$ in a greedy fashion is sufficient, i.e., the route $\mathscr{R} = P_1P_2$ is always feasible (and thus, also always optimal in regards to total weight).

Suppose this is not the case, that is, $P_1 cut P_2 \neq \emptyset$. Among all nodes in $P_1 cut P_2$, let u and v be, resp., the first and the last nodes w.r.t. to the order of visits in $\mathscr{R}$. Let P_i^{xy} denote the sub-path connecting x to y in P_i. Thereby we have $\mathscr{R} = P_1P_2 = P_1^{su}P_1^{uv}P_1^{vw}P_2^{wu}P_2^{uv}P_2^{vt}$ (Fig. 5.2). Let $\bar{\mathcal{P}}$ be the reverse of any walk $\mathcal{P}$ obtained by

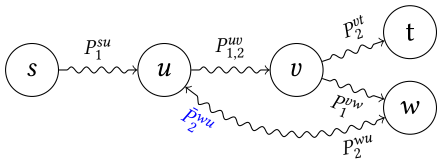


Figure 5.2: The directed path from u to v is traversed two times in $\mathscr{R}$.

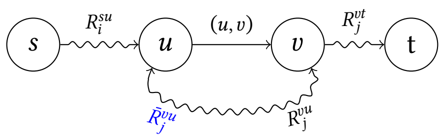


Figure 5.3: The link (u, v) is traversed multiple times in $\mathscr{R}$.

replacing each link $(x, y) \in \mathcal{P}$ with its anti-parallel link (y, x). Observe that for $P_1' = P_1^{su}\bar{P}_2^{wu}$ and $P_2' = \bar{P}_1^{vw}P_2^{vt}$, the feasible route $\mathscr{R}' = P_1'P_2'$ is shorter than $\mathscr{R}$, contradicting P_1 and P_2 both being shortest paths.

Being greedy on the right order is optimal. Next, we show that even more waypoints can be handled efficiently as long as their number is limited: if the optimal traversal order is known, selecting shortest paths in a greedy fashion is again optimal, but to find the optimal order, $k!$ has to be tractable.

We first give an auxiliary lemma, which will also be of later use. We note that Klein and Marx gave an analogous construction for undirected graphs in [122, Figure 1]. The idea can be explained with Fig. 5.3: when a link (u, v) is traversed at least twice, we can take a shortcut from u to v. Iterating this idea ensures at most one traversal per link.

Lemma 5.2.1. *Any (unfeasible) route $\mathscr{R}$ that traverses some link (u, v) more than once, can be efficiently transformed into a shorter route $\hat{\mathscr{R}}$ that traverses each link at most once.*

Proof. There is at least one link that is being shared by at least two sub-routes along $\mathscr{R}$. For every $k, l : k < l$, denote the set of all links $e \in R_k cut R_l$ by W. Let $(u, v) \in W$ be a link traversed by R_i and R_j s.t. $i < j = \arg\min_l((u, v) \in R_i cut R_l)$ (Fig. 5.3). Thus, $\mathscr{R} = R_iR_j = R_i^{su}(uv)R_j^{vu}(uv)R_j^{vt}$. Now consider the new route $\mathscr{R}' = R_1^{su}\bar{R}_2^{vu}R_2^{vt}$. Note that $\mathscr{R}'$ might still traverse (u, v) more than once. Repeat the same procedure for (u, v) until this is not the case. Each time we choose the two sub-routes that preceed any other sub-route traversing (u, v). This ensures the other parts of $\mathscr{R}$ will not be skipped after this transformation. Now we remove this link from W. Since we included additional links for the rerouting, there might be some link $(u', v') \in \bar{R}^{vu} cut \mathscr{R}, (u', v') \neq (u, v)$, that is being traversed more than once in $\mathscr{R}'$, but not in $\mathscr{R}$. Add all such links to W. Repeating the same procedure for every $(x, y) \in W$ transforms $\mathscr{R}'$ into a new route $\mathscr{R}''$ that traverses (x, y) at most once. On the other hand, for each newly included link we exclude its anti-parallel link during the transformation. Therefore the new route is shorter by at least one link, i.e. $|\mathscr{R}''| < |\mathscr{R}|$. Hence, there can be only $O(|\mathscr{R}|)$ iterations, the last of which necessarily ends up at the desired route $\hat{\mathscr{R}}$ satisfying the lemma. □

Using this idea, we can now show that in an optimal ordering, the shortest paths will not overlap.

Lemma 5.2.2. *Given the permutation σ of waypoints in the (first visit) order of an optimal route $\mathscr{R}^*$, we can efficiently construct a route $\mathscr{R}_\sigma$ s.t. $|\mathscr{R}_\sigma| = |\mathscr{R}^*|$.*

Proof. Let $\mathcal{W}_\sigma = w_{\sigma_1} w_{\sigma_2} ... w_{\sigma_k}$ be the order we intend to visit the waypoints, and $\mathscr{R}'$ be an empty route. For each consecutive pair w_{σ_i} and $w_{\sigma_{i+1}}$ add the shortest path links of $SP(w_{\sigma_i}, w_{\sigma_{i+1}})$ and $SP(w_{\sigma_{i+1}}, w_{\sigma_i})$ to $\mathscr{R}'$. We claim that $\mathscr{R}'$ is a feasible, and constitutes an optimal solution to UBWRP. For the sake of contradiction assume this is not the case. Using Lemma 5.2.1 we can construct a feasible route $\hat{\mathscr{R}}$ s.t. $|\hat{\mathscr{R}}| < |\mathscr{R}'|$. This implies that for some shortest path $SP(w_{\sigma_i}, w_{\sigma_j}), |i - j| = 1$, when replaced by the path $\hat{\mathscr{R}}^{w_{\sigma_i} w_{\sigma_j}}$, yields a shorter route, contradicting the optimality of $SP(.)$. □

This directly implies the following Theorem 5.2.3, which is essentially a brute-force approach. We note that implications from the connections between UBWRP and metric TSP in the next section will improve Theorem 5.2.3 in such a way that $k \in \mathcal{O}(\log n)$ becomes tractable.

If the order of visits in $\mathscr{R}^*$ was part of the input, then by Lemma 5.2.2 the union of shortest paths between consecutive waypoints would be necessarily a feasible and therefore an optimal solution. Since this is not the case (i.e. we cannot know the optimal order in advance), one can iterate over all permutations of waypoints and apply Lemma 5.2.2. Then the best of all these iterations will give an optimal route. Thus, for constants c and $c' = c + c\log c$, there are $k! = \left(\frac{c\log n}{\log\log n}\right)! < (c\log n)^{\left(\frac{c\log n}{\log\log n}\right)} = ((c\log n)^{\log n})^{\frac{c}{\log\log n}} < (n^{c\log c} n^c) \in \mathcal{O}(n^{c'})$ iterations. After multiplying by the cost of $O(k)$ calls to $SP(.)$, the polynomial time follows immediately. Thus we have the following theorem:

Theorem 5.2.3. *For $k \in \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$, UBWRP is polynomially solvable.*

We now turn our attention to the general case of $k \in \mathcal{O}(n)$. We first establish a connection to (Steiner) tree problems as well as to the Traveling Salesman problem. Subsequently, we will derive stronger results leveraging the metric TSP (Sec. 5.2.2).
UBWRP is always feasible. Interestingly, UBWRP is always feasible. We begin with the case of $s = t$: first, compute a minimum spanning tree T_U in the undirected version of G, in G_U. Then, traverse T with a DFS-tour starting at s, using every directed link in the bidirected version of T_U exactly once. As every node $v \in V$ will be visited, so will all waypoints $\mathscr{W}$.

The case of $s \neq t$ is similar: Removing the links from the unique $s - t$-path $s, v_1, v_2, \dots, v_p, t$ decomposes T_U into a forest, still containing all nodes in V. We now traverse as follows: Take a DFS-tour of the tree attached to s, then move to v_1, take a DFS-tour of the tree attached to v_2, …, until lastly arriving at t, then taking a DFS-tour of the tree attached to t, finishing at t.

Corollary 5.2.4. *Any instance of UBWRP admits a walk from s to t that traverses any link at most once.*

Approximations via the Steiner Tree problem. For $\mathscr{W} = V$, the above MST approach directly yields a 2-approximation, cf. TSP in [50]. The 2-approximation fails though when $\mathscr{W} \neq V$: e.g., if only one node is a waypoint, visiting all other nodes can add arbitrarily high costs. However, there is a direct duality to the Steiner Tree problem: When setting all waypoints (including s, t) as terminals, an optimal Steiner Tree for these terminals in G_U is a lower bound for an optimal solution to UBWRP: taking the link-set of any route $\mathscr{R}$ in G_U contains the links of a Steiner Tree as a subset. Hence, the construction is analogous to the MST one for $\mathscr{W} = V$. The best known randomized and deterministic approximation ratios for the Steiner Tree problem are respectively $\ln 4 + \varepsilon < 1.39$ [36] and $1 + \frac{\ln 3}{2} \approx 1.55$ [159] any $\varepsilon > 0$, yielding approximation ratios $2 \ln 4 + \varepsilon < 2.78$ and $2 + \ln 3 \approx 3.09$ for UBWRP.

Corollary 5.2.5. *The optimal UBWRP can be approximated with randomized and deterministic algorithms within factors* 2.78 *and* 3.09 *respectively.*

5.2.2 Hardness and Improved Approximation

Next, we show that metric TSP (denoted ΔTSP for the remainder of this section for clarity) is equivalent to our problem of UBWRP on general graphs, in the sense that their corresponding optimal solutions have identical cost.

Theorem 5.2.6. *Let I be an instance of UBWRP on a bidirected graph G and let I' be an instance of the (path) ΔTSP on the metric closure of the corresponding G_U restricted to $\mathscr{W} \cup \{s, t\}$. The cost of optimal solutions for I and I' are identical.*

Proof. We start with $s = t$ for UBWRP. By setting $V = \mathscr{W} \cup \{s, t\}$, the first reduction follows directly. For the other direction, we first construct an instance of ΔTSP. Then, an optimal solution to ΔTSP must imply an optimal solution to UBWRP and vice versa. Let G'_U be the metric closure of G_U restricted to nodes in $\mathscr{W} \cup \{s, t\}$. An optimal TSP cycle C^* in G'_U (after replacing back the shortest path links) corresponds to a route $\mathscr{R}_{C^*}$ on G_U s.t. $|\mathscr{R}_{C^*}| = |C^*|$. Furthermore, $\mathscr{R}_{C^*}$ possibly violates some link capacities in G. Using Lemma 5.2.1 we turn $\mathscr{R}_{C^*}$ to a route $\mathscr{R}'_{C^*}$ feasible for UBWRP. We claim that $\mathscr{R}'_{C^*}$ is optimal. Assume this is not the case, i.e. $|\mathscr{R}'_{C^*}| > |\mathscr{R}^*|$. Let σ be the permutation corresponding to the order of waypoints in $\mathscr{R}^*$. By Lemma 5.2.2, we can construct a feasible route $\mathscr{R}_\sigma$, such that $|\mathscr{R}_\sigma| = |\mathscr{R}^*|$ and it uses only the shortest path links chosen by $SP(w_{\sigma_i}, w_{\sigma_{(i+1) \bmod k}}), 0 \leq i \leq k$. That is, for the cycle C_σ induced by σ on the links of G', we have $|C_\sigma| = \sum_{i=0}^{k} |SP(w_{\sigma_i}, w_{\sigma_{(i+1) \bmod k}})| = |\mathscr{R}_\sigma| \overset{\text{Lemma 5.2.2}}{=} |\mathscr{R}^*| < |\mathscr{R}'_{C^*}| \leq |\mathscr{R}_{C^*}| = |C^*|$ (by Lemma 5.2.1), which contradicts C^* being optimal.

It remains to show, given $\mathscr{R}^*$ and the order of waypoints therein, σ, the cycle C_σ is optimal for ΔTSP (i.e. $|C_\sigma| = |C^*|$). Assume $|C_\sigma| > |C^*|$. As it was shown previously, we can construct a route $\mathscr{R}'_{C^*}$ s.t. $|\mathscr{R}'_{C^*}| \overset{\text{Lemma 5.2.1}}{=} |C^*| < |C_\sigma| \overset{\text{Lemma 5.2.2}}{=} |\mathscr{R}^*|$, which contradicts the optimality of $\mathscr{R}^*$. The proof construction for $s \neq t$ is analogous, replacing the TSP cycle with a path from s to t. □

No PTAS for UBWRP, but good approximation ratios. As already seen in the proof of Theorem 5.2.6, solutions between the corresponding instances of UBWRP and (path) ΔTSP can be efficiently transformed to one of less or identical cost. As such, we can make use of known algorithms and complexity results, resulting in the following two corollaries regarding hardness and approximability:

Corollary 5.2.7. *UBWRP is an NP-hard problem, no better polynomial-time approximation ratio than* $123/122 \approx 1.008$ *is possible [116], unless P=NP.*

Corollary 5.2.8. *For* $s = t$*, UBWRP can be approximated in polynomial time with a ratio of 1.5 [48]. For* $s \neq t$*, a ratio of* $3/2 + 1/34 \approx 1.53$ *can be obtained [168].*

Relations to $(0, \infty)$-PC-TSP & Subset TSP. In the prize-collecting (PC) variant, the classical Steiner Tree problem can be formulated as a $(0, \infty)$-PC-ST: the terminals must be included (∞), while all other nodes are not relevant (0). In an analogous fashion, one can solve the $(0, \infty)$-PC-(path)-TSP on undirected graphs G_U, where the nodes with ∞ are the waypoints (and s, t).

As such, we can now apply all algorithmic and hardness results from the $(0, \infty)$ variant of the prize-collecting (path) TSP. However, the known results on the prize-collecting version of TSP are weaker than the ones of ΔTSP: this fact is not surprising, as PC-TSP is a generalization of its $(0, \infty)$ variant and ΔTSP.

The $(0, \infty)$-PC-TSP with $s = t$ may also be formulated as the Subset TSP, which asks for a (shortest) closed tour that visits a subset of nodes.

At this point one may wonder why to bother with the Subset TSP, given the parallels between UBWRP and the general metric TSP? As pointed out by Klein and Marx [122], the metric closure can destroy graph properties, e.g., planarity. For a more concrete example, consider the metric closure of a tree with unit link weights, removing unit weight properties. Hence, focusing on Subset TSP allows for algorithms with better approximation ratios on special graph classes.

Leveraging Subset TSP results for UBWRP. Klein and Marx [122] consider the Subset TSP problem as a cycle rather than a path. For planar graphs with k "waypoints", they give an algorithm with a runtime of $\left(2^{\mathcal{O}(\sqrt{k}\log k)} + W\right) \cdot n^{\mathcal{O}(1)}$, where W is the size of the largest link weight. Furthermore, they point out that the classic TSP dynamic programming techniques [24],[103] can be applied to Subset TSP (with $s = t$), solving it optimally in a runtime of $2^k \cdot n^{\mathcal{O}(1)}$.

We show an extension to $s \neq t$, which enables us to use "cycle" algorithms as a black box: create a new node *st*, which serves as start and endpoint of the cycle, connecting *st* to two new waypoints w_s, w_t, and in turn w_s to s and w_t to t, where all new links have some arbitrarily large weight γ, see Fig. 5.4. W.l.o.g., we can assume that an optimal cycle solution of this modified graph starts with the path st, w_s, s. It is left to show that the subsequent tour ends with t, w_t, st: if not, the two nodes w_s, w_t would be traversed three times, which is a contradiction to optimality due to the choice of their incident link weight γ. Observe that instead of setting γ to "∞", $\gamma = W \cdot n^2$ suffices. Hence, we can use the algorithms froms [24],[103], [122] for the path version of subset TSP, and therefore, for UBWRP.

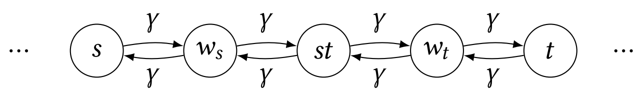


Figure 5.4: Illustration of how to add the node st to the graph, connecting it to both s and t via w_s and w_t, respectively. Observe that a tour must traverse both w_s, w_t.

Corollary 5.2.9. *UBWRP can be solved optimally in a runtime of $2^k \cdot n^{\mathcal{O}(1)}$ for general graphs and in a runtime of $\left(2^{\mathcal{O}(\sqrt{k}\log k)} + W\right) \cdot n^{\mathcal{O}(1)}$, where W is the maximal link weight, for planar graphs.*

I.e., on general graphs, setting $k \in \mathcal{O}(\log n)$ is polynomial. For planar graphs, we can fix any $0 < x < 1$ with $k \in \mathcal{O}(\log^{1+x} n)$, $W \in n^{\mathcal{O}(1)}$ for polynomiality.

5.3 Ordered BWRP

The ordered version of BWRP turns out to be quite different in nature from the unordered one. First, we observe that while every BWRP instance is feasible, there are infeasible OBWRP instances due to capacity constaints. E.g., consider Figure 5.5 with unit capacities and two waypoints. A special case is $k = 1$, which is identical to the unordered case, i.e., routing via one waypoint is always feasible and can be solved optimally in polynomial time.

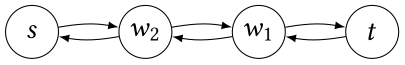


Figure 5.5: In this unit capacity network, the task is to route the flow of traffic from s to w_1, then to w_2, and lastly to t. To this end, the link from w_2 to w_1 must be used twice.

In the following, we study OBWRP in three contexts: (1) polynomial-time algorithms for computing feasible routes if $k \in \mathcal{O}(1)$ (Sec. 5.3.1), NP-hardness of optimality (Sec. 5.3.2), and optimality on cactus graphs, for any number of waypoints and constant capacities (Sec. 5.3.3). The latter is practically motivated by the fact that real computer networks often have specific topologies, and especially router-level wide-area topologies are usually quite sparse.

5.3.1 A Constant Number of Waypoints is Feasible

There is a direct algorithmic connection from the link-disjoint path problem to OBWRP with unit capacities. By setting $s_1 = s$, $t_1 = w_1$, $s_2 = w_1$, $t_2 = w_2$, …, a $k + 1$ link-disjoint path algorithm also solves unit capacity OBWRP for k waypoints. This method can be extended to general capacities via a standard technique, by replacing each link of capacity $c(e)$ with $\lfloor c(e) \rfloor$ parallel links of unit capacity and identical weight. To get rid of the parallel links, replace each link with a path of length two by "placing" a node on it, with the path weight being identical to the link weight.

Hence, we can apply the algorithm from Jarry and Prennes [113], which solves the feasibility of the link-disjoint path problem on bidirected graphs for a constant number of paths in polynomial runtime.

Theorem 5.3.1. *Let $k \in \mathcal{O}(1)$. Feasible solutions for OBWRP can be computed in polynomial time.*

The optimal solution already for few link-disjoint paths still puzzles researchers on bidirected graphs, but the problem seems to be non-trivial on undirected graphs as well: while feasibility for a constant number of link-disjoint paths is polynomial in the undirected case as well [117],[158], optimal algorithms for 3 or more link-disjoint paths are not known, and even for 2 paths the best result is a recent randomized high-order polynomial time algorithm [31]. For directed graphs, already 2 link-disjoint paths pose an NP-hard problem [82]. Results for waypoint routing on directed and undirected graphs are analogous [4].

5.3.2 Optimally Solving OBWRP is NP-Hard

If we transition from a constant number to an arbitrary number of waypoints, we can show that then solving OBWRP optimally becomes NP-hard:

Theorem 5.3.2. *Solving OBWRP optimally is NP-hard.*

Proof. Reduction from the NP-hard link-disjoint path problem on bidirected graphs $G = (V, E)$ [40]: given k source-destination node-pairs (s_i, t_i), $1 \leq i \leq k$, are there k corresponding pairwise link-disjoint paths?

For every such instance I, we create an instance I' of OBWRP as follows, with all unit capacities: Set $s = s_1$ and $t = t_k$, also setting waypoints as follows: $w_1 = t_1$, $w_3 = s_2$, $w_4 = t_2$, $w_6 = s_3$, $w_7 = t_3$, $\ldots$, $w_{3k-3} = s_k$. We also create the missing $k - 1$ waypoints w_2, w_5, w_8, $\ldots$, w_{3k-4} as new nodes and connect them as follows, each time with bidirected links of weight γ: w_2 to $w_1 = t_1$ and $w_3 = s_2$, w_5 to $w_4 = t_2$ and $w_6 = s_3$, $\ldots$, w_{3k-4} to $w_{3k-3} = s_k$ and $w_{3k-5} = t_{k-1}$. I.e., we sequentially connect the end- and start-points of the paths.

Observe that OBWRP is feasible on I' if I is feasible: We take the k link-disjoint paths from I and connect them via the $k - 1$ new nodes in I'.

We now set γ to some arbitrarily high weight, e.g., $3k$ times the sum of all link weights. I.e., it is cheaper to traverse every link of I even $3k$ times rather than paying γ once. As thus, if I is feasible, the optimal solution of I' has a cost of less than $2 \cdot k \cdot \gamma$.

Assume I is not feasible, but that I' has a feasible solution $\mathscr{R}$. Observe that a feasible solution of I' needs to traverse the $k - 1$ new waypoints, i.e., has at least a cost of $2(k-1)\gamma$. As I was not feasible, we will now show that traversing every new waypoint $w_2, w_5, \ldots$ only once is not sufficient for a feasible solution of I'. Assume for contradiction that one traversal of $w_2, w_5, \ldots$ suffices: for each of those traversals of such a w_j, it holds that it must take place after traversing all waypoints with index smaller than j. Hence, we can show by induction that the removal of the links incident to the waypoints $w_2, w_5, \ldots$ from $\mathscr{R}$ contains a feasible solution for I. As thus, at least one of the waypoints $w_2, w_5, \ldots$ must be traversed twice, i.e., $\mathscr{R}$ has a cost of at least $2 \cdot k \cdot \gamma$.

We can now complete the polynomial reduction, by studying the cost (feasibility) of an optimal solution of I': if the cost is less than $2 \cdot k \cdot \gamma$, I is feasible, but if the cost is at least $2 \cdot k \cdot \gamma$ (or infeasible), I is not feasible. □

While many OBWRP instances are not feasible (already in Figure 5.5), we conjecture that the feasibility of OBWRP with arbitrarily many waypoints is NP-hard as well. This conjecture is supported by the fact that the analogous link-disjoint feasibility problems are NP-hard on undirected [88], directed [82], and bidirected graphs [40], also for undirected and directed ordered waypoint routing [4]. We thus turn our attention to special graph classes.

5.3.3 Optimality on the Cactus with Constant Capacity

The difficulty of OBWRP lies in the fact that the routing from w_i to w_{i+1} can be done along multiple paths, each of which could congest other waypoint connections. Hence, it is easy to solve OBWRP optimally (or check for infeasibility) on trees, as each path connecting two successive waypoints is unique.

Lemma 5.3.3. *OBWRP can be solved optimally in polynomial time on trees.*

For multiple path options, the problem turns NP-hard though (Theorem 5.3.2). To understand the impact of already two options, we follow-up by studying rings.

Lemma 5.3.4. *OBWRP is optimally solvable in polynomial time on bidirected ring graphs where for at least one link e holds: $c(e) \in \mathcal{O}(1)$.*

Proof. We begin our proof with $c(e) = c(e') = 1$. Observe that every routing between two successive waypoints has two path options P, clockwise or counter-clockwise. We assign one arbitrary path P_e to traverse e, and another arbitrary path $P_{e'}$ to traverse e'. By removing the fully utilized e and e', the remaining graph is a tree with two leaves, where all routing is fixed, cf. Lemma 5.3.3.

We now count the path assignment possibilities for e, e': by also counting the "empty assignment", we have at most $(n+1)n$ options, where the optimal routing immediately follows for each option. For these $\mathcal{O}(n^2)$ possibilities, we pick the shortest feasible one. I.e., OBWRP can be solved optimally in polynomial time on rings with unit capacity. To extend the proof to constant capacities $c(e) \in \mathcal{O}(1)$, we use an analogous argument, the number of options for assignments to e and e' are now $\mathcal{O}\left(n^{2c(e)}\right) \in \mathrm{P}$. As thus, the lemma statement holds. □

We now focus on the important case of cactus networks. Our empirical study using the Internet Topology Zoo[1] data set shows that over 30% are *cactus graphs.*

[1]See http://www.topology-zoo.org/.

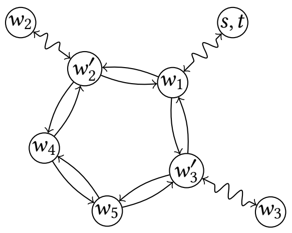


Figure 5.6: In this cactus graph, we illustrate the algorithm of Theorem 5.3.5 w.r.t. the permutation $w_1w_2w_3w_4w_5$.

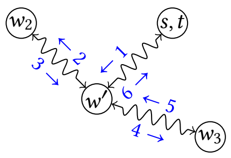


Figure 5.7: Once the ring links are contracted, w' replaces the whole ring. Consequently, the permutation reduces to $w'w_2w_3$. The sub-routes are numbered sequentially.

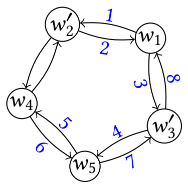


Figure 5.8: The permutation induced on the ring is $w_1w_2'w_3'w_4w_5$. In the sub-problem, we have $s = t = w_1$. The numbers represent the order of node traversal in the optimal route.

Theorem 5.3.5. *OBWRP is optimally solvable in polynomial time on cactus graphs with constant capacity.*

Proof. The idea is to 1) shrink the cactus graph down to a tree, 2) see if for the relevant subset of waypoints (to be described shortly) the feasibility holds on that tree, 3) reincorporate the excluded rings and find the optimal choice of path segments within each ring, and 4) construct an optimal route by stitching together the sub-routes obtained from the tree and the segments from each ring.

Let C be the cactus graph (Fig. 5.6) and T_C be the tree obtained after contracting all the links on each rings. As a result of this link contraction, those waypoints previously residing on rings are now replaced by new (super) waypoints in T_C (Fig. 5.7). Each super node represents either a subtree of adjacent rings or just an isolated ring. Let $\mathscr{W}'$ denote the waypoints in T_C.
Observe that any feasible route in C through $\mathscr{W}$ corresponds to one unique feasible route in T_C through nodes in $\mathscr{W}'$. Next, we show that either the feasible route in T_C (if exists) can be expanded to an optimal route for C, or there is no feasible route in C at all. If T_C is not feasible then we are done. Otherwise, let $\mathscr{R}$ be the (unique) route in this tree. For each ring, $\mathscr{R}$ induces some *endpoints* (Fig. 5.8), one endpoint on each node that is either a) the joint of T_C and the ring, or b) the joint with its adjacent rings. Now we focus on the subproblem induced by this ring and the new waypoint set $\mathscr{W}''$ (to be specified) as follows.

For each endpoints that is visited by $\mathscr{R}$ add a waypoint to $\mathscr{W}''$. Then, using the algorithm described in the proof of Lemma 5.3.4, find an optimal route $\mathscr{R}_{ring}$ visiting all the nodes in $\mathscr{W}''$ respecting the order imposed by $\mathscr{R}$. If no such route exists, the instance is not feasible. Otherwise, remove from $\mathscr{R}$ every occurrence of the super node that represents this ring to get a disconnected route. For each missing part, reconnect the endpoints using the segment of $\mathscr{R}_{ring}$ restricted to these endpoints. Repeat this for every ring; denote the resulting route as $\mathscr{R}'$.

Finally, we argue that $\mathscr{R}'$ is optimal. This is the case because its pieces were taken from sets of sub-routes, where each set, covers a disjoint–or more precisely, vertex-disjoint up to endpoints–component of C. Moreover, the set of sub-routes taken from an individual (disjoint) component (i.e. tree or ring) is optimal on that component. Therefore the total length is optimal. □

5.4 Conclusion

We initiated the study of a natural problem in full-duplex networks: routing through a given set of network functions, the so-called waypoints. We showed that an optimal routing through a super-constant number of $\mathcal{O}(\log n)$ unordered waypoints can be computed in polynomial time, but that the general optimization problem is NP-hard. Nonetheless, we provided approximation algorithms with small constant competitive ratios for any number of waypoints, via connections to the Steiner Tree and (prize-collecting) Traveling Salesman problems. We also presented hardness results and polynomial-time algorithms for the ordered variant. In particular, we derived an exact polynomial-time algorithm for cactus graphs.

We believe that our work opens several interesting directions for future research. In general, while practically relevant, bidirected networks are not well-understood today, and assume an interesting position between directed and undirected networks. In particular, it would be interesting to understand for which bidirected graph classes the ordered and the unordered waypoint routing problem permits polynomial-time algorithms, and for up to how many waypoints. Another interesting direction for future research concerns the study of randomized algorithms.

Bibliography

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. “Competitive analysis of randomized paging algorithms.” In: *Theoretical Computer Science* 234.1–2 (2000), pp. 203–218.

[2] Sugam Agarwal, Murali S. Kodialam, and T. V. Lakshman. “Traffic engineering in software defined networks.” In: *INFOCOM*. IEEE, 2013, pp. 2211–2219.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. 1993.

[4] S. Akhoondian Amiri, K.-T. Foerster, R. Jacob, and S. Schmid. “Charting the Complexity Landscape of Waypoint Routing.” In: *arXiv preprint arXiv:1705.00055* (2017).

[5] S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid. “Walking Through Waypoints.” In: *arXiv preprint arXiv:1708.09827* (2017).

[6] Dan Alistarh, Jennifer Iglesias, and Milan Vojnovic. “Streaming Min-max Hypergraph Partitioning.” In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 1900–1908.

[7] Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, Mahmoud Parham, and Stefan Schmid. “Waypoint Routing in Special Networks.” In: *Proc. IFIP Networking*. 2018.

[8] Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid. “Charting the Algorithmic Complexity of Waypoint Routing.” In: *CCR* 48.1 (2018), pp. 42–48.

[9] Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid. “Walking Through Waypoints.” In: *Algorithmica* 82.7 (2020), pp. 1784–1812.

[10] Konstantin Andreev and Harald Räcke. “Balanced Graph Partitioning.” In: *Theory of Computing Systems* 39.6 (2006), pp. 929–939.

[11] George Andrews and Kimmo Eriksson. *Integer Partitions*. Cambridge University Press. ISBN: 0-521-60090-1.

[12] Aaron Archer, Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi, and Howard J. Karloff. “Improved Approximation Algorithms for Prize-Collecting Steiner Tree and TSP.” In: *SIAM J. Comput.* 40.2 (2011), pp. 309–332.

[13] Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. “A Polynomial-Time Approximation Scheme for Weighted Planar Graph TSP.” In: *Proc. SODA*. 1998.

[14] Sanjeev Arora, David R. Karger, and Marek Karpinski. “Polynomial Time Approximation Schemes for Dense Instances of NP-Hard Problems.” In: *Journal of Computer and System Sciences* 58.1 (1999), pp. 193–210.

[15] Alia K Atlas and Alex Zinin. “Basic specification for IP fast-reroute: loop-free alternates.” In: *IETF RFC 5286* (2008).

[16] FranAois Aubry. “Models and Algorithms for Network Optimization with Segment Routing.” PhD thesis. UCLouvain, 2020.

[17] François Aubry, David Lebrun, Stefano Vissicchio, Minh Thanh Khong, Yves Deville, and Olivier Bonaventure. “Scmon: Leveraging segment routing to improve network monitoring.” In: *Proc. IEEE INFOCOM*. 2016.

[18] François Aubry, Stefano Vissicchio, Olivier Bonaventure, and Yves Deville. “Robustly disjoint paths with segment routing.” In: *CoNEXT*. ACM, 2018, pp. 204–216.

[19] Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. “Dynamic Balanced Graph Partitioning.” In: *SIAM J. Discret. Math.* 34.3 (2020), pp. 1791–1812.

[20] Chen Avin, Louis Cohen, Mahmoud Parham, and Stefan Schmid. “Competitive clustering of stochastic communication patterns on a ring.” In: *Computing* 101.9 (2019), pp. 1369–1390.

[21] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. “On the Complexity of Traffic Traces and Implications.” In: *Proc. ACM SIGMETRICS*. 2020.

[22] Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. “Online Balanced Repartitioning.” In: *DISC* (2016), pp. 243–256.

[23] Nikhil Bansal, Kang-Won Lee, Viswanath Nagarajan, and Murtaza Zafer. “Minimum congestion mapping in a cloud.” In: *Proc. PODC*. 2011.

[24] Richard Bellman. “Dynamic Programming Treatment of the Travelling Salesman Problem.” In: *J. ACM* 9.1 (1962), pp. 61–63.

[25] Randeep Bhatia, Fang Hao, Murali Kodialam, and TV Lakshman. “Optimized network traffic engineering using segment routing.” In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE. 2015, pp. 657–665.

[26] Randeep Bhatia, Fang Hao, Murali Kodialam, and TV Lakshman. “Optimized network traffic engineering using segment routing.” In: *IEEE INFOCOM*. 2015.

[27] Randeep Bhatia, Fang Hao, Murali S. Kodialam, and T. V. Lakshman. “Optimized network traffic engineering using segment routing.” In: *INFOCOM*. IEEE, 2015, pp. 657–665.

[28] Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jirí Sgall, and Pavel Veselý. "Improved Analysis of Online Balanced Clustering." In: *CoRR* abs/2107.00145 (2021). arXiv: `2107.00145`.

[29] Marcin Bienkowski, Jaroslaw Byrka, and Marcin Mucha. "Dynamic Beats Fixed: On Phase-based Algorithms for File Migration." In: *ACM Transactions on Algorithms* 15.4 (2019), 46:1–46:21.

[30] Andreas Björklund, Thore Husfeld, and Nina Taslaman. "Shortest Cycle Through Specified Elements." In: *Proc. SODA*. 2012.

[31] Andreas Björklund and Thore Husfeldt. "Shortest two disjoint paths in polynomial time." In: *Proc. ICALP*. 2014.

[32] Jeremias Blendin, Julius Rückert, Nicolai Leymann, Georg Schyguda, and David Hausheer. "Position paper: Software-defined network service chaining." In: *2014 Third European Workshop on Software Defined Networks*. IEEE. 2014, pp. 109–114.

[33] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[34] Michael Borokhovich and Stefan Schmid. "How (Not) to Shoot in Your Foot with SDN Local Fast Failover: A Load-Connectivity Tradeoff." In: *Proc. OPODIS*. 2013.

[35] Costas Busch, Srikanth Surapaneni, and Srikanta Tirthapura. "Analysis of link reversal routing algorithms for mobile ad hoc networks." In: *Proc. ACM SPAA*. ACM, 2003.

[36] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. "Steiner Tree Approximation via Iterative Randomized Rounding." In: *J. ACM* 60.1 (2013), 6:1–6:33.

[37] C. Hopps. *Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992.* Tech. rep. Apr. 2000.

[38] Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. "A Distributed and Robust SDN Control Plane for Transactional Network Updates." In: *Proc. IEEE INFOCOM*. 2015.

[39] B. Carpenter and S. Brim. *Middleboxes: Taxonomy and Issues.* RFC 3234. `http://www.rfc-editor.org/rfc/rfc3234.txt`. RFC Editor, Feb. 2002.

[40] Pascal Chanas. "Reseaux atm: conception et optimisation." PhD thesis. University of Grenoble, 1998, 143 P.

[41] Marco Chiesa, Andrei Gurtov, Aleksander Mądry, Slobodan Mitrović, Ilya Nikolaevkiy, Aurojit Panda, Michael Schapira, and Scott Shenker. "Exploring the Limits of Static Failover Routing (v4)." In: *arXiv:1409.0034 [cs.NI]* (2016).

[42] Marco Chiesa, Andrei V. Gurtov, Aleksander Madry, Slobodan Mitrovic, Ilya Nikolaevskiy, Michael Schapira, and Scott Shenker. “On the Resiliency of Randomized Routing Against Multiple Edge Failures.” In: *Proc. ICALP.* 2016.

[43] Marco Chiesa, Guy Kindler, and Michael Schapira. “Traffic Engineering With Equal-Cost-MultiPath: An Algorithmic Perspective.” In: *IEEE/ACM Transactions on Networking* 25.2 (2017), pp. 779–792.

[44] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. “On the Resiliency of Static Forwarding Tables.” In: *IEEE/ACM Trans. Netw.* 25.2 (2017), pp. 1133–1146.

[45] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Aurojit Panda, Andrei Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. “The Quest for Resilient (Static) Forwarding Tables.” In: *Proc. IEEE INFOCOM.* 2016.

[46] Hongsik Choi, Suresh Subramaniam, and Hyeong-Ah Choi. “On double-link failure recovery in WDM optical networks.” In: *Proc. IEEE INFOCOM.* 2002.

[47] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. “Managing Data Transfers in Computer Clusters with Orchestra.” In: *ACM SIGCOMM* 41.4 (2011), pp. 98–109.

[48] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Technical Report 388. Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[49] Cisco. *Configuring OSPF.* Tech. rep. Apr. 1997.

[50] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

[51] Daniel Sleator and Robert Tarjan. “Amortized efficiency of list update and paging rules.” In: *Communications of the ACM* 28.2 (1985), pp. 202–208.

[52] Luca Davoli, Luca Veltri, Pier Luigi Ventre, Giuseppe Siracusano, and Stefano Salsano. “Traffic engineering with segment routing: SDN-based architectural design and open source implementation.” In: *Proc. EWSDN.* 2015.

[53] Advait Abhay Dixit, Pawan Prakash, and Ramana Rao Kompella. “On the efficacy of fine-grained traffic splitting protocolsin data center networks.” In: *SIGCOMM.* ACM, 2011, pp. 430–431.

[54] Fabien Duchêne, David Lebrun, and Olivier Bonaventure. “SRv6Pipes: enabling in-network bytestream functions.” In: *Proc. IFIP Networking.* 2018.

[55] Jack Edmonds and Ellis Johnson. “Matching: a well-solved class of linear programs.” In: *Combinatorial Structures and their Applications: Proceedings of the Calgary Symposium.* New York: Gordon and Breach, 1970, pp. 88–92.

[56] Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. “IP fast rerouting for multi-link failures.” In: *IEEE/ACM Trans. Netw* 24.5 (2016), pp. 3014–3025.

[57] Gábor Enyedi, Gábor Rétvári, and Tibor Cinkler. "A novel loop-free IP fast reroute algorithm." In: *Meeting of the European Network of Universities and Companies in Information and Communication Engineering*. Springer. 2007, pp. 111–119.

[58] Leah Epstein, Csanád Imreh, Asaf Levin, and Judit Nagy-György. "On Variants of File Caching." In: *Proc. 38th Int. Colloq. on Automata, Languages and Programming (ICALP)*. 2011, pp. 195–206.

[59] ETSI. "Network Functions Virtualisation – Introductory White Paper." In: *White Paper* (Oct. 2013).

[60] ETSI. "Network Functions Virtualisation – Introductory White Paper." In: *White Paper* (Oct. 2013).

[61] Guy Even, Moti Medina, and Boaz Patt-Shamir. "Online Path Computation and Function Placement in SDNs." In: *Proc. SSS*. 2016.

[62] Guy Even, Matthias Rost, and Stefan Schmid. "An Approximation Algorithm for Path Computation and Function Placement in SDNs." In: *Proc. SIROCCO*. 2016.

[63] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The Road to SDN." In: *Queue* 11.12 (Dec. 2013), 20:20–20:40.

[64] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The road to SDN: an intellectual history of programmable networks." In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 87–98.

[65] Uriel Feige and Robert Krauthgamer. "A polylogarithmic approximation of the minimum bisection." In: *SIAM Journal on Computing* 31.4 (2002), pp. 1090–1118.

[66] Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. "Approximating the minimum bisection size (extended abstract)." In: *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*. 2000, pp. 530–536.

[67] Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. "Brief Announcement: On the Resilience of Routing Tables." In: *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*. 2012, pp. 237–238.

[68] Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. "On the Resilience of Routing Tables." In: *CoRR* abs/1207.3732 (2012). arXiv: `1207.3732`.

[69] Thomas Fenz, Klaus-Tycho Förster, Mahmoud Parham, Stefan Schmid, and Nikolaus Süß. "Traffic Engineering with Joint Link Weight and Segment Optimization." In: (Sept. 2021). `https://whatif-tools.net/segment-routing`.

[70] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. “Competitive paging algorithms.” In: *Journal of Algorithms* 12.4 (1991), pp. 685–699.

[71] Clarence Filsfils, Nagendra Kumar Nainar, Carlos Pignataro, Juan Camilo Cardona, and Pierre Francois. “The segment routing architecture.” In: *Proc. GLOBECOM.* 2015.

[72] Clarence Filsfils, Stefano Previdi, Bruno Decraene, Stephane Litkowski, and Rob Shakir. “Segment Routing Architecture.” In: *IETF Internet-Draft.* 2017.

[73] Clarence Filsfils, Stefano Previdi, Bruno Decraene, Stephane Litkowski, and Rob Shakir. “Segment Routing Architecture.” In: *IETF Internet-Draft.* 2017.

[74] Clarence Filsfils, Stefano Previdi, John Leddy, S. Matsushima, and D. Voyer. *IPv6 Segment Routing Header (SRH).* Internet-Draft draft-ietf-6man-segment-routing-header-14. Work in Progress. Internet Engineering Task Force, June 2018. 29 pp.

[75] Clarence Filsfils et al. “Segment Routing Architecture.” In: *Segment Routing Use Cases, IETF Internet-Draft.* 2014.

[76] Clarence Filsfils et al. “Segment Routing Architecture.” In: *Segment Routing Use Cases, IETF Internet-Draft.* 2014.

[77] Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. “On the Feasibility of Perfect Resilience with Local Fast Failover.” In: *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021.* Ed. by Michael Schapira. SIAM, 2021, pp. 55–69.

[78] Klaus-Tycho Foerster, Mahmoud Parham, Marco Chiesa, and Stefan Schmid. “TI-MFA: Keep calm and reroute segments fast.” In: *Global Internet Symposium (GI).* 2018.

[79] Klaus-Tycho Foerster, Yvonne Anne Pignolet, Stefan Schmid, and Gilles Trédan. “Local Fast Failover Routing With Low Stretch.” In: *CCR* 48.1 (2018), pp. 35–41.

[80] Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan. “Local Fast Failover Routing With Low Stretch.” In: *ACM SIGCOMM CCR* 1 (Jan. 2018), pp. 35–41.

[81] Tobias Forner, Harald Raecke, and Stefan Schmid. “Online Balanced Repartitioning of Dynamic Communication Patterns in Polynomial Time.” In: *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS).* 2021.

[82] Steven Fortune, John E. Hopcroft, and James Wyllie. “The Directed Subgraph Homeomorphism Problem.” In: *Theor. Comput. Sci.* 10 (1980), pp. 111–121.

[83] Bernard Fortz and Mikkel Thorup. “Increasing Internet Capacity Using Local Search.” In: *Computational Optimization and Applications* 29.1 (2004), pp. 13–48.

[84] Bernard Fortz and Mikkel Thorup. “Internet Traffic Engineering by Optimizing OSPF Weights.” In: *INFOCOM*. IEEE Computer Society, 2000, pp. 519–528.

[85] Pierre François, Clarence Filsfils, Ahmed Bashandy, and Bruno Decraene. *Topology Independent Fast Reroute using Segment Routing*. Internet-Draft draft-francois-segment-routing-ti-lfa-00. Internet Engineering Task Force, Nov. 2013.

[86] Pierre François, Clarence Filsfils, John Evans, and Olivier Bonaventure. “Achieving sub-second IGP convergence in large IP networks.” In: *CCR* 35.3 (2005), pp. 35–44.

[87] Eli M. Gafni and Dimitri P. Bertsekas. “Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology.” In: *IEEE Transactions on Communications* 29.1 (Jan. 1981), pp. 11–18. ISSN: 0090-6778.

[88] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[89] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. “Some Simplified NP-Complete Graph Problems.” In: 1.3 (1976), pp. 237–267.

[90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. 1990.

[91] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. “Understanding network failures in data centers: measurement, analysis, and implications.” In: *ACM SIGCOMM CCR*. Vol. 41. 2011, pp. 350–361.

[92] Alessio Giorgetti, Piero Castoldi, Filippo Cugini, Jeroen Nijhof, Francesco Lazzeri, and Gianmarco Bruno. “Path encoding in segment routing.” In: *Proc. IEEE GLOBECOM*. 2015.

[93] Frank Göring. “Short proof of Menger’s Theorem.” In: *Discret. Math.* 219.1-3 (2000), pp. 295–296.

[94] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. “BCube: a high performance, server-centric network architecture for modular data centers.” In: *Proc. ACM SIGCOMM*. 2009.

[95] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. “Traveling with a pez dispenser (or, routing issues in mpls).” In: *SIAM Journal on Computing* 34.2 (2005), pp. 453–474.

[96] *Gurobi Optimizer 9.1.2.* `https://support.gurobi.com/hc/en-us/articles/360060235871-Gurobi-9-1-2-released`. Last Accessed: 2021-06-28.

[97] Prashanth Hande, Mung Chiang, Robert Calderbank, and Sundeep Rangan. "Network pricing and rate allocation with content-provider participation." In: *Proc. IEEE INFOCOM.* 2010.

[98] Fang Hao, Murali Kodialam, and TV Lakshman. "Optimizing restoration with segment routing." In: *Proc. INFOCOM.* 2016.

[99] Ed Harrison, Adrian Farrel, and Ben Miller. "Protection and restoration in MPLS networks." In: *Data Connection White Paper* (2001).

[100] Renaud Hartert. "Fast and scalable optimization for segment routing." PhD thesis. UCLouvain, 2018.

[101] Renaud Hartert, Stefano Vissicchio, Pierre Schaus, Olivier Bonaventure, Clarence Filsfils, Thomas Telkamp, and Pierre Francois. "A declarative and expressive approach to control forwarding paths in carrier-grade networks." In: *ACM SIGCOMM CCR.* Vol. 45. 4. 2015, pp. 15–28.

[102] Mu He, Alberto Martínez Alba, Arsany Basta, Andreas Blenk, and Wolfgang Kellerer. "Flexibility in Softwarized Networks: Classifications and Research Challenges." In: *IEEE Commun. Surv. Tutorials* 21.3 (2019), pp. 2600–2636.

[103] Michael Held and Richard M. Karp. "The traveling-salesman problem and minimum spanning trees: Part II." In: *Math. Program.* 1.1 (1971), pp. 6–25.

[104] Urs Hengartner, Sue B. Moon, Richard Mortier, and Christophe Diot. "Detection and analysis of routing loops in packet traces." In: *Proceedings of the 2nd ACM SIGCOMM Internet Measurement Workshop, IMW 2002, Marseille, France, November 6-8, 2002.* ACM, 2002, pp. 107–112.

[105] Monika Henzinger, Stefan Neumann, Harald Raecke, and Stefan Schmid. "Tight Bounds for Online Graph Partitioning." In: *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA).* 2021.

[106] Monika Henzinger, Stefan Neumann, and Stefan Schmid. "Efficient Distributed Workload (Re-)Embedding." In: *Proceedings of ACM SIGMETRICS / IFIP Performance 2019.* 2019.

[107] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. "Achieving high utilization with software-driven WAN." In: *Proc. ACM SIGCOMM.* 2013.

[108] Sandy Irani. "Page Replacement with Multi-Size Pages and Applications to Web Caching." In: *Algorithmica* 33.3 (2002), pp. 384–409.

[109] J. Feigenbaum et al. "BA: On the resilience of routing tables." In: *Proc. ACM Symposium on Principles of Distributed Computing (PODC).* 2012, pp. 237–238.

[110] J. Moy. *OSPF Version 2.* Tech. rep. Apr. 1998.

[111] J. Sherry et al. "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service." In: *Proc. ACM SIGCOMM.* 2012.

[112] Aubin Jarry. "Multiflows in symmetric digraphs." In: *Discrete Applied Mathematics* 160.15 (2012), pp. 2208–2220.

[113] Aubin Jarry and Stéphane Pérennes. "Disjoint paths in symmetric digraphs." In: *Discrete Applied Mathematics* 157.1 (2009), pp. 90–97. ISSN: 0166-218X.

[114] Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen. "P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures." In: *Proc. ACM CoNEXT.* 2018.

[115] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan S. Owicki. "Competitive Randomized Algorithms for Nonuniform Problems." In: *Algorithmica* 11.6 (1994), pp. 542–571.

[116] Marek Karpinski, Michael Lampis, and Richard Schmied. "New inapproximability bounds for TSP." In: *J. Comput. Syst. Sci.* 81.8 (2015), pp. 1665–1677.

[117] Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. "The disjoint paths problem in quadratic time." In: *J. Comb. Theory, Ser. B* 102.2 (2012), pp. 424–435.

[118] Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein, and Stefan Schmid. "Adaptable and Data-Driven Softwarized Networks: Review, Opportunities, and Challenges." In: *Proc. IEEE* 107.4 (2019), pp. 711–731.

[119] Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. "Processor Efficient Parallel Algorithms for the Two Disjoint Paths Problem and for Finding a Kuratowski Homeomorph." In: *SIAM J. Comput.* 21.3 (1992), pp. 486–506.

[120] Samir Khuller and Baruch Schieber. "Efficient Parallel Algorithms for Testing k-Connectivity and Finding Disjoint s-t Paths in Graphs." In: *SIAM J. Comput.* 20.2 (1991), pp. 352–375.

[121] Philip N. Klein. "A subset spanner for Planar graphs, : with application to subset TSP." In: *Proc. STOC.* 2006.

[122] Philip N. Klein and Dániel Marx. "A subexponential parameterized algorithm for Subset TSP on planar graphs." In: *Proc. SODA.* 2014.

[123] Simon Knight, Hung X. Nguyen, Nick Falkner, Rhys Alistair Bowden, and Matthew Roughan. "The Internet Topology Zoo." In: *IEEE J. Sel. Areas Commun.* 29.9 (2011), pp. 1765–1775.

[124] Robert Krauthgamer and Uriel Feige. "A Polylogarithmic Approximation of the Minimum Bisection." In: *SIAM Review* 48.1 (2006), pp. 99–130.

[125] Diego Kreutz, Fernando MV Ramos, P Esteves Verissimo, C Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. "Software-defined networking: A comprehensive survey." In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[126] Kin-Wah Kwong, Lixin Gao, Roch Guérin, and Zhi-Li Zhang. "On the feasibility and efficacy of protection routing in IP networks." In: *IEEE/ACM Trans. Netw* 19.5 (2011), pp. 1543–1556.

[127] Karthik Lakshminarayanan, Matthew Caesar, Murali Rangan, Tom Anderson, Scott Shenker, and Ion Stoica. "Achieving convergence-free routing using failure-carrying packets." In: *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007.* ACM, 2007, pp. 241–252.

[128] David LeBrun. *Virtual networks testing framework (nanonet).* `https://github.com/segment-routing/nanonet`. Last Accessed: 2021-06-28.

[129] David Lebrun. "Reaping the Benefits of IPv6 Segment Routing." In: *PhD Thesis (preliminary).* 2017.

[130] David Lebrun. "Reaping the Benefits of IPv6 Segment Routing." PhD thesis. UCLouvain / ICTEAM / EPL, Oct. 2017.

[131] David Lebrun and Olivier Bonaventure. "Implementing IPv6 Segment Routing in the Linux Kernel." In: *Proc. ACM ANRW.*

[132] David Lebrun, Mathieu Jadin, François Clad, Clarence Filsfils, and Olivier Bonaventure. "Software Resolved Networks: Rethinking Enterprise Networks with IPv6 Segment Routing." In: *Proc. ACM SOSR.*

[133] Ming-Chieh Lee and Jang-Ping Sheu. "An Efficient Routing Algorithm Based on Segment Routing in Software-defined Networking." In: *Comput. Netw.* 103.C (July 2016), pp. 44–55. ISSN: 1389-1286.

[134] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. "Traffic engineering with forward fault correction." In: *Proc. ACM SIGCOMM.* 2014.

[135] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. "Ensuring Connectivity via Data Plane Mechanisms." In: *Proc. USENIX NSDI.* 2013.

[136] Tamas Lukovszki and Stefan Schmid. "Online Admission Control and Embedding of Service Chains." In: *Proc. SIROCCO.* 2015.

[137] Grzegorz Malewicz, Alexander Russell, and Alexander A. Shvartsman. "Distributed scheduling for disconnected cooperation." In: *Distributed Computing* 18.6 (2005), pp. 409–420.

[138] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, and Christophe Diot. “Characterization of failures in an IP backbone.” In: *Proc. IEEE INFOCOM*. 2004.

[139] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. “Characterization of failures in an operational IP backbone network.” In: *IEEE/ACM Trans. Netw.* 16.4 (2008), pp. 749–762.

[140] Lyle McGeoch and Daniel Sleator. “A Strongly Competitive Randomized Paging Algorithm.” In: *Algorithmica* 6.6 (1991), pp. 816–825.

[141] Jeffrey C Mogul and Lucian Popa. “What we talk about when we talk about cloud network performance.” In: *ACM SIGCOMM Computer Communication Review* 42.5 (2012), pp. 44–48.

[142] Eduardo Moreno, Alejandra Beghelli, and Filippo Cugini. “Traffic engineering in segment routing networks.” In: *Computer Networks* 114 (2017), pp. 23–31.

[143] J. Napper, W. Haeffner, M. Stiemerling, D. R. Lopez, and J. Uttaro. *Service Function Chaining Use Cases in Mobile Networks.* Internet-Draft. IETF, Apr. 2016. 26 pp.

[144] Guyslain Naves and András Sebö. “Multiflow Feasibility: An Annotated Tableau.” In: *Research Trends in Combinatorial Optimization.* Ed. by William J. Cook, László Lovász, and Jens Vygen. Springer, 2008, pp. 261–283.

[145] Srihari Nelakuditi, Sanghwan Lee, Yinzhe Yu, Zhi-Li Zhang, and Chen-Nee Chuah. “Fast local rerouting for handling transient link failures.” In: *IEEE/ACM Trans. Netw* 15.2 (2007), pp. 359–372.

[146] *NetworKit 8.1.* `https://networkit.github.io/`. Last Accessed: 2021-06-28.

[147] *NetworkX 2.5.1.* `networkx.github.io/documentation/networkx-2.5/`. Last Accessed: 2021-06-28.

[148] Mohammad Noormohammadpour and Cauligi S Raghavendra. “Datacenter Traffic Control: Understanding Techniques and Tradeoffs.” In: *IEEE Communications Surveys & Tutorials* 20.2 (2017), pp. 1492–1525.

[149] *NumPy 1.20.3.* `https://numpy.org/devdocs/release/1.20.3-notes.html`. Last Accessed: 2021-06-28.

[150] Eunseuk Oh, Hongsik Choi, and Jong-Seok Kim. “Double-Link Failure Recovery in WDM Optical Torus Networks.” In: *Information Networking, Networking Technologies for Broadband and Mobile Networks, International Conference ICOIN.* 2004.

[151] P. Skoldstrom et al. “Towards Unified Programmability of Cloud and Carrier Infrastructure.” In: *Proc. European Workshop on Software Defined Networking (EWSDN).* 2014.

[152] P. Pan, G. Swallow, and A. Atlas. *Fast Reroute Extensions to RSVP-TE for LSP Tunnels.* RFC 4090. RFC Editor, May 2005.

[153] Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan. "Load-Optimal Local Fast Rerouting for Dependable Networks." In: *Proc. IEEE/IFIP DSN.* 2017.

[154] Michal Pióro, Áron Szentesi, János Harmatos, Alpár Jüttner, Piotr Gajowniczek, and Stanislaw Kozdrowski. "On open shortest path first related network optimisation problems." In: *Perform. Evaluation* 48.1/4 (2002), pp. 201–223.

[155] *Python 3.7.10.* `https://www.python.org/downloads/release/python-3710/`. Last Accessed: 2021-06-28.

[156] R. Hartert et al. "Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks." In: *Proc. ACM SIGCOMM.* 2015.

[157] Harald Räcke. "Optimal hierarchica decompositions for congestion minimization in networks." In: *Proc. 40th ACM Symposium on Theory of Computing (STOC).* 2008, pp. 255–264.

[158] Neil Robertson and Paul D. Seymour. "Graph Minors .XIII. The Disjoint Paths Problem." In: *J. Comb. Theory, Ser. B* 63.1 (1995), pp. 65–110.

[159] Gabriel Robins and Alexander Zelikovsky. "Tighter Bounds for Graph Steiner Tree Approximation." In: *SIAM J. Discrete Math.* 19.1 (2005), pp. 122–134.

[160] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. "Inside the social network's (datacenter) network." In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication.* 2015, pp. 123–137.

[161] Y. Saad and M. H. Schultz. "Topological properties of hypercubes." In: *IEEE Transactions on Computers* 37.7 (July 1988), pp. 867–872. ISSN: 0018-9340.

[162] Yousef Saad and Martin H. Schultz. "Data Communication in Hypercubes." In: *J. Parallel Distrib. Comput.* 6.1 (1989), pp. 115–135.

[163] Stefano Salsano, Luca Veltri, Luca Davoli, Pier Luigi Ventre, and Giuseppe Siracusano. "PMSR—Poor Man's Segment Routing, a minimalistic approach to Segment Routing and a Traffic Engineering use case." In: *Proc. IEEE/IFIP NOMS.* 2016.

[164] Jerome H Saltzer, David P Reed, and David D Clark. "End-to-end arguments in system design." In: *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984), pp. 277–288.

[165] Huzur Saran and Vijay Vazirani. "Finding k Cuts within Twice the Optimal." In: *SIAM Journal on Computing* 24.1 (1995), pp. 101–108.

[166] Stefan Schmid and Jiri Srba. "Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks." In: *Proc. IEEE INFOCOM.* 2018.

[167] *SciPy 1.6.3.* `https://docs.scipy.org/doc/scipy/reference/release.1.6.3.html`. Last Accessed: 2021-06-28.

[168] András Sebö and Anke van Zuylen. "The Salesman's Improved Paths: A 3/2+1/34 Approximation." In: *Proc. FOCS.* 2016.

[169] A. Sgambelluri, F. Paolucci, A. Giorgetti, F. Cugini, and P. Castoldi. "Experimental Demonstration of Segment Routing." In: *Journal of Lightwave Technology* 34.1 (2016), pp. 205–212.

[170] Aman Shaikh, Chris Isett, Albert Greenberg, Matthew Roughan, and Joel Gottlieb. "A case study of OSPF behavior in a large enterprise network." In: *Proc. ACM SIGCOMM Workshop on Internet Measurment.* 2002.

[171] Mike Shand and Stewart Bryant. "IP fast reroute framework." In: (2010).

[172] Lu Shen, Xi Yang, and Byrav Ramamurthy. "Shared risk link group (SRLG)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks." In: *IEEE/ACM Transactions on Networking (ToN)* 13.4 (2005), pp. 918–931.

[173] Abhinav Kumar Singh, Ravindra Singh, and Bikash C. Pal. "Stability Analysis of Networked Control in Smart Grids." In: *IEEE Trans. Smart Grid* 6.1 (2015), pp. 381–390.

[174] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. "Jupiter rising: A decade of CLOS topologies and centralized control in Google's datacenter network." In: *ACM SIGCOMM Computer Communication review* 45.4 (2015), pp. 183–197.

[175] Robert Soulé, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer, and Nate Foster. "Merlin: A Language for Provisioning Network Resources." In: *Proc. 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT).* 2014, pp. 213–226.

[176] Isabelle Stanton. "Streaming Balanced Graph Partitioning Algorithms for Random Graphs." In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms.* SODA 2014. Portland, Oregon, 2014, pp. 1287–1301. ISBN: 9781611973389.

[177] Brent Stephens, Alan L Cox, and Scott Rixner. "Scalable Multi-Failure Fast Failover via Forwarding Table Compression." In: *SOSR. ACM* (2016).

[178] Brent Stephens, Alan L. Cox, and Scott Rixner. "Plinko: Building Provably Resilient Forwarding Tables." In: *Proc. 12th ACM HotNets.* 2013.

[179] János Tapolcai, Balázs Vass, Zalán Heszberger, József Bıró, David Hay, Fernando A Kuipers, and Lajos Rónyai. "A Tractable Stochastic Model of Correlated Link Failures Caused by Disasters." In: *Proc. IEEE INFOCOM.* 2018.

[180] *Topology Zoo.* `http://www.topology-zoo.org/dataset.html`. Last Accessed: 2021-01-05.

[181] Frederic Trate. *Bringing Segment Routing and IPv6 together.* Cisco. Aug. 2016. URL: `https://blogs.cisco.com/sp/bringing-segment-routing-and-ipv6-together`.

[182] George Trimponias, Yan Xiao, Hong Xu, Xiaorui Wu, and Yanhui Geng. "On Traffic Engineering with Segment Routing in SDN based WANs." In: *arXiv preprint arXiv:1703.05907* (2017).

[183] Rita Vachani, Alexander Shulman, Peter Kubat, and Julie Ward. "Multicommodity Flows in Ring Networks." In: *INFORMS Journal on Computing* 8.3 (1996), pp. 235–242.

[184] Emmanouel A. Varvarigos and Dimitri P. Bertsekas. "Performance of hypercube routing schemes with or without buffering." In: *IEEE/ACM Trans. Netw.* 2.3 (1994), pp. 299–311.

[185] Pier Luigi Ventre, Stefano Salsano, Marco Polverini, Antonio Cianfrani, Ahmed Abdelsalam, Clarence Filsfils, Pablo Camarillo, and François Clad. "Segment Routing: A Comprehensive Survey of Research Activities, Standardization Efforts, and Implementation Results." In: *IEEE Commun. Surv. Tutorials* 23.1 (2021), pp. 182–221.

[186] Junling Wang and Srihari Nelakuditi. "IP fast reroute with failure inferencing." In: *Proc. SIGCOMM Workshop on Internet Network Management.* 2007, pp. 268–273.

[187] Ye Wang, Hao Wang, Ajay Mahimkar, Richard Alimi, Yin Zhang, Lili Qiu, and Yang Richard Yang. "R3: resilient routing reconfiguration." In: *Proc. ACM SIGCOMM.* 2010.

[188] Ye Wang, Hao Wang, Ajay Mahimkar, Richard Alimi, Yin Zhang, Lili Qiu, and Yang Richard Yang. "R3: resilient routing reconfiguration." In: *ACM SIGCOMM CCR.* Vol. 40. 4. 2010, pp. 291–302.

[189] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Antony I. T. Rowstron. "Better never than late: meeting deadlines in datacenter networks." In: *Proc. ACM SIGCOMM.* 2011.

[190] Xipeng Xiao, Alan Hannan, Brook Bailey, and Lionel M. Ni. "Traffic engineering with MPLS in the Internet." In: *IEEE Netw.* 14.2 (2000), pp. 28–33.

[191] Baobao Zhang, Jianping Wu, and Jun Bi. "RPFP: IP fast reroute with providing complete protection and without using tunnels." In: *Proc. IWQoS.* 2013.

[192] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghirmalani, Ramesh Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. "Steering: A software-defined networking for inline service chaining." In: *2013 21st IEEE international conference on network protocols (ICNP).* IEEE. 2013, pp. 1–10.