Roma Tre University
Ph.D. in Computer Science and Engineering

The Role of Routing Policies in the Internet: Stability, Security, and Load-Balancing

Marco Chiesa

The Role of Routing Policies in the Internet: Stability, Security, and Load-Balancing

A thesis presented by
Marco Chiesa
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Engineering

Spring 2014

Committee:
Prof. Giuseppe Di Battista (Università degli Studi Roma Tre)

Reviewers:
Prof. Sergey Gorinsky (IMDEA Networks)
Prof. Gordon Wilfong (Bell Labs)

Contents

Introduction

Communication in the Internet relies on both a physical interconnection network and a distributed routing protocol responsible for computing routing paths among all network devices. Data and control traffic are routed along these paths, whose characteristics (e.g., latency, congestion, and stability) can impact the quality of the user experience. Selecting the "best" paths is a non-trivial challenge that puzzled network operators and protocol designers for decades. Moreover, computing these paths in a distributed environment does not alleviate this task.

The high-level objective of network organizations is to achieve the best trade-off between the total cost of ownership (TCO) and the quality of the user experience. To obtain this, network operators attempt to drive routing protocols to compute paths that maximize network performance. However, different routing goals arise when operators focus on *intradomain* or *interdomain* routing. In intradomain routing, i.e., routing traffic within a single network, operators want to maximize the network utilization within their own infrastructure. In interdomain routing, i.e., routing traffic from/to different neighbor networks, operators may want to minimize the cost of sending traffic outside their network, where the cost depends on economic agreements, or to avoid to transit traffic without getting any revenue. To achieve these different goals, a network runs both an intradomain and an interdomain routing protocol. Each routing protocol supports a *routing policy* language that network operators use to control what paths are selected. Routing policy languages offered by different routing protocols also differ in terms of expressiveness. Roughly speaking, the more expressive a routing policy language, the wider the set of operations that an operator can perform to influence the selection of routing paths. This difference is particularly evident between interdomain and intradomain routing protocols.

Traditional interdomain routing protocols compute paths "hop-by-hop" based

on the path-vector paradigm, i.e., each router selects one of its available routes from its neighbors and propagates it to some of its neighbors. When this mechanism is combined with a very expressive policy language, predicting how routes propagate throughout a network becomes a difficult task. Failure to predict the routing outcome means impossibility to perform network debugging, routing configuration migrations, and traffic-engineering without causing service disruptions.

This thesis makes three important contributions that answer different questions about routing protocols. The first two questions relate to the process of computing a set of routing paths in interdomain routing protocols, hence dealing with route propagations issues. The third question deals with the problem of selecting routing paths (i.e., traffic-engineering) that maximize network utilization in intradomain routing protocols.

We focus on destination-based routing, where a packet is forwarded based on its destination fields, and study the following problems by leveraging formal tools for analyzing computational complexity.

Question 1 *Given a path-vector-based routing protocol and a set of router configurations, does the routing protocol compute a set of routing paths in finite time?*

Extensive literature showed that the standard de-facto interdomain routing protocol adopted in the Internet (i.e., the Border Gateway Protocol, BGP) may fail to perform this task in finite time, i.e., it may indefinitely continue to change the computed routing paths, causing routing instabilities. This unwanted behavior arises when conflicting routing policies cannot be simultaneously satisfied. In particular, the BGP routing policy language is so expressive that it allows operators to take routing decisions autonomously without any guarantee that they can all be simultaneously satisfied.

In Chapter 2, we show that the most interesting problems related to Question 1 are computationally hard even if policies are restricted to be neighbor-based. Our findings suggest that the computational intractability of stability is an intrinsic property of policy-based path vector routing protocols that allow policies to be specified in complete autonomy. Stimulated by this negative result, we investigate whether stability problems can be made tractable by sacrificing the expressive power of policy configurations, while preserving organizational autonomy. We show that computational tractability of BGP stability can be achieved without sacrifying neighbor-based ranking by filtering routes in a very specific manner.

In Chapter 3, we continue our investigation on (Question 1), providing the main insight into BGP intractability. We unveil an intriguing analogy between logic circuits and BGP network configurations that leads to a surprising result: "BGP is Turing-Complete". Put another way, analyzing BGP routing configurations is as hard as debugging any computer program, which in most cases is an unfeasible feat. To achieve this result, it suffices to show how elementary BGP configurations can replicate the behavior of AND, OR, and NOT gates. In particular, the logic signals 1 and 0 are mapped to the absence or presence of a BGP route, respectively. Two elements are nonetheless missing to build a Turing machine: the ability to store information in the BGP configuration (i.e., a memory) and a clock. These elements are built by exploiting two peculiar real-world configurations that have troubled network operators and the IETF community for over a decade, showing that some real-world BGP configurations are capable of storing information in a very similar way electronic flip-flops do. BGP is therefore powerful enough to encode logic circuits of arbitrary complexity and, as such, the BGP routing system constitutes the "*biggest computer*" ever made!

In a second part of the thesis (Chapter 4), we continue this study on BGP route propagation answering the following intriguing question.

Question 2 *How can a network operator trigger global routing changes in the Internet?*

The foremost motivation for this study is inspired by a famous YouTube hijacking attack, in which a small network hijacked a large portion of the traffic directed to YouTube by simply announcing that it owned the YouTube IP subnet. This announcement propagated throughout the Internet with unpredictable effects on global routing. Eventually, it caused a world-wide service disruption for YouTube users. Although the attack was fairly simple, past work showed that more clever attack strategies exist.

We believe that Question 2 is intriguing from at least two points of view: the one of a hacker who wants to hijack a communication flow between two different parties and, conversely, the one of a network operator who wants to eventually plan a counterattack strategy to restore the hijacked IP prefixes. In both cases, only local modifications to routing announcements are permitted.

Computing a hijacking strategy is tricky since route propagation, as shown in Chapter 2, is itself unpredictable. Four results are shown in Chapter 4, where it is assumed that routing policies are specified according to typical economic agreements, which limit their expressiveness and autonomy. First, we

present an efficient algorithm that optimally computes attacking strategies for hijacking traffic. Second, we show that adding cryptographic security into the routing protocol makes it computationally harder to understand if an attacking strategy exists. This result is not an obvious consequence of using cryptography. Third, a hacker cannot create a global routing instability: Routing is always guaranteed to converge to a stable state. Finally, we show a strategy to hijack traffic without creating a "black-hole", i.e., the traffic is forwarded to the correct destination after it is hijacked.

In the last part of the thesis, we tackle the problem of maximizing network utilization. While interdomain routing protocols offer expressive routing policies to support economic agreements, intradomain routing protocols do not necessarily do it. As a matter of fact, in traditional routing protocols like OSPF and RIP, operators can only have an indirect control of routing paths. Namely, they set link weights in the network and the routing protocol routes traffic along shortest paths. A widespread technique to achieve high network utilization is to equally split traffic when multiple shortest paths are available. However, computing the optimal routing paths is difficult: in order to deal with modern-day challenges, networks are overprovisioned (e.g., in wide area networks only a poor 30% of network resources are utilized). This motivates our last question.

Question 3 *Can we compute routing paths that maximize network utilization using equal-split load-balancers?*

In Chapter 5, we contribute to this question with a striking result: it is hard to distinguish in polynomial time whether two routers can exchange a data flow of size 1 or x, for any constant $x > 0$. This result applies to a broad range of definitions of "network utilization": minimizing the most congested edge, where congestion is modeled as the ratio between the amount of data routed through a link and its bandwidth; maximizing the throughput of the network; minimizing the sum of link costs, where the cost can be any convex function of the flows routed through a link.

This negative result may be alleviated when the network topology is constrained. We design an efficient algorithm for recently proposed data-center topologies where the computed paths can be expressed as shortest-paths.

This thesis is organized as follows. A high-level overview of the main routing protocols used in the Internet (i.e., BGP and OSPF) and a description of a BGP model are introduced in Chapter 1. Chapter 2 deals with the complexity of BGP convergence. Chapter 3 shows an intriguing mapping between logic gates

and BGP configurations. Chapter 4 studies the impact of local routing changes on global routing from the point of view of a malicious AS. Chapter 5 tackles the problem of maximizing network utilization within a network. We draw conclusions and directions for future research in the last part of the thesis.

Chapter 1

Internet Routing Policies

The Internet has been growing and evolving for decades. Nowadays, it consists of thousands of interconnected networks owned by different independent organizations, called Autonomous Systems (ASes). The number of Internet devices that are connected to these networks exceeded 8 billions. Communication between these devices happens by means of packets that are forwarded throughout the network along routing paths. Routing protocols are responsible for computing these paths and *routers* are the physical network devices that handle data and control packets. For each packet in transit, a router uses a forwarding function to determine where the packet must be forwarded. The specific forwarding function that is installed in a router is configured by a routing protocol that performs the following actions: It computes the "best" routing paths and it translates them into a set of forwarding functions that are installed into each router that appears in these paths. These actions can be performed both in a centralized or in a distributed way and both simultaneously or sequentially.

Selecting the "best" routes is far from being an easy task. Routing protocols can be configured via a *routing policy language*, which defines a set of constructs that can be used to influence the routing decision process. Network organizations control routing paths by specifying a set of routing policies that is used to rank paths according to a certain metric. A routing policy can be more or less expressive. The higher the expressivity, the more fine-grained the control on the routing paths.

Computing paths within a single network domain poses different challenges from the task of computing paths that traverse different networks. In the for-

mer case, the entire topology is known, the traffic matrix can be sometime estimated, and network operators have (almost) complete control of their infrastructure. In the latter case, both topology, routing policies, and traffic demands are not known (except for this information that is explicitly shared among networks). Moreover, operators do not have any control outside their domain. For these reasons, computing routes within a single network requires a different routing protocol from the one that is used for computing routes across different domains. The first task is handled by an *intradomain* routing protocol, while the second one is handled by an *interdomain* routing protocol.

Expressiveness. Nowadays, interdomain routing protocols must cope with the need of network operators to control their routing. Economic agreements between ASes are established with the purpose of exchanging traffic data among different networks. These economic agreements may be very different, depending on the nature of each AS. For instance, a *stub* AS, which has a single connection to another AS, does not have many alternatives on where to route outgoing traffic. On the contrary, a *multihomed* AS, which has at least two connections with different ASes, has several choices: using one connection as a primary route and the second one as a backup route; load-balancing its outgoing traffic between these two ASes; prevent transit traffic between the two neighbors from traversing its network. Finally, a *transit* AS, which has at least two connection with other ASes and want to transit traffic only between specific pairs of its neighbors, requires a more complex routing protocol to satisfy its requirements than a stub or multihomed AS, e.g. accept to send traffic to a specific neighbor only if it is received from those neighbors that are paying for this service. The consequences of supporting such expressive routing policies are studied in Chapter 2, Chapter 3, and Chapter 4.

In intradomain routing protocols, achieving maximum network utilization is the most simple objective function for reducing infrastructure costs. In fact, the higher the network utilization, the smaller the network infrastructure. This translates into less expensive network devices and, consequently, a smaller power consumption. Traditional intradomain routing protocols offer to network operators a limited number of configurable parameters (e.g., setting link weights and routing along shortest paths based only on the destination address). With the ever growing increase of Internet traffic, network operators are struggling in an effort to deal with a new set of modern-day challenges by tweaking these protocols. As a consequence, the costs of managing a network are far from being optimal: in order to deal with sporadic peaks of traffic, networks are highly overprovisioned. In wide area networks only a poor 30%

of network resources are utilized [JKM+13]. In datacenter networks, recent works advocate the need for more expressive routing protocols, showing that network utilization can be improved by 50% if certain specific traffic flows are routed according to both the source and destination address (flow-based routing) [AFRR+10][BAAZ11]. Intradomain routing protocols are discussed in Chapter 5, where we consider both destination-based and flow-based routing protocols and paths can be be constrained along shortest paths or not.

In Section 1.1, we introduce the Border Gateway Protocol (BGP), i.e., the standard de-facto interdomain routing protocol. In Section 1.2, we describe Open Shortest Path first (OSPF), i.e. a widely adopted intradomain routing protocol based on shortest-path routing. In Section 1.3, we present the standard model for BGP, i.e. SPP [GSW02], which can be adapted for modeling shortest-path protocols. In Section 1.4, we provide an overview of the computational complexity notions that are widely used throughout the thesis. The reader that is familiar with BGP, OSPF and computational complexity should read only Section 1.3, where a model for BGP is introduced.

1.1 Border Gateway Protocol (BGP)

The Border Gateway Protocol is the standard de-facto interdomain routing protocol that that glues the Internet together. It is described in RFC4271 [RLH06], RFC4276 [HR06], and RFC4277 [MP06].

Routing information is exchanged by establishing a BGP *session* between two routers, called BGP *speakers*. A session is a TCP connection that is used to exchange *BGP messages*. BGP messages carry reachability information and are discussed later in this section. BGP speakers can be located within the same AS or in different ASes. Each BGP speaker belongs to a unique AS, which is identified with an integer, that is, the Autonomous System Number (ASN).

BGP has two modes of operation. The *external* BGP (eBGP) is used to exchange information among BGP speakers belonging to different ASes. If there exists more than one eBGP speaker in the same AS, then *internal* BGP (iBGP) is required. We stress the fact that iBGP is not an intradomain routing protocol. It is responsible for disseminating routing information learnt from eBGP speakers to all the eBGP speakers within the same AS. It does not compute routing paths within the same AS, which is the goal of an intradomain routing protocol. Border routers run both iBGP and eBGP sessions (see Fig. 1.1). Internal BGP does not require that iBGP speakers are directly connected by

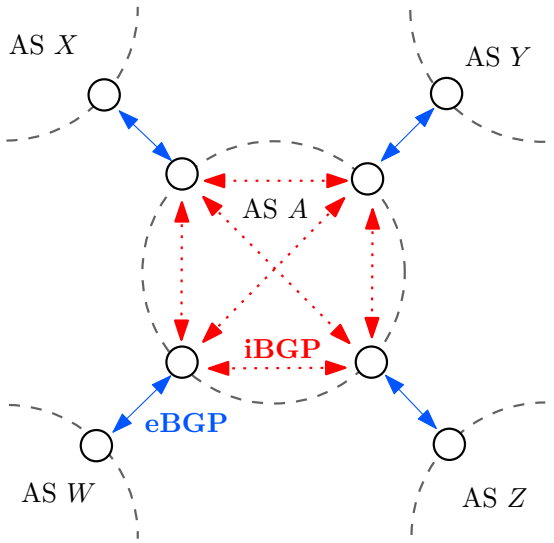

Figure 1.1: High-level overview of eBGP and iBGP. Dashed gray circles represent AS domains. BGP routers are depicted by non-dashed circles. Blue arrows (red arrows) represent eBGP (iBGP) sessions. Internal topology of each domain is not shown.

a physical link. As we already pointed out, an intradomain routing protocol is used to keep connectivity among iBGP routers. However, since the number of iBGP sessions would be quadratical with respect to the number of eBGP speakers, techniques known as Route Reflector [BCC06] or Confederations [TMS07] are typically adopted. Conversely, two eBGP speakers must be interconnected by a physical link as there is not any intradomain routing protocol running between two different domain networks.

Path attributes of BGP messages

BGP defines the type of messages that are exchanged in a session between two BGP speakers. Each message carries reachability information for a set of destinations. Destinations are IP subnets identified by IP prefixes. Each prefix has a set of *attributes* possibly configured by each AS. There are mainly two kind of messages: *announcements* and *withdrawals*. Announcement messages are sent

by an AS that is willing to forward traffic from the receiver of the announcement to the set of destinations specified in the announcement. Receivers of withdrawal messages are notified that the set of destinations contained in the announcement are no longer reachable passing by the sender of the withdrawal message.

It follows a list of attributes. The first three are relevant for the purposes of this thesis.

- `AS-path`. It is a well-known mandatory attribute. It contains the sequence of ASes that forwarded that announcement. It corresponds to the route to the destination. When a BGP speaker announces a route to an external peer, the receiver updates the `AS-path` attribute by prepending its ASN. On the contrary, when a BGP speaker announces a route to an internal peer, the receiver shall not modify the `AS-path` attribute associated with this route, i.e., it does not prepend its ASN. This field is used to avert the count-to-infinity problem as in any path-vector routing protocol.

- `local-preference`. It is a well-known attribute. It allows a BGP router to indicate the relative degree of preference that is locally associated with the route contained in the BGP update. This attribute enables administrators to specify powerful routing policies in the path selection process.

- `next-hop`. It is a well-known mandatory attribute that represents the IP address to which forward traffic in order to reach the associated destination. That address belongs to a different AS.

- `origin`. It is a well-known mandatory attribute. It is set by the originator of the announcement and signals how the prefix has been injected in BGP (e.g., IGP, EGP). Its value should not be changed by any other speaker.

- `multi-exit-discriminator`. It is an optional non-transitive attribute. It is used in order to rank different outgoing links that interconnect two different ASes. It is used for inbound traffic-engineering.

- `atomic-aggregate`. It is a well-known discretionary attribute. It is used to indicate that the route contained in the BGP message corresponds to the aggregation of multiple contiguous IP prefixes that share the same attribute values.

- `aggregator`. It is an optional transitive attribute that indicates the AS number and the IP address of the last BGP router that aggregated the prefix.
- `community`. It is an optional transitive attribute that can be used to add tags to announcements. Receivers can interpret tags to perform several actions.

BGP Dynamics

An AS announces to its neighboring ASes of its existence and which destinations it owns. When an AS receives an announcement, it appends its ASN into the `AS-path` field of the message and propagates it according to its economic agreements with its neighboring ASes. Loop detection is performed by checking the content of the `AS-path` attribute when a BGP announcement is received. Namely, an AS discards a BGP announcement whereas the message already contains its ASN in the `AS-path`. Each network has only a partial view of the topology. In particular, an AS can only see routes announced to it by its neighbors.

We now define what data structures BGP keeps in memory and how they are used during the reception of a BGP message. Each BGP speaker B store a Routing Information Base (RIB), which consists of three distinct data structures:

- *Adj-RIBs-In.* It stores routing information learnt from BGP messages that has been advertised to B by its peers. Whenever a BGP message is received, it is matched against a set of local *import filter* policies, which contains also loop-detection mechanisms. If the message is not discarded, then it is stored in the Adj-RIBs-In and if there already exists an entry for the same prefix and from the same neighbor, the older message is overwritten.
- *Loc-RIB.* It contains the output of the path selection process, that is, for each prefix, the application of the local policies on the paths stored in the Adj-RIBs-In is performed. The decision process usually consists of seven steps that are shown in Table 1.1. The decision process starts from the step 1. When the results of a single step contains exactly one path, the process halts, otherwise the following step is used to break ties.
- *Adj-RIBs-Out.* When the speaker selects a new best path for a certain prefix, it must announce it to its neighbors according to its *export filter*

Step	Criterion
1	Selects the route with the highest `local-preference`
2	Selects the route with the lower `AS-path` length
3	Prefers the route with `origin` IGP over BGP and `origin` BGP over others
4	Among the routes received from the same AS neighbor, discard those having higher `multi-exit-discriminator` than the lowest
5	Prefer routes learnt via eBGP to those learned via iBGP
6	Prefer routes with lower IGP metric to the egress point
7	Prefer the route announced by the BGP router with the lowest router-id (i.e., IP address)

Table 1.1: Steps in the path selection process of BGP.

policies. The Adj-RIBs-Out stores the BGP messages that are scheduled to be sent to the neighbors of the speaker.

Ranking and Filtering

Path ranking. The `local-preference` attribute is the most powerful tool for ranking paths. It is typically used to rank routes according to their next-hops. It is set when a route is received from an eBGP session and its value is not changed when the message is propagated in iBGP. Routes with a higher value of `local-preference` are preferred over routes with a smaller value of `local-preference`. As an example, consider again Figure 1.1. Suppose that AS A purchases transit services from both X and Y, while it offers transit services to both W and Z. When A receives four different routes for a specific IP prefix from all of its neighbors, it prefers to forward traffic to those neighbors where it does not have to pay (i.e., probably its customer W and Z). For this reason, AS A set a higher `local-preference` to those announcements received by W and Z and a lower `local-preference` to those received by X and Y.

Path filtering. Operators can filter paths during both the import and the export phase. During the import phase, an operator may decide to discard all the routes that traverse a specific AS. During the export phase, an operator may prevent some routes to be exported to some of its neighbors. Consider the following example based on Figure 1.1. Suppose that AS A is a multihomed domain that purchases transit services from both X and Y. Since this domain

does not earn money from its two providers X and Y, it does not want to overload its network with transit traffic between X and Y. To avoid this, A can configure its import and export filters in such a way that it does not export to B any routes that it learnt from A, and viceversa.

1.2 Open Shortest Path First (OSPF)

Open Shortest Path First (OSPF) [Moy98] is an intradomain routing protocol that is used to compute all-pairs shortest-paths between routers based on configurable static link weights (where a link's weight specifies its distance in the shortest-path computation). It is a link-state routing protocol. Namely, each router announces local information (e.g., its directly-connected neighbors and link weights values) to every other router in the network, which, in turn, use them to reconstruct a map of the connectivity of the network and to compute routing paths. These routing paths are computing by running Dijkstra's algorithm for finding shortest paths. Since all routers have the same view of the networks, forwarding loops are prevented, unless during transient states (e.g., link failures). To exploit multiple paths, OSPF is equipped with the Equal-Cost Multi Path (ECMP) feature. This feature is introduced to exploit shortest-path diversity in a network. It enables routers to "split" traffic among different multiple shortest-paths via per-flow static hashing [CWZ00].

Network operators configure OSPF using a routing policy language that limits the expressiveness as follows: (1) traffic from a source to a destination in the network can only flow along the shortest paths between them (for the given configuration of link weights); and (2) traffic can only be split between multiple shortest paths (if multiple shortest paths exist) in a very specific manner (i.e., per-flow hashing).

It is worth to observe that OSPF provides a less expressive routing policy language than BGP. In fact, Step 2 in the BGP routing process (Table 1.1) is a shortest path metric based on the number of ASes contained in the `AS-path` field. To simulate link weights, an AS can prepend its own ASN several times. In Chapter 5, we show that even in OSPF, finding an "optimal" set of routes is computationally intractable. The same result applies to BGP as well.

1.3 A Model for Routing Policies

In [GW99] a BGP model is proposed where policies are described by means of functions that implement import and export filters, similarly to real-world

BGP configuration languages.

This section describes the well-known *Stable Paths Problem* (SPP) formalism [GSW02] and defines k-SPP, a variation of SPP which is suitable to study how policy expressiveness impacts the computational complexity of stability problems. Moreover, the class of models of k-SPP with $k = 3$ has been shown to capture the so-called Local-Transit-Policies, a very common router configuration language where policies are functions only of the ingress and the egress points.

The Standard Path Problem Model (SPP)

SPP models a BGP network as an undirected graph $G = (V, E)$, where vertices $V = \{d, 1, \ldots, n\}$ represent ASes and edges in E correspond to BGP peerings between ASes. Vertex d is a special vertex in that it is the destination every other vertex attempts to establish a path to. Since different destinations are independently handled by BGP [RLH06], d is assumed to be the only destination, without loss of generality. A path P is a sequence of $k + 1$ vertices $P = (v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$, $v_i \in V$, such that $(v_i, v_{i-1}) \in E$ for $i = 1, \ldots, k$. Vertex v_{k-1} is the *next hop* of v_k in P. For $k = 0$ we obtain the trivial path (v_0) consisting of vertex v_0 alone. The empty path represents inability to reach the destination and is denoted by ϵ. The *concatenation* of two nonempty paths $P = (v_k\ v_{k-1}\ \ldots\ v_i)$, $k \geq i$, and $Q = (v_i\ v_{i-1}\ \ldots\ v_0)$, $i \geq 0$, denoted as PQ, is the path $(v_k\ v_{k-1}\ \ldots\ v_i\ v_{i-1}\ \ldots\ v_0)$. We assume that $P\epsilon = \epsilon P = \epsilon$, that is, the empty path can never extend or be extended by other paths.

BGP policies are modeled by explicitly listing and ranking all permitted paths. More precisely, for each vertex $u \in V$ is assigned a set of *permitted paths* $\mathcal{P}^u$ which represent the paths that u can use to reach d. All the paths in $\mathcal{P}^u$ are simple (i.e., without repeated vertices), start from u and end in d. The empty path, representing unreachability of d, is permitted at each vertex $u \neq d$. Vertex d can reach itself only directly, hence $\mathcal{P}^d = \{(d)\}$. Let $\mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u$. For each $u \in V$, a *ranking function* $\lambda^u : \mathcal{P}^u \to \mathbb{N}$ determines the relative level of preference $\lambda^u(P)$ assigned by u to path P. If $P_1, P_2 \in \mathcal{P}^u$ and $\lambda^u(P_2) < \lambda^u(P_1)$, then P_2 is *preferred* over P_1. Let $\Lambda = \{\lambda^u | u \in V\}$.

The following conditions hold on permitted paths of each vertex $u \in V - \{d\}$:

1. $\forall P \in \mathcal{P}^u, P \neq \epsilon$: $\lambda^u(P) < \lambda^u(\epsilon)$ (unreachability of d is the last resort);
2. $\forall P_1, P_2 \in \mathcal{P}^u, P_1 \neq P_2 : \lambda^u(P_1) = \lambda^u(P_2) \Rightarrow P_1 = (u\ v)P_1', P_2 = (u\ v)P_2'$, (strict ranking is assumed on all paths but those with the same next hop).

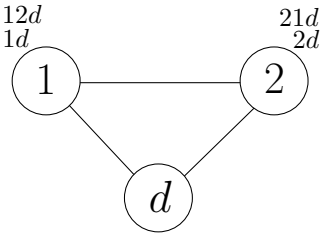

Figure 1.2: An SPP instance known in literature as DISAGREE [GW99].

An instance S of SPP is a triple $(G, \mathcal{P}, \Lambda)$.

An example of SPP instance is shown in Fig. 1.2, using the same graphical convention as in [GSW02]. The list beside each vertex u represents the paths in $\mathcal{P}^u$ sorted by increasing values of λ^u. For example, vertex 2 can use paths in $\mathcal{P}^2 = \{(2\ 1\ d), (2\ d)\}$ to reach d and prefers $(2\ 1\ d)$. The empty path and $\mathcal{P}^d$ are omitted for brevity.

A *path assignment* is a function π that maps each vertex $v \in V$ to a path $\pi(v) \in \mathcal{P}^v$, representing the fact that the BGP process running at vertex v is selecting $\pi(v)$ as its preferred path to reach the destination. We always have $\pi(d) = (d)$.

BGP dynamics are modeled by a distributed algorithm called *Simple Path Vector Protocol* (SPVP) [GSW02], which computes a path assignment π_t at each iteration t. Since we consider discrete time, iterations and time are interchangeable concepts. SPVP works as follows (details can be found in [GSW02]). Vertex d keeps announcing its presence to its neighbors. Every other vertex u collects announcements from its neighbors and stores announcements in a data structure called rib_in_t. In particular, $\text{rib\_in}_t(u \Leftarrow v)$ contains the latest announcement from neighbor v. If the last announcement received from a neighbor v is discarded, then $\text{rib\_in}_t(u \Leftarrow v) = \epsilon$. Thus, u can select a path in the following set:

$$\text{choices}_t(u) = \begin{cases} \{(u\ v)\ \text{rib\_in}_t(u \Leftarrow v)\} & \text{if } u \neq d \\ \{(d)\} & \text{if } u = d \end{cases}$$

We say that a path p is *available* at vertex u at time t if $p \in \text{choices}_t(u)$.

Let W be the set of available paths at u. At this point, u selects the best ranked path in W according to its ranking function λ^u:

$$\text{best}(W,u) = \begin{cases} \arg\min_{P \in W} \lambda^u(P) & \text{if } W \neq \oslash \\ \epsilon & \text{if } W = \oslash \end{cases}$$

If this operation updated u's selected path, then u sends announcements to all its neighbors. Notice that, at any time t, SPVP computes a path assignment π_t such that each vertex selects the best available path.

Given an SPP instance, we say that π_t is a *stable path assignment* if, $\forall u \in V$: $\pi_t(u) = \text{best}(\text{choices}_t(u), u)$, that is, every vertex has settled to the best possible choice and cannot switch to a better ranked alternative.

An activation sequence [GW00] $\sigma = (A_1 \ \ldots \ A_i \ \ldots)$ is a (possibly infinite) sequence where A_t is a set representing the announcements that are received by vertices at time t. Set A_t contains an ordered pair $(u,v)|(u,v) \in E$ for each vertex v that processes a message from u at time t. Simultaneous activations are allowed. We say that edge (u,v) is activated at time t. Since the delay introduced by edges is finite, announcements are eventually delivered. As an example, consider Figure 1.2. Before SPVP starts working, we assume $best_t(u) = \epsilon$ for all vertices but d. A possible activation sequence is $\sigma = (A_1 \ A_2)$ with $A_1 = (d,1),(d,2)$, and $A_2 = (1,2),(2,1)$. Namely, at time $t = 1$ vertices 1 and 2 simultaneously receive an announcement from d, stating that vertex d is directly reachable. Hence, vertex 2 inserts into its rib-in path $(2\ d)$, i.e., $rib - in_1(2 \Leftarrow d) = (2\ d)$. Similarly, for vertex 1 $\text{rib\_in}_1(1 \Leftarrow d) = (1\ d)$. Since vertices 1 and 2 have no other alternatives, they both select the direct path as the best route to d, i.e., $best_1(1) = (1\ d)$ and $best_1(2) = (2\ d)$. Because of A_2 , at time $t = 2$ vertex 3 also receives an announcement from d, sets $\text{rib\_in}_2(2 \Leftarrow 1) = (2\ 1\ d)$, and computes its best path $best_2(2) = (2\ 1\ d)$. At the same time, vertex 1 receives an announcement from 2, which advertises path $(2\ d)$. Hence, vertex 2 sets $\text{rib\_in}_2(1 \Leftarrow 2) = (1\ 2\ d)$, and computes a new best path $best_2(1) = (1\ 2\ d)$.

An activation sequence is *fair* if any edge (u,v) is eventually activated after u has sent a message to v. In the following, we consider only fair activation sequences.

As shown in [Cit10], simplifying the definition of activation sequences (e.g. by activating vertices instead of edges or by not allowing simultaneous activations) leads to inability to model a subset of the possible routing oscillations. It has been shown [GSW99] that, possibly depending on the specific activation sequence, the SPVP algorithm might oscillate indefinitely, never converging to a stable state. An SPP instance S is *safe* [GSW99] if SPVP is guaranteed to eventually reach a stable path assignment on S for any fair activation sequence.

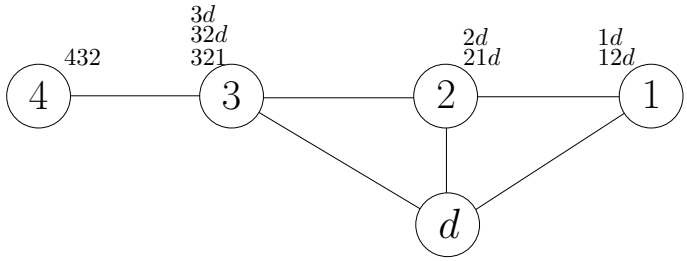

Figure 1.3: An instance of the 3-SPP model.

A *state* s is the representation of all the the RIBs at each vertex at a certain time. The *evaluation digraph* of an SPP instance $S = ((V, E), P, \Lambda)$, denoted $Eval(S)$, is a labeled directed graph having one node for each possible state. There exists an edge e between two vertices s_1 and s_2 only if there exists a set of edges $\bar{A} \subseteq E$ such that if they are simultaneously activated in the state s_1, the resulting state is s_2. As in previous work [SSZ09], we refer to the special state in which all the RIBs are empty as the 0-state.

3-SPP and k-SPP

SPP can model every possible BGP policy specification. However, since it requires explicit listing and ranking of all paths, it is mostly a theoretical model. In fact, network operators configure BGP policies without knowing the entire network topology. Also, the *size* of an SPP instance is bound to the size of $\mathcal{P}$, which can be exponential in $|V|$.

Several models have been proposed in alternative to SPP, either adopting a more realistic specification of policies (e.g., [GW99]) or limiting the expressiveness of the policies that can be expressed in the model (e.g., [JR04]). However, these models either have the same expressive power of SPP or have important limitations.

We now describe 3-SPP, a variant of SPP in which vertices are forced to rank and filter announcements on the basis of the first 3 distinct hops in the path. This model allows ASes to specify Local Transit policies [GGSS09], that is, policies in which ASes define filters based on neighbor pairs (e.g., paths received from neighbor x should not be exported to neighbor y).

We now formally define the 3-SPP model. Let $G = (V, E)$ be defined as for standard SPP instances. Each vertex $u \in V$, with $u \neq d$, is assigned a

set of *permitted path fragments* $\tilde{\mathcal{P}}^u$ such that $(u\ d)$ can be in $\tilde{\mathcal{P}}^u$ if $(u,d) \in E$ and paths $(u\ v\ w)$ can be in $\tilde{\mathcal{P}}^u$ if u, v, and w are distinct vertices in V and $(u,v),(v,w) \in E$. The only permitted path fragment at vertex d is $\tilde{\mathcal{P}}^d = \{(d)\}$. To reach d, a vertex $u \in V - \{d\}$ can use any path $P = (u\ v_1\ \ldots\ v_n\ d)$, starting with a fragment in $\tilde{\mathcal{P}}^u$ and obtained by concatenating any permitted fragment at each vertex v_i, with $i = 1,\ldots,n$. Path fragments contain exactly 3 vertices except for the case of d and of its neighbors, which can reach d directly. Let $\tilde{\mathcal{P}} = \bigcup_{u \in V} \tilde{\mathcal{P}}^u$.

Each vertex $u \in V - \{d\}$ ranks path fragments in $\tilde{\mathcal{P}}^u$ according to a function $\tilde{\lambda}^u : \tilde{\mathcal{P}}^u \to \mathbb{N}$ which assigns a level of preference to paths starting with a fragment in $\tilde{\mathcal{P}}^u$. Namely, if $\tilde{\lambda}^u((u\ v\ w)) < \tilde{\lambda}^u((u\ x\ y))$ then any path starting with $(u\ v\ w)$ is preferred to any path starting with $(u\ x\ y)$.

Similarly to the SPP model, the empty path is always permitted, i.e., $\epsilon \in \tilde{\mathcal{P}}^u$, $\forall u \in V - \{d\}$, and unreachability is the last resort, i.e., $\forall P \in \tilde{\mathcal{P}}^u, P \neq \epsilon$: $\tilde{\lambda}^u(P) < \tilde{\lambda}^u(\epsilon)$.

Differently from the SPP model, two path fragments can have the same rank even if they have a different next hop. Moreover, paths through the same neighbor always have the same rank, i.e., if $(u\ v\ w)$ and $(u\ v\ z)$ are two path fragments in $\tilde{\mathcal{P}}^u$ then $\tilde{\lambda}^u((u\ v\ w)) = \tilde{\lambda}^u((u\ v\ z))$. Any deterministic criterion (e.g., shortest path) can be used to break ties.

An instance $\tilde{S}$ of 3-SPP is a triple $(G, \tilde{\mathcal{P}}, \tilde{\Lambda})$. An example 3-SPP instance is depicted in Fig. 1.3 using a graphical convention analogous to the one used in [GSW02]. The list beside each vertex u represents the permitted path fragments in $\tilde{\mathcal{P}}^u$ sorted by increasing values of $\tilde{\lambda}^u$. For example, vertex 2 can use path fragments in $\tilde{\mathcal{P}}^2 = \{(2\ 1\ d), (2\ d)\}$ to reach d and prefers $(2\ d)$. The empty path and $\tilde{\mathcal{P}}^d$ are omitted for brevity. Vertex 3 decides not to propagate the path received from d to 2, and permitted paths fragments at vertex 2 result from filtering action performed by 3 and ranking configured at 2. In Figure 1.3 we observe that path fragment 432 at vertex 4 models two distinct paths from 4 to d, namely $432d$ and $4321d$, that have the same rank.

The 3-SPP model can be generalized to the k-SPP model, where permitted path fragments defined at each vertex contain k hops. The number of path fragments at each vertex is $O(n^{k-1})$, where $n = |V|$, hence the size of an instance of k-SPP is $O(n^k)$. It is easy to verify that, given a specific tie break criterion, an instance of k-SPP can be uniquely translated to an instance of SPP (e.g., by concatenating path fragments to generate permitted paths at each node), while the opposite is in general not true. In other words, k-SPP allows us to trade policy expressiveness for policy succinctness.

A natural generalization of next-hop based preferences are class based routing policies [JR04]. The ranking (filtering) component of a routing policy is class based if function λ^v at vertex v ranks P (filters) paths according to a total order defined on a partition of v's neighbors into classes. That is, vertex v prefers paths announced by neighbors in class C_j to paths announced by neighbors in class C_k , $k > j$, and does not accept paths from neighbors in specific classes. The set of available classes is a network-wide constant. Observe that customer-provider instances are a special case of class based instances where each vertex partitions its neighbors into 3 classes: customers, peers, and providers. In this setting, a polynomial time algorithm is given to check whether the structure of the classes can lead to specific BGP policies in which oscillations are possible.

The 3-SPP model is similar to the one used in [JR04] in that it also limits the expressiveness of BGP policies, however it is more general since it allows each AS to preserve its autonomy.

Shortest-Path and 3-SPP

The 3-SPP model is capable of modeling shortest-path routing protocols. In fact, when two path fragments have the same ranking functions, ties are broken by shortest-paths criterium (see Table 1.1), which corresponds to selecting routes that contain fewer ASN in the `AS-path` field. Moreover, if each link has a weight w, then each AS prepends its ASN w times.

1.4 Computational Complexity Overview

This section provides a brief overview of the main computational complexity notions that are required to understand this thesis. For a more detailed introduction to this topic, the reader should refer to [Sip97], [GJ79], and [Pap94].

Computational complexity theory classifies problems according to their difficulty. To do this, it requires a model of computation that captures the amount of resources that are needed to solve a problem, such as time and space. The more the resources utilized, the harder the problem. We consider the standard computational model of a *Turing Machine* (TM), i.e., a simple device that reads and writes symbols on a infinite tape according to a table of rules. A TM can be adapted to compute any computer algorithm. The time resource is represented by the number of steps (i.e., write/read operations) that a TM must perform to solve an instance of a problem. The space resource is represented by the number of cells of the tape that are utilized. A *non-deterministic*

Turing Machine (NTM) is a TM that at each step can take a non-deterministic decision, e.g., write on a cell two different values. When this happens, the computation is split in two branches that continue to solve the problem independently. It suffices that a single branch of the computation solves a problem for the NTM to solve the problem. In that case, the time and space resources used by the NTM correspond to the time and space resourced used by the branch that solved the problem. Although a NTM does not model a real-world machine, it has a fundamental role in the classification of the hardness of the problems.

A *decision* problem is a yes-or-no question on an infinite set of input instances. A *computational complexity class* contains problems that can "asymptotically" be solved with the same amount of resources with respect to the size of the input. For the purpose of this thesis, we consider the following computational complexity classes. *P* contains all the decision problems that can be solved by a TM in polynomial time. *NP* contains all decision problems that can be solved by a NTM in polynomial time. *PSPACE* contains all the decision problems that can be solved by a NTM in polynomial space. *NPSPACE* contains all the decision problems such that their complement can be solved by a NTM in polynomial space. *coNP* contains all the decision problems such that their complement is in NP. *coNPSPACE* contains all the decision problems such that their complement is in NPSPACE.

The following relationships among these classes hold:

$$\begin{gathered} \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \\ \mathrm{P} \subseteq \mathrm{coNP} \subseteq \mathrm{PSPACE} \\ \mathrm{PSPACE} = \mathrm{NPSPACE} = \mathrm{coNPSPACE}. \end{gathered}$$

The last relationship is known as the Savitch's Theorem [Sip97]. All problems not contained in P are computationally intractable, as no polynomial time algorithm is known to solve them.

An alternative way of defining complexity classes is by means of reductions. A *reduction* is a transformation f of an instance I_1 of a problem P_1 into an instance I_2 of another problem P_2 such that if I_1 is a 'yes' ('no') instance of P_1, then I_2 is a 'yes' ('no') instance of P_2. A *polynomial time* reduction is a reduction that takes polynomial time with respect to the input instance and can be used to define all the complexity classes harder than P as follows. A problem P is *hard* for a class of problems C, with $C \in \{NP, coNP, PSPACE, NPSPACE, coNPSPACE\}$, if every problem in C can be reduced to P in polynomial time. Hence, an efficient algorithm for P allows us to solve any

problem in C efficiently. Hardness is a powerful tool. In the last 40 years, many great mathematicians failed to prove whether P = NP or P $\neq$ NP. Proving that a problem is NP-hard gives a strong evidence that we cannot "easily" come up with a polynomial time algorithm.

Chapter 2

Stability Testing *

In this chapter, we study the problem of checking whether a BGP network is guaranteed to converge to a stable routing or not. We address several problems related to BGP stability, stating the computational complexity of testing if a given configuration is safe, is robust, or is safe under filtering. Further, we determine the computational complexity of checking popular sufficient conditions for stability.

We adopt a model that captures Local Transit policies, i.e., policies that are functions only of the ingress and the egress points. The focus on Local Transit policies is motivated by the fact that they represent a configuration paradigm commonly used by network operators. We also address the same BGP stability problems in the widely adopted SPP model.

Unfortunately, we find that the most interesting problems are computationally hard even if policies are restricted to be as expressive as Local Transit policies. Our findings suggest that the computational intractability of BGP stability is an intrinsic property of policy-based path vector routing protocols that allow policies to be specified in complete autonomy.

2.1 Introduction

In this section, we overview the BGP stability problem, we motivate it, and we present our contributions to the topic.

*Part of the material presented in this chapter is based on the following publications: M. Chiesa, L. Cittadini, G. Di Battista, S. Vissicchio. Local Transit Policies and the Complexity of BGP Stability Testing. In *Proc. INFOCOM*, IEEE, 2011.

In Chapter 1, we showed that BGP supports a very expressive policy routing language that can be used by network operators to enforce economic interests in the BGP routing process. However, because of this rich expressiveness, BGP is renowned to be prone to oscillations: it can fail to find a stable routing either because there does not exist any [VGE00] or because of bad ordering of messages [GSW02].

Since the effects of BGP *oscillations* can range from performance degradation [WMW+06] to denial of service [KKK07], BGP *stability* attracted lots of research interest. Necessary [FJB07] and sufficient [GSW99, GSW02] conditions for stability have been found and changes to the protocol [ERC+07] or limitations to the expressiveness of the policies [GR00, JR04] have been proposed. However, changing the protocol faces severe deployment issues. Moreover, enforcing sufficient conditions for stability may be incompatible with the need for expressiveness and autonomy that BGP was designed to address. For these reasons, the problem of checking a given BGP configuration for stability has also been deeply studied. As an example, it has been shown that deciding whether a given BGP configuration admits a stable routing is NP-hard [GW99, GSW02].

In this chapter, we consider several fundamental problems related to BGP stability: (*i*) SAFETY [GW99, GSW02], i.e., the problem of verifying that a BGP configuration is guaranteed to converge. (*ii*) SUF [FJB07] and ROBUSTNESS [GSW02], i.e., the problems of verifying that a safe BGP configuration is guaranteed to converge under any filtering action and any link failure, respectively. (*iii*) NO-DW [GSW02] and NO-DR [CBRV09] i.e., the problems of verifying that a BGP configuration does not contain a *dispute wheel* and a *dispute reel*, respectively. The absence of a dispute wheel is a sufficient condition for SAFETY while the absence of a dispute reel is a characterization for SUF.

Despite prior work shed some light on BGP stability, many problems remain open. For example, determining how hard it is to check that a BGP network is guaranteed to converge, i.e., SAFETY, has been an elusive research goal up to now.

We study the complexity of the above problems within three different models for BGP policies: (*i*) The widely adopted SPP model [GSW02], that captures arbitrarily complex BGP policies. (*ii*) The 3-SPP model, that captures the so-called Local Transit policies [GGSS09], a very common configuration paradigm where policies are functions only of the ingress and the egress points. (*iii*) The 2-SPP model, a simplified version of 3-SPP, where either only the ingress point, i.e., the BGP neighbor that announced the route, is considered. In all three cases we adopt the well-known SPVP [GSW02] model for BGP dynamics.

	2-SPP	3-SPP	SPP
SAFETY	P	coNP-hard	coNP-hard
NO-DW	?	coNP-complete	P [GSW99]
NO-DR	?	coNP-complete	coNP-complete
SUF	?	coNP-hard	coNP-complete
ROBUSTNESS	?	coNP-hard	coNP-hard

Table 2.1: Complexity of BGP stability problems in different models. P stands for Polynomial time solvable.

We exploit those models to study how expressive BGP policies can be in order to allow an efficient static assessment of BGP stability, assuming that ASes fully preserve their autonomy. Unfortunately, we find that the most interesting problems are computationally hard even if policies are restricted to be Local Transit only (see Table 2.1). First, solving a long standing open problem [GW99, GSW02], we prove that SAFETY is coNP-hard both in SPP and in 3-SPP. In Chapter 3, we further investigate this result and show a surprising mapping between logic gates and BGP configurations. Second, we prove that SUF is coNP-complete in SPP and that ROBUSTNESS is coNP-hard both in SPP and in 3-SPP. Third, we show that even the NO-DW problem, which can be solved efficiently in SPP [GSW99], is coNP-complete in 3-SPP. Also, we find that the NO-DR problem is coNP-complete both in SPP and in 3-SPP. As a side effect, since any 3-SPP configuration can be expressed in the model proposed in [GW99] without changing the size of the input, our negative results can be extended to the model in [GW99].

Stimulated by the above list of negative results, we investigate whether stability problems can be made tractable by sacrificing the expressive power of policy configurations, while preserving ASes' autonomy. Eventually, we find that SAFETY is solvable in polynomial time in 2-SPP, where policies are restricted in such a way that they are unsuitable for most practical uses.

This chapter is organized as follows. In Section 2.2, we introduce the problem of routing oscillations and we formally state all the BGP stability related problems. In Sections 2.3 and 2.4, we study the complexity of SAFETY and NO-DW respectively, while Section 2.5 studies NO-DR, SUF, and ROBUSTNESS. Section 2.6 reviews related work in the field of BGP stability. Finally, we conclude in Section 2.7. We recall that BGP models have been introduced in Chapter 1.

2.2 BGP Routing Oscillations

We introduce some examples of routing oscillations borrowed from past work [GW99].

Weak and strong oscillations. We show an example of routing oscillation in Fig. 2.1, which has already been discussed in Chapter 1. In the initial state, both vertex 1 and vertex 2 do not have any available path to d. After d announces itself to 1 and 2, they selects (1 d) and (2 d) as their best paths, respectively. Now, consider an activation sequence $A = (A_1\ A_2)$, where $A_1 = A_2 = \{(1,2),(2,1)\}$. After triggering A_0, both vertices 1 and 2 receive a new route from 2 and 1, respectively. Vertex 1 selects path (1 2 d) and 2 selects (2 1 d). Now, since both of them changed their best path, they announce this new route to each other. This is modeled by A_2, where both edges $(1,2)$ and $(2,1)$ are activated. When 1 receives the announcement from 2, it learns that 2 is no longer selecting (2 d) as its best route, but is now selecting (2 1 d). As a consequence, it selects (1 d) as its best route and announces it to 2. By symmetry, 2 selects and announces (2 d), which correspond to the same state before A_1 was triggered. By repeating the same activation pattern, the gadget will never stabilize on a stable routing.

The reader may observe that if the activation of edge $(1,2)$ and $(2,1)$ is not perfectly synchronized, then the oscillation halts. Namely, if 1 announces to 2 that he selected (1 2 d), before that 2 selects (2 1 d), then 2 will not change its best route from (2 d) to (2 1 d) since (1 d) will not be anymore available at 2. This leads to a stable routing.

A routing oscillation is *weak* if there exists an activation sequence that leads to a stable state, as in the DISAGREE example. Otherwise, the oscillation is *strong*, i.e., for any activation sequence, the network never reaches a stable routing. Routing oscillations correspond to cycles in the $Eval(\cdot)$ graph. If a vertex in this cycle has a path to a vertex representing a stable state, then the oscillation is weak, otherwise it is strong. An example of a strong oscillation is the BAD-GADGET, depicted in Fig. 2.2, where routing will keep changing due to unsatisfiable routing policies. The probability that a DISAGREE gadget is going to oscillate in practice is very low. However, in larger networks this probability may be much higher and number of iterations before the network converges may be extremely high. As an example, we can embed a DISAGREE into a larger network, where vertices 1 and 2 are two ASes far away from each other and edges $(1,2)$ and $(2,1)$ are two disjoint paths in the network. In that case an oscillation may persist for a very long time.

For this reason, in this chapter we are interested in the study of both weak

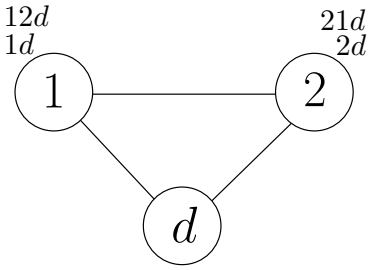

Figure 2.1: DISAGREE.

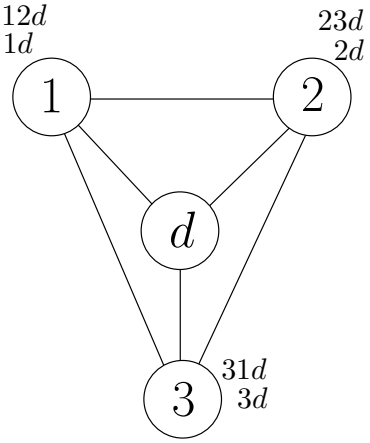

Figure 2.2: BAD-GADGET.

and strong routing oscillations. The problem of checking if a network may be trapped in a strong routing oscillation has been studied in [GW99] and proved to be NP-hard. From now on, we use refer to routing oscillations as both weak and strong oscillations.

We recall that an SPP instance S is *safe* if S is guaranteed to converge for every fair activation sequence and every initial state. Hence, if there exists an arbitrary initial state such that there exists an infinite fair activation sequence, then S is not safe. In our analysis, we limit the space of admissible initial states to the set of states that can be reached from the 0-state. In fact, inconsistent states that cannot be reached from the 0-state can only be triggered by external events (e.g., a link failure). In this case, we say that an instance is *robust* if it is safe under every possible link failure.

The SAFETY problem is defined as follows.

Problem. SAFETY. Given an SPP instance, is it safe?

We are also interested in checking whether an instance is safe for any possible link failures (i.e., is robust) and under any additional filtering. In that case, we refer to the ROBUSTNESS and SUF problem, respectively.

Problem. ROBUSTNESS. Given an SPP instance S, is it robust? More formally, given an SPP instance $((V, E), \mathcal{P}, \Lambda)$, is $((V, E^*), \mathcal{P}, \Lambda)$ safe for any $E^* \subseteq E$?

Problem. SUF. Given an SPP instance $((V, E), \mathcal{P}, \Lambda)$, is $((V, E), \mathcal{P}^*, \Lambda)$ safe for any $\mathcal{P}^* \subseteq \mathcal{P}^*$?

In [GSW02] a sufficient condition for SAFETY, ROBUSTNESS, and SUF has been introduced. Namely, it has been shown that safety is guaranteed if the BGP network does not contain a *dispute wheel* (DW), a particular structure that involves cyclic preferences which cannot be simultaneously satisfied. Hence, the BGP last stability related problem that we tackle is the following one.

Problem. NO-DW. Given an SPP instance $((V, E), \mathcal{P}, \Lambda)$, does it contain a DW ?

In the analysis of these problems, we will make use several times of the following fundamental result about BGP stability.

Theorem 2.1 *[SSZ09] Let I be an SPP instance that admits two or more stable states. Then there exists an infinite fair activation sequence.*

It is worth to observe that, in the proof of Theorem 2.1, the infinite fair activation sequence is a weak routing oscillation in which the network fails to reach any of its stable states.

In Section 2.3, we use this result to show that checking whether a network may trigger a (weak or strong) routing oscillation is computationally intractable. Further, we use the same result to prove limitation to the expressiveness (as in the 2-SPP model) makes it feasible to solve SAFETY.

2.3 The Complexity of the Safety Problem

We recall that an SPP instance S is *safe* if S is guaranteed to eventually reach a stable path assignment for any fair activation sequence. Given an SPP instance, the SAFETY problem asks whether that instance is safe or not.

In this section we study the computational complexity of SAFETY in the SPP model, in the 3-SPP model, and in the 2-SPP model.

We first show that the SAFETY can be solved in polynomial space. More formally, it belongs to PSPACE. This is a "positive" result since it bounds the amount of memory resources that are needed to solve SAFETY.

Theorem 2.1 SAFETY *is in PSPACE in the SPP and* 3*-SPP model.*

Proof: We prove the theorem by showing that the complement of the SAFETY problem can be solved in polynomial space using a Non-Deterministic Turing Machine (NDTM). By Savitch's Theorem [Sip97] and since PSPACE is closed under complement, it follows that SAFETY is in PSPACE.

We non-deterministically generate every possible initial state and store that state on the tape of the machine. We observe that a state has polynomial size w.r.t. the size of the instance both in SPP and in 3-SPP. Then, we generate all the possible activation sequences by non-deterministically activating, at each step of the computation, a random edge in the network. While computing these paths, each independent branch of the computation stores into two sets E_1 and E_2 the edges that are activated at least once during the computation and the edges that must be activated for fairness at least once, respectively. A branch of the computation accepts an instance if it returns to the same initial state and $E_1 = E_2$ (i.e., all the edges that must be activated for fairness are activated during the oscillation). We recall that a NDTM accepts an instance only if at least one of its branches accepts.

It is now trivial to observe that if an instance is not safe, then at least one of the branches of the computation will eventually find a fair activation sequence that corresponds to a routing oscillation. We remind that it does not matter how "long" is the oscillation since PSPACE membership does not enforce any time constraint. Otherwise, if the instance is safe, all the branches will eventually terminate on a stable state. This proves the theorem.

□

Safety is coNP-Hard in the SPP and in the 3-SPP models

We now prove that SAFETY is coNP-hard in the SPP model using a reduction from SAT COMPLEMENT [Pap94]. In order to prove such a result, we first need to show some technical properties regarding the SPP instance of Fig. 2.3, which we call TWISTED gadget. TWISTED has vertex set $V = \{d, x, \bar{x}, a, b, c_1, \ldots, c_m\}$ and edge set $E = \{(d,a),(d,b),(a,x),(b,\bar{x}),(x,\bar{x})\} \cup \{(c_1,x),(c_1,\bar{x}),\ldots,(c_m,x),(c_m,\bar{x})\}$. Policies are as described in Fig. 2.3. Vertices c_i, with $i = 1,\ldots,m$, also have links to another portion of the network not explicitly shown in Fig. 2.3. Each

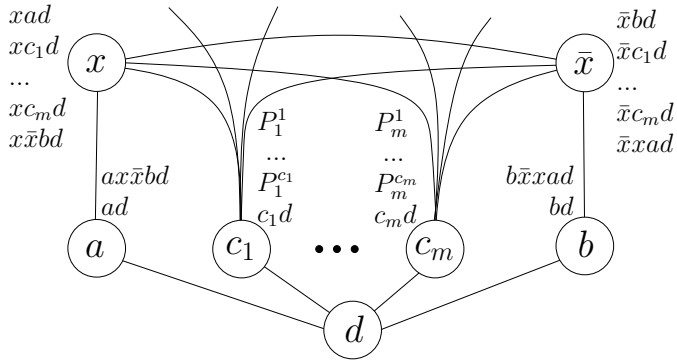

Figure 2.3: TWISTED gadget.

path P_i^j passes through the portion of the network that is not shown and is ranked better than $(c_i\ d)$.

We now prove two important properties of TWISTED.

Lemma 2.2 *For each activation sequence, there do not exist two instants t' and t'' such that $\pi_{t'}(x) = (x\ \bar{x}\ b\ d)$ and $\pi_{t''}(\bar{x}) = (\bar{x}\ x\ a\ d)$.*

Proof: Suppose, for a contradiction, that there exists an activation sequence such that $\pi_{t'}(x) = (x\ \bar{x}\ b\ d)$ and $\pi_{t''}(\bar{x}) = (\bar{x}\ x\ a\ d)$. Denote by t_P the first time when path $P = (v\ \ldots\ d)$ is selected by vertex v. By definition of SPVP, we have that $t_{ad} < t_{xad} < t_{\bar{x}xad}$ and $t_{bd} < t_{\bar{x}bd} < t_{x\bar{x}bd}$. Since vertex d can never withdraw path (d), vertex a (b) cannot select the empty path after t_{ad} (t_{bd}).

Suppose $t_{x\bar{x}bd} \geq t_{xad}$. Note that, after t_{ad}, vertex a can withdraw path $(a\ d)$ only by announcing path $(a\ x\ \bar{x}\ b\ d)$. However, a cannot select path $(a\ x\ \bar{x}\ b\ d)$ because this would imply $t_{ax\bar{x}bd} \leq t_{xad} \leq t_{x\bar{x}bd} < t_{ax\bar{x}bd}$, hence a contradiction. On the other hand, if vertex a does not withdraw path $(a\ d)$ then vertex x never selects path $(x\ \bar{x}\ b\ d)$ because of the availability of the better ranked path $(x\ a\ d)$.

Then it must be $t_{x\bar{x}bd} < t_{xad}$ and, by symmetry, $t_{\bar{x}xad} < t_{xbd}$. A contradiction: $t_{xad} < t_{\bar{x}xad} < t_{\bar{x}bd} < t_{x\bar{x}bd} < t_{xad}$.

□

Lemma 2.3 *For each fair activation sequence, if a vertex c_j and a time t' exist such that $\forall t > t'$ $\pi_t(c_j) = (c_j\ d)$, then a time t'' exists such that $\forall t > t''$ $\pi_t(x) = (x\ a\ d)$ and $\pi_t(\bar{x}) = (\bar{x}\ b\ d)$.*

Proof: By definition of fair activation sequence, there must exist a time $t_1 > t'$ after which paths $(x\ c_j\ d)$ and $(\bar{x}\ c_j\ d)$ are always available to vertices x and $\bar{x}$, respectively. This indefinitely prevents vertex x from selecting path $(x\ \bar{x}\ b\ d)$ and vertex $\bar{x}$ from selecting path $(\bar{x}\ x\ a\ d)$.

As a consequence and because of the fairness, there must exist a time $t_2 > t_1$ such that vertex a can only select path $(a\ d)$ and vertex b can only select path $(b\ d)$. Analogously, there must exist a time $t_3 > t_2$ after which paths $(x\ a\ d)$ and $(\bar{x}\ b\ d)$ are always available at vertices x and $\bar{x}$.

The statement follows by noting that $(x\ a\ d)$ is the most preferred by x and $(\bar{x}\ b\ d)$ is the most preferred by $\bar{x}$.

□

We now use the TWISTED gadget and the results from Lemmas 2.2 and 2.3 to reduce the opposite of the SAT problem, namely SAT COMPLEMENT, to SAFETY. Let F be a logical formula in conjunctive normal form with variables $X_1 \ldots X_n$ and clauses $C_1 \ldots C_m$. We construct an SPP instance S in polynomial time with respect to the size of the SAT COMPLEMENT instance as follows (see Figure 2.4).

For each clause C_i, add a vertex c_i to S. For each variable X_i, add a copy of the TWISTED gadget with x, $\bar{x}$, a, and b replaced by x_i, $\bar{x}_i$, a_i, and b_i, respectively. In the copy, for each clause C_j, $(x_i\ c_j\ d) \in \mathcal{P}^{x_i}$ and $(\bar{x}_i\ c_j\ d) \in \mathcal{P}^{\bar{x}_i}$. For each vertex c_j, path $(c_j\ x_i\ \bar{x}_i\ b_i\ d) \in \mathcal{P}^{c_j}$ if literal X_i is in C_j and path $(c_j\ \bar{x}_i\ x_i\ a_i\ d) \in \mathcal{P}^{c_j}$ if literal $\bar{X}_i$ is in C_j. Path $(c_j\ d)$ is the least preferred path at each vertex c_j, while the relative preference among other paths is not significant.

Theorem 2.4 SAFETY *is coNP-hard in the SPP model.*

Proof: Consider a logical formula F and construct the corresponding SPP instance $S = ((V, E), \mathcal{P}, \Lambda)$ as described above. We now prove the statement in two parts.

If F is unsatisfiable then S is safe.

Consider any fair activation sequence and assume that all vertices c_j select a path $P \neq (c_j\ d)$ infinite times. Let $W = \{x_i \in V \mid \exists c_j, \exists t : \pi_t(c_j) = (c_j\ x_i\ \bar{x}_i\ a_i\ d)\}$ and $Z = \{\bar{x}_i \in V \mid \exists c_j, \exists t : \pi_t(c_j) = (c_j\ \bar{x}_i\ x_i\ b_i\ d)\}$. Consider the boolean assignment M such that X_i is assigned to TRUE if $x_i \in W$, and

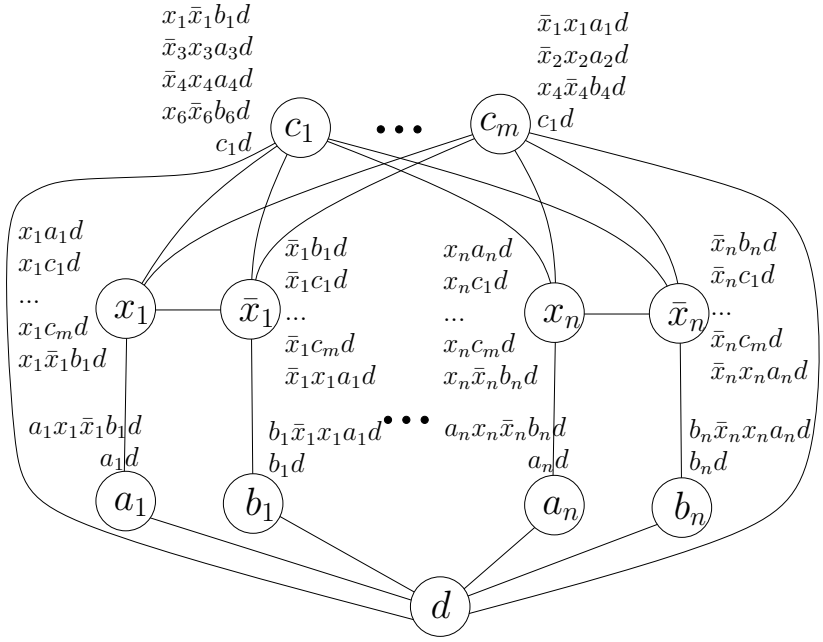

Figure 2.4: Reduction from SAT COMPLEMENT to SAFETY.

X_i is assigned to FALSE if $\bar{x}_i \in Z$. Lemma 2.2 ensures that $Z \cap W = \emptyset$. By construction of S, each clause in F is satisfied by at least a variable in M, that is a contradiction.

Then there must exist a time t' and a vertex c_k such that $\forall t > t'\ \pi_t(c_k) = (c_k\ d)$. By Lemma 2.3, this implies that there exists a time $t'' > t'$ after which each vertex x_i always selects path $(x_i\ a_i\ d)$ and each vertex $\bar{x}_i$ always selects path $(\bar{x}_i\ b_i\ d)$. The fairness of the activation sequence guarantees that, eventually, each vertex c_j permanently selects $(c_j\ d)$, each vertex a_i permanently selects $(a_i\ d)$, and each vertex b_i permanently selects $(b_i\ d)$. It is easy to check that such a path assignment is stable. Since any fair activation sequence leads to a stable path assignment, if F is unsatisfiable then S is safe.

If F is satisfiable then S is not safe.

Let M be a boolean assignment that satisfies F. We now show that S has at least two stable path assignments.

Let π' be a path assignment such that $\pi'(x_i) = (x_i\ a_i\ d)$, $\pi'(\bar{x}_i) = (\bar{x}_i\ b_i\ d)$,

$\pi'(a_i) = (a_i\ d)$, $\pi'(b_i) = (b_i\ d)$, and $\pi'(c_j) = (c_j\ d)$, where $i = 1, \ldots, n$ and $j = 1, \ldots, m$. It is easy to check that π' is a stable path assignment.

Also, consider path assignment π'' defined as follows. For each variable X_i such that $M(X_i) = \top$, let $\pi''(x_i) = (x_i\ \bar{x}_i\ b_i\ d)$, $\pi''(\bar{x}_i) = (\bar{x}_i\ b_i\ d)$, $\pi''(a_i) = (a_i\ x_i\ \bar{x}_i\ b_i\ d)$, $\pi''(b_i) = (b_i\ d)$. For each variable X_i such that $M(X_i) = \bot$, let $\pi''(\bar{x}_i) = (\bar{x}_i\ x_i\ a_i\ d)$, $\pi''(x_i) = (x_i\ a_i\ d)$, $\pi''(a_i) = (a_i\ d)$, $\pi''(b_i) = (b_i\ \bar{x}_i\ x_i\ a_i\ d)$. Each vertex c_j selects in π'' the most preferred among paths in set $R_j = \{(c_j\ x_i\ \bar{x}_i\ b_i\ d) \in \mathcal{P}^{c_j} | M(X_i) = \top\} \cup \{(c_j\ \bar{x}_i\ x_i\ a_i\ d) \in \mathcal{P}^{c_j} | M(X_i) = \bot\}$.

Observe that $\forall j\ R_j \neq \oslash$ since each clause is satisfied by at least one variable in M. We now show that path assignment π'' is stable. Each vertex c_j, $j = 1, \ldots, m$, selects the best ranked path in R_j and, by construction, no better alternative is available at c_j. For each variable X_i such that $M(X_i) = \top$ ($M(X_i) = \bot$) vertices a_i (b_i) and $\bar{x}_i$ (x_i) select their best ranked path, while vertices b_i (a_i) and x_i ($\bar{x}_i$) cannot select any other path except the one defined by π''.

We conclude that, if F is satisfiable, then S has two stable path assignments. The statement follows by Theorem 3.1 of [SSZ09], which proves that any SPP instance with two distinct stable path assignments is not safe.

□

Theorem 2.5 SAFETY *is coNP-hard in the* 3*-SPP model.*

Proof: We can use the same reduction from SAT COMPLEMENT to SAFETY applied in Theorem 2.4. In fact, the SPP instance constructed in the reduction can be easily translated into a 3-SPP instance, since every permitted path at each vertex is uniquely identified by the first three hops in the path. The reduction proves the statement.

□

Safety can be efficiently checked in the 2-SPP model

The 2-SPP model allows ASes to only specify path fragments of length 2. In other words, policies can be specified only on a per-neighbor basis: all paths from the same neighbor are either accepted or filtered and are equally preferred. As in 3-SPP, any arbitrary deterministic criterion can break ties. By applying the technique in [FSS06], it can be shown that every 2-SPP instance has at least a stable path assignment π and π can be computed in polynomial time.

Observe, however, that 2-SPP allows configurations that are not safe, e.g., the famous SPP instance DISAGREE [GW99] can be represented in 2-SPP.

Given a 2-SPP instance $\tilde{S} = (G = (V, E), \tilde{\mathcal{P}}, \tilde{\Lambda})$, a path fragment $(u\ v)$, with $u, v \in V$, is *consistent* if there exists a sequence of permitted path fragments $P_1, P_2, \ldots, P_n$ in $\tilde{\mathcal{P}}$ such that $(u\ v)P_1P_2 \ldots P_n(d)$ is a simple path on G. Consistency of a given path fragment can be trivially checked in polynomial time. In the following, we consider only 2-SPP instances in which all permitted path fragments are consistent.

We show an algorithm, called NH-GREEDY, that efficiently solves SAFETY in 2-SPP. NH-GREEDY is an adaptation of the greedy algorithm in [GSW02]. NH-GREEDY incrementally grows a set of *stable* vertices for which convergence is guaranteed. The set of stable vertices at iteration i of NH-GREEDY is denoted by V_i. At iteration i NH-GREEDY also computes a partial path assignment π_i^*, that is, a path assignment where $\forall u \notin V_i\ \pi_i^*(u) = \epsilon$. At the beginning, $V_0 = \{d\}$ and $\pi_0^*(d) = (d)$. Let H_i be the set of vertices $u \notin V_i$ such that the most preferred path fragment is either $B^u = \epsilon$ or $B^u = (u\ v)$, where $v \in V_i$. If H_i is not empty, then $V_{i+1} = V_i \cup H_i$, $\pi_{i+1}^*(u) = \pi_i^*(u)$ if $u \in V_i$, and $\pi_{i+1}^*(u) = B^u\pi_i^*(u)$ for each $u \in H_i$. Otherwise, if H_i is empty, NH-GREEDY terminates. At each iteration, NH-GREEDY either inserts at least one vertex in V_i or terminates, hence it terminates after at most $|V|$ iterations. If NH-GREEDY terminates after k iterations with $V_k = V$ then we say that it *succeeds*, otherwise it *fails*. Being derived from the algorithm in [GSW02], NH-GREEDY inherits the properties shown in [CRCD09]. In particular, this implies that NH-GREEDY is correct, i.e., after k iterations each vertex $v \in V_k$ is guaranteed to eventually select path $\pi_k^*(v)$ in any fair activation sequence. As a consequence, if NH-GREEDY succeeds then the 2-SPP instance is safe. We now show that if NH-GREEDY fails the instance is not safe.

Let $G' = (V, E')$ be the directed graph such that $(u, v) \in E'$ iff $(u\ v) \in \tilde{\mathcal{P}}^x$. Given a partial path assignment π and a vertex u such that $\tilde{\mathcal{P}}^u \neq \{\epsilon\}$ and $\pi(u) = \epsilon$, the *ideal path* P_u^π of u in π is the simple path from u to d obtained by performing a depth-first visit on G' starting from u. Vertices are visited according to $\tilde{\Lambda}$, i.e., the neighbor with the highest preference is visited first. By definition, $P_u^\pi = (w_1\ \ldots\ w_n\ v_1\ \ldots\ v_m)$, where $w_1 = u$, $v_m = d$, $n \geq 1$, $m \geq 1$, $(u\ w_1)$ is the most preferred fragment in $\mathcal{P}^u$, $\pi(w_i) = \epsilon$ for $i = 1, \ldots, n$, and $\pi(v_j) = (v_j\ \ldots\ v_m)$ for $j = 1, \ldots, m$. Intuitively, the ideal path of u traverses the best ranked neighbor of u and such that all vertices $w_i \in P_u^\pi$ select the best-ranked simple path that extends a path in π and ends in d. Observe that such a path must exist because all path fragments are assumed to be consistent, i.e., $(u\ w_1)$ generates at least a path on G.

Assume that NH-GREEDY fails on a 2-SPP instance $\tilde{S}$ after k iterations with partial path assignment π_k^* and let u be any vertex in $V - V_k$.

Lemma 2.6 *There exists a stable path assignment $\bar{\pi}$ on $\tilde{S}$ such that u selects its ideal path, i.e., $\bar{\pi}(u) = P_u^{\pi_k^*}$.*

Proof: We construct a sequence of partial path assignments $\pi_1, \pi_2, \ldots, \bar{\pi}$ by iteratively growing π_k^*. Let $P_u^{\pi_k^*} = (u\ w_1\ \ldots\ w_n\ v_1\ \ldots\ v_m)$ be the ideal path of vertex u in π_k^*. Let $\pi_1(u) = P_u^{\pi_k^*}$, for each $w_i \in P_u^{\pi_k^*}$ let $\pi_1(w_i) = (w_i\ \ldots\ w_n\ v_1\ \ldots\ v_m)$ and for each $v \in V_k$ let $\pi_1(v) = \pi_k^*(v)$. Then, we consider any other vertex z such that $\pi_1(z) = \epsilon$ and $z \in V - V_k$ (if one exists, otherwise stop). Given $P_z^{\pi_1}$ the ideal path of z, we construct the (partial) path assignment π_2 by extending π_1 as above. Since V is finite, we eventually find a path assignment $\bar{\pi}$ defined for each $v \in V$.

We now show that $\bar{\pi}$ is stable. Suppose, for a contradiction, that there exists a vertex v that has an alternative path towards d that is preferred to $\bar{\pi}(v)$. By construction, v must either be in V_k or be part of the ideal path of some vertex x. In the first case, being $\bar{\pi}$ an extension of π_k^*, v is guaranteed to select path $\bar{\pi}(v)$. In the latter case, by definition of ideal path, v can not have a better-ranked alternative, since the depth-first visit analyzes paths at each vertex in a decreasing order of preference. In both cases, we have a contradiction.

□

Theorem 2.7 SAFETY *can be solved in polynomial time in the 2-SPP model.*

Proof: Given a 2-SPP instance S, S is safe if and only if NH-GREEDY succeeds. We have already discussed that if NH-GREEDY succeeds S is safe. On the other hand, if NH-GREEDY fails after k iterations, it is possible to build two distinct stable path assignments. In fact, let u be any vertex in $V - V_k$. Lemma 2.6 ensures that there exists a stable path assignment π' such that $\pi'(u) = P_u^{\pi_k^*}$. Path $P_u^{\pi_k^*}$ must be in the form $P_u^{\pi_k^*} = P'(z\ v)P''$ where $z \notin V_k$ and $v \in V_k$. Observe that $z \neq u$, since $u \notin V_k$. Consider the stable path assignment π'' such that $\pi''(z) = P_z^{\pi_k^*}$, constructed as in Lemma 2.6. Obviously, $\pi' \neq \pi''$ at least for vertex z since $z \notin V_k$. Since two distinct stable path assignments exist, by Theorem 3.1 of [SSZ09] S is not safe.

□

2.4 Searching for Dispute Wheels

In Section 2.3 we proved that SAFETY turns out to be a computationally hard problem. A possible way to overcome the unfeasibility of testing SAFETY could be verifying if at least sufficient conditions for SAFETY are satisfied.

In [GSW02] a celebrated sufficient condition for SAFETY has been introduced. Namely, it has been shown that safety is guaranteed if the BGP network does not contain a *dispute wheel* (DW), a particular structure that involves cyclic preferences which cannot be simultaneously satisfied. In the SPP model, a DW $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ is a sequence of vertices $\vec{\mathcal{U}} = (u_0\ u_1 \dots u_{k-1})$ and sequences of nonempty paths $\vec{\mathcal{Q}} = (Q_0\ Q_1 \dots Q_{k-1})$, called *spoke paths*, and $\vec{\mathcal{R}} = (R_0\ R_1 \dots R_{k-1})$, called *rim paths*, such that:

(i) R_i is a path from u_i to u_{i+1}

(ii) $Q_i \in \mathcal{P}^{u_i}$

(iii) $R_i Q_{i+1} \in \mathcal{P}^{u_i}$

(iv) $\lambda^{u_i}(Q_i) \geq \lambda^{u_i}(R_i Q_{i+1})$

where all indexes are to be intended modulo k. Since an instance of k-SPP can be uniquely translated into an SPP instance, we can extend the definition of DW as follows: we say that an instance of k-SPP *contains* a DW if its translation to SPP contains a DW. Verifying the absence of a DW in a BGP network is referred to as the NO-DW problem. In the SPP model NO-DW can be solved in polynomial time [GSW99] by finding a cycle in an auxiliary graph called *dispute digraph*, whose construction takes polynomial time.

In the following, we analyze the computational complexity of NO-DW in the 3-SPP model. We do it in two steps. First, we deal with the basic problem of deciding whether a given vertex of a 3-SPP instance can establish a path to d. We call this problem PATH and we show that it is NP-complete. Second, we exploit such a result to prove that NO-DW in the 3-SPP model is coNP-complete.

PATH is NP-complete since it is possible to reduce 3-SAT to PATH. Let F be a 3-SAT formula with variables $X_1, \dots, X_n$ and clauses $C_1, \dots, C_m$. We construct a 3-SPP instance as follows. For each variable X_i we insert vertices v_i, x_i, and $\bar{x}_i$, and we build a gadget having edges (v_i, x_i) and $(v_i, \bar{x}_i)$. For each clause C_j we build a gadget consisting of vertices c_j and $c_{j,k}$ and edges $(c_j, c_{j,k})$, $(c_{j,k}, c_{j+1})$ with $k = 1, 2, 3$. Also, we add to the instance vertices v_{n+1}, c_{m+1}

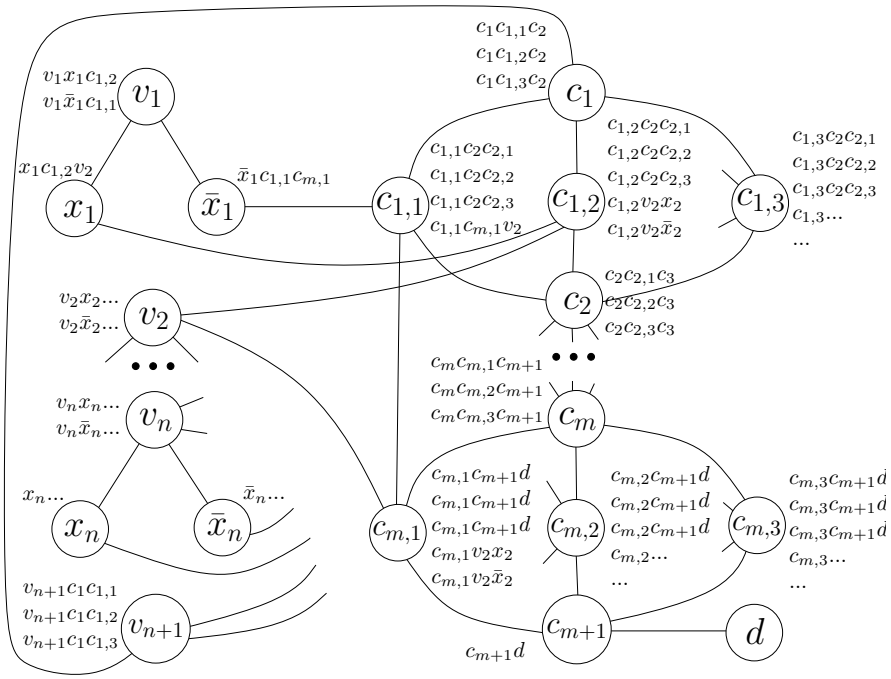

Figure 2.5: Reduction from 3-SAT to PATH.

and d, and edges (c_{m+1}, d) and (v_{n+1}, c_1). Fig. 2.5 shows an example of the construction, where variable gadgets are on the left side while clause gadgets are on the right side.

Intuitively, vertex v_i attempts to establish a path to d via x_i ($\bar{x}_i$) if the corresponding 3-SAT variable X_i is TRUE (FALSE). Vertices $c_{j,k}$ are called *literal* vertices because each of them represents one of the three literals that appear in clause C_j.

Consider literal X_i, $i = 1, \ldots, n$. Let $P = (v_i\ x_i\ c_{j_1,k_1}\ \ldots\ c_{j_n,k_n}\ v_{i+1})$ be the path from vertex v_i to vertex v_{i+1} that traverses all the literal vertices c_{j_p,k_p} such that the corresponding literal in clause C_{j_p} is $\bar{X}_i$. If there are no such literals, then path P simply consists of edges (v_i, x_i) and (x_i, v_{i+1}). We add to the graph constructed so far all the edges of P. We apply exactly the same procedure for literal $\bar{X}_i$. We then get from path P all the ordered triples of consecutive vertices and add each triple $(u\ v\ w)$ to $\tilde{\mathcal{P}}^u$. For example, in

Fig. 2.5 there is a path $(v_1\ \bar{x_1}\ c_{1,1}\ c_{m,1}\ v_2)$ because we assume, without loss of generality, that the first literal both in C_1 and in C_m is X_1. For each vertex c_j, set $\tilde{\mathcal{P}}^{c_j}$ only contains paths $(c_j\ c_{j,k}\ c_{j+1})$, with $k = 1, 2, 3$ and for each vertex $c_{j,k}$, we add to $\tilde{\mathcal{P}}^{c_{j,k}}$ paths $(c_{j,k}\ c_{j+1}\ c_{j+1,l})$, with $l = 1, 2, 3$. This construction ensures that if vertex v_i attempts to establish a path to d via x_i $(\bar{x_i})$, it cannot use a path including $c_{j,k}$ iff $\bar{X}_i$ (X_i) is the k-th literal in C_j, representing the fact that clause C_j cannot be satisfied by literal $c_{j,k}$. We define $\tilde{\mathcal{P}}^{v_{n+1}} = \{(v_{n+1}\ c_1\ c_{1,k}) | \forall k = 1, 2, 3\}$ and $\tilde{\mathcal{P}}^{c_{m+1}} = \{(c_{m+1}\ d)\}$.

Function $\lambda^{c_{j,k}}$, where $j = 1, \ldots, m$ and $k = 1, 2, 3$, is such that paths $(c_{j,k}\ c_{j+1}\ c_{j+1,l})$, with $l = 1, 2, 3$, are better ranked than others. Preferences at vertices v_i, x_i, $\bar{x_i}$ and c_j, where $i = 1, \ldots, n+1$ and $j = 1, \ldots, m+1$, can be assigned arbitrarily. It is easy to check that the instance of 3-SPP can be built in polynomial time.

Lemma 2.1 PATH *is NP-complete in the* 3*-SPP model.*

Proof: Consider the construction depicted in Fig. 2.5. We now show that vertex v_1 can establish a path to d iff the corresponding 3-SAT formula F is satisfiable.

Observe that every path P from v_1 to d, if any, must be in the form $P = AB$ where $A = (v_1 \ldots v_2 \ldots v_{n+1})$ and $B = (v_{n+1}\ c_1\ c_{1,j_1}\ \ldots c_m\ c_{m,j_m} d)$. Since vertex v_i must choose either x_i or $\bar{x_i}$ and there is only one path connecting x_i $(\bar{x_i})$ to v_{i+1}, path A can be mapped to a boolean assignment for F. By construction, only literal vertices $c_{j,k}$ can appear twice in P, since they can appear both in A and in B.

Now, if $P = AB$ exists, then every c_j can reach d via one of its neighbors $c_{j,1}$, $c_{j,2}$ and $c_{j,3}$ which is not traversed by path A. By construction, this implies that the boolean assignment mapped to path A satisfies at least one literal in every clause, hence F is satisfiable.

On the other hand, if there is no path P from v_1 to d, then for any choice of path A there exists a vertex c_j that is unable to reach d via any of its neighbors because they all appear in A. By construction, this implies that for each boolean assignment there exists a clause C_j that is false, hence F is unsatisfiable.

The above arguments prove that PATH is NP-hard. NP-completeness follows by noting that a path P from v_1 to d is a succinct certificate for PATH because P has polynomial size and it takes polynomial time to check if P can be generated by any fragment of v_1.

□

We now use the reduction as above for proving that NO-DW is coNP-complete. First of all, we prove that the 3-SPP instance built in the reduction does not contain any DW.

Lemma 2.2 *The 3-SPP instance S constructed in the reduction from* 3-SAT *to* PATH *(see Fig. 2.5) contains no DW.*

Proof: Suppose, for a contradiction, that S contains a DW and assume that no vertex c_i can appear in any rim path. We now show that rim paths of such a DW do not form a cycle, that is a contradiction since concatenating rim paths must result in a cycle by definition of DW (each rim path connects a pivot vertex with its successor).

By construction, permitted paths of all the vertices in S are subpaths of $P = P_1 \ \ldots \ P_n \ (v_{n+1} \ c_1) \ Q_1 \ \ldots \ Q_m \ (c_{m+1} \ d)$. Paths Q_i are such that $Q_i = \{c_i \ c_{i,j} \ c_{i+1}\}$, where j is either 1, 2, or 3. Each path P_i starts at v_i, ends in v_{i+1}, and traverses x_i ($\bar{x}_i$) and all the vertices $c_{j,k}$ such that the corresponding literal in clause C_j is $\bar{X}_i$ (X_i). This implies that $P_j \cap P_{j+1} = \{v_{j+1}\}$ for each j, and $P_j \cap P_k = \oslash$, if $k \notin \{j, j+1\}$. Since no rim path can contain a node c_i, all the rim paths must be subpaths of $P_1 \ P_2 \ \ldots \ P_n$. However, since vertices v_i are ordered and all paths P_i intersects only at vertices v_i, no cycle among rim paths can be built, yielding a contradiction.

The proof is completed by showing that no vertex c_i can appear in any rim path of any DW Π. In fact, suppose that there exists a non empty set of vertices $Z = \{c_j, \ldots, c_k\}$ such that each vertex $c_i \in Z$ appears in one or more rim paths. Obviously, c_{m+1} cannot belong to Z. Consider, among all the vertices in Z, the vertex c_h with the highest index. Let R be a rim path in which appears c_h and let $R[c_h]$ be the subpath of R starting from c_h. By definition of c_h and by construction of S, $R[c_h]$ can only be $(c_h \ c_{h,h'})$, with $h' = 1, 2, 3$. In fact, all permitted paths at c_h are sequences of vertices c_i and $c_{i,j}$, such that $i > h$ and c_{h+1} cannot appear in $R[c_h]$ by definition of c_h. Hence, vertex $c_{h,h'}$ must be a pivot vertex of Π, and its spoke path must be a path $(c_{h,h'} \ c_{h+1} \ \ldots \ d)$ since it must be extended by a permitted path of c_h. By definition of DW, the rim path of $c_{h,h'}$ should be one among paths $(c_{h,h'} \ c_{h+1} \ \ldots \ d)$, that is, c_{h+1} is also on a rim path. This leads to a contradiction, because c_h is defined to be the vertex with the highest index among those appearing in a rim path.

□

Theorem 2.3 NO-DW *is coNP-complete in the* 3*-SPP model.*

Proof: We prove the statement by reducing 3-SAT COMPLEMENT to NO-DW. Let F be a logical formula with variables $X_1, \ldots, X_n$ and clauses $C_1, \ldots, C_m$. We construct an instance $\tilde{S} = ((V, E), \tilde{\mathcal{P}}, \tilde{\Lambda})$ of 3-SPP as follows. Let $\tilde{S}' = ((V', E'), \tilde{\mathcal{P}}', \tilde{\Lambda}')$ be the 3-SPP instance constructed as above (see Fig. 2.5). Let $V = V' \cup \{1, 2\}$, let $E = E' \cup \{(1, v_1), (1, 2), (2, d)\}$, let $\tilde{\mathcal{P}} = \tilde{\mathcal{P}}' \cup \{(1\ v_1\ x_1)$, $(1\ v_1\ \bar{x_1})$, $(2\ d)$,$(1\ 2\ d)$, $(2\ 1\ v_1)\}$ and let $\tilde{\Lambda} = \tilde{\Lambda}' \cup \{\tilde{\lambda}^1, \tilde{\lambda}^2\}$, where $\tilde{\lambda}^1$ is such that path $(1\ 2\ d)$ is most preferred and $\tilde{\lambda}^2$ is such that path $(2\ 1\ d)$ is most preferred.

Intuitively, we added two extra vertices 1 and 2, and defined policies such that a DW exists in $\tilde{S}$ only if 1 can establish a path to d. By applying the same arguments as in the proof of Lemma 2.1 we therefore have that $\tilde{S}$ has no dispute wheel iff F is unsatisfiable. This implies that NO-DW is coNP-hard in the 3-SPP model. The proof is complete by noting that a DW on $\tilde{S}$ is a succinct disqualification for NO-DW, that is, a succinct proof that $\tilde{S}$ is a negative instance.

□

2.5 Safety Under Filtering and Robustness

In this section we study the computational complexity of safety under filtering and robustness.

Problem SAFETY UNDER FILTERING (SUF) [FJB07] is defined as follows: given an SPP instance S, will S remain safe under arbitrary filtering of paths? Similarly, the ROBUSTNESS problem [GSW02] requires that the input SPP instance be safe even under arbitrary link failures. It has been proved in [CBRV09] that the two problems are distinct, as there exist SPP instances that are robust but not safe under filtering.

It is known [CBRV09] that an SPP instance is safe under filtering iff it does not contain a dispute reel (DR). Intuitively, a dispute reel is a dispute wheel such that spoke paths form a tree T and rim paths never intersect T nor contain more than two pivot vertices. Let $P[v]$ denote the subpath of P starting at vertex v. A *dispute reel* (DR) is a dispute wheel such that

(*i*) (*Pivot vertices appear in exactly three paths*) – for each $u_i \in \vec{\mathcal{U}}$, u_i only appears in paths Q_i, R_i and R_{i-1}.

(*ii*) (*Spoke and rim paths do not intersect*) – for each $u \notin \vec{\mathcal{U}}$, if $u \in Q_i$ for some i, then no j exists such that $u \in R_j$.

(*iii*) (*Spoke paths form a tree*) – for each distinct $Q_i, Q_j \in \vec{\mathcal{Q}}$, if $v \in Q_i \cap Q_j$, then $Q_i[v] = Q_j[v]$.

SUF, ROBUSTNESS and DR are defined in the SPP model. The definition of DR can be extended to k-SPP by translating the considered k-SPP instance to SPP. SUF and ROBUSTNESS are defined in 3-SPP as the problems of determining if an input 3-SPP instance is safe even under arbitrary filtering of path fragments or under arbitrary link failures, respectively. It is easy to check that a 3-SPP instance is robust iff the corresponding SPP instance is robust. On the contrary, it is not known if a SUF 3-SPP instance corresponds to a SUF SPP instance, nor if the absence of a DR is a characterization for SUF in the 3-SPP model.

No Dispute Reel is CoNP-Complete

We now prove that NO-DR is coNP-hard by reducing 3-SAT COMPLEMENT to SUF in polynomial time. Refer to Fig. 2.6 for an example of the reduction.

Let F be a logical formula, with variables $X_1, \ldots, X_n$ and clauses $C_1, \ldots, C_m$. For each variable X_i, we add to the SPP instance a gadget consisting of three vertices, namely a_i, x_i, and $\bar{x}_i$, and four edges, namely $(x_i\ d)$, $(\bar{x}_i\ d)$, $(a_i\ x_i)$ and $(a_i\ \bar{x}_i)$. Vertices x_i and $\bar{x}_i$ have no permitted paths other than $(x_i\ d)$ and $(\bar{x}_i\ d)$, respectively. Permitted paths at vertex a_i are $\mathcal{P}^{a_i} = \{(a_i\ x_i\ d), (a_i\ \bar{x}_i\ d)\}$ and the ranking among them is not significant. Intuitively, a_i represents variable X_i. Gadgets corresponding to variables are at the bottom of Fig. 2.6.

For each clause C_j, we add to the SPP instance a gadget containing vertices c_j, $c_{j,i}$, and edges $(c_j, c_{j,i})$ and $(c_{j,i}, c_{j+1})$, where $i = 1, \ldots, 3$. Intuitively, vertex c_j (*clause vertex*) represents clause C_j while vertex $c_{j,i}$ (*literal vertex*) represents the i-th literal in C_j. Further, if X_l appears in the i-th literal in C_j, then we add an edge $(a_l, c_{j,i})$, and we set $\mathcal{P}^{c_{j,i}} = \{(c_{j,i}\ c_{j+1}\ d), (c_{j,i}\ a_l\ x_l\ d)\}$ if literal represented by $c_{j,i}$ is X_l, $\mathcal{P}^{c_{j,i}} = \{(c_{j,i}\ c_{j+1}\ d), (c_{j,i}\ a_l\ \bar{x}_l\ d)\}$ otherwise. Among the two paths in $\mathcal{P}^{c_{j,i}}$, $(c_{j,i}\ c_{j+1}\ d)$ is the most preferred. The permitted paths at vertex c_j are $(c_j\ d)$ plus the extension of the longest path permitted at each vertex $c_{j,i}$, $i = 1, \ldots, 3$. Path $(c_j\ d)$ is the least preferred path, while the ranking of other paths can be arbitrary. Gadgets corresponding to clauses are placed at the top of Fig. 2.6.

Observe that the SPP instance built in the reduction contains several DWs. Vertices a_i, x_i, $\bar{x}_i$ can not be pivot vertices of any dispute wheel, since they only have direct paths to d. In fact, by arbitrarily picking exactly one literal

vertex $c_{j,i}$ for each clause vertex c_j, we construct a DW where pivot vertices are all clause vertices and the selected literal vertices.

For any DW Π, each pivot appears in exactly three paths and spoke paths never intersect rim paths, hence conditions (*i*) and (*ii*) of the definition of DR are satisfied. However, spoke paths are not guaranteed to form a tree (condition (*iii*) of the definition of DR), so DWs are not guaranteed to be DRs.

Since spoke paths in Π only share vertices a_i, condition (*iii*) is satisfied only if there are no two distinct spoke paths Q_1 and Q_2 in Π such that $Q_1 = (\ldots a_i\ x_i\ d)$ and $Q_2 = (\ldots a_i\ \bar{x_i}\ d)$, which represents the fact that variable X_i cannot be TRUE and FALSE at the same time.

Theorem 2.1 NO-DR *is coNP-complete in the SPP model.*

Proof: Consider a logical formula F and construct the corresponding SPP instance S as described above.

If F is unsatisfiable then S does not contain a DR.

Suppose, for a contradiction, that S contains a DR Π. Then, condition (*iii*) ensures that, for each a_i, either path $(a_i\ x_i\ d)$ or path $(a_i\ \bar{x_i}\ d)$ is a subpath of all spoke paths that traverse vertex a_i. This property allows us to construct a boolean assignment for F by setting variable X_i to TRUE if there exists a spoke path $Q' = (\ldots a_i\ x_i\ d)$ or to FALSE if there exists a spoke path $Q'' = (\ldots a_i\ \bar{x_i}\ d)$.

As we already observed, Π contains exactly one literal vertex for each clause vertex. By construction of S, we have that the boolean assignment corresponding to Π satisfies at least one literal in each clause in F, contradicting the hypothesis that F is unsatisfiable.

If F is satisfiable then S contains at least one DR.

Consider a boolean assignment M that satisfies F. We will now show a DR $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ in S. Vertices c_j must be pivot vertices, that is, $u_{2j-1} = c_j$ and $Q_{2j-1} = (c_j\ d)$ for $j = 1, \ldots, m$. For each literal vertex $c_{j,i}$, if its least preferred path is $(c_{j,i}\ a_i\ x_i\ d)$ and $M(X_i) = \top$ then we set $u_{2j} = c_{j,i}$, $Q_{2j} = (c_{j,i}\ a_i\ x_i\ d)$, $R_{2j-1} = (c_j\ c_{j,i})$, and $R_{2j} = (c_{j,i}\ c_{j+1})$. We set u_{2j}, R_{2j} and R_{2j-1} to the same values also if the least preferred path of $c_{j,i}$ is $(c_{j,i}\ a_i\ \bar{x_i}\ d)$ and $M(X_i) = \bot$, however in this case we set a different spoke path $Q_{2j} = (c_{j,i}\ a_i\ \bar{x_i}\ d)$. Whenever multiple literal vertices $c_{j,i}$ for the same clause vertex c_j satisfy the above conditions, we arbitrarily pick only one among them.

It is easy to see that, since each clause in F is satisfied by at least one literal, Π is a DW. Moreover, by construction of Π we have that for each vertex

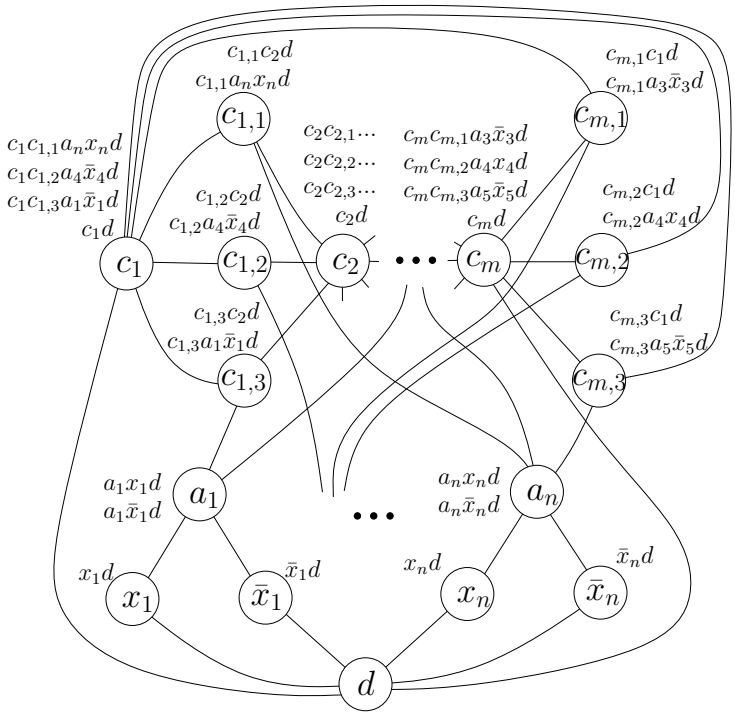

Figure 2.6: Reduction from SAT COMPLEMENT to SUF.

a_i only one among $(a_i\ x_i\ d)$ and $(a_i\ \bar{x}_i\ d)$ can be traversed by spoke paths in Π, hence satisfying condition (iii) of the definition of DR. Conditions (i) and (ii) are trivially satisfied by Π. Hence, Π is a DR.

CoNP-completeness follows from noting that a DR on S is a succinct disqualification for NO-DR.

□

We now state the complexity of NO-DR in 3-SPP.

Theorem 2.2 NO-DR *is coNP-complete in the* 3*-SPP model.*

Proof: Observe that all the permitted paths in SPP instance built in the reduction 3-SAT COMPLEMENT to 3-SPP are entirely identified by the first three hops. Hence, an analogous reduction can be applied from 3-SAT COMPLEMENT

to 3-SPP. The statement follows from the fact that a DR on a 3-SPP instance is a succinct disqualification for NO-DR.

□

Complexity of Safety Under Filtering and Robustness

Since the absence of a DR is a characterization of SUF in the SPP model, we can state the following theorems.

Theorem 2.3 SUF *is coNP-complete in the SPP model.*

Proof: The statement directly follows from Theorem 2.1 considering that the absence of a DR is a necessary and sufficient condition for SUF in the SPP model [CBRV09]. □

Theorem 2.4 SUF *is coNP-hard in the* 3*-SPP model.*

Proof: Let S be the SPP instance in Fig. 2.6 and construct the 3-SPP instance S' by truncating all paths in S with length greater than 3. Since each permitted path in S is identified by its first three hops, there is a one-to-one mapping between permitted paths in S and permitted paths in S'. This implies that each filter in S can be mapped to a unique filter in S'. We conclude that S' is SUF iff S is SUF, hence a construction analogous to that described in Section 2.5 can be applied to reduce from 3-SAT COMPLEMENT to SUF in 3-SPP.

□

In general, SUF implies ROBUSTNESS, while the opposite does not hold [CBRV09]. However, observe that the SPP instance in Fig. 2.6 is SUF iff it is also robust. In fact, filtering a path $P = (u\ v\ \ldots d)$ at vertex u is equivalent to removing edge (u, v) from the graph. This property allows us to reduce 3-SAT COMPLEMENT to ROBUSTNESS using the same reduction used in Theorem 2.3.

Theorem 2.5 ROBUSTNESS *is coNP-hard in the SPP model.*

Since a 3-SPP instance is robust iff the corresponding SPP instance is robust, we can directly extend Theorem 2.5.

Theorem 2.6 ROBUSTNESS *is coNP-hard in the* 3*-SPP model.*

2.6 Related Work

In [GW99] a BGP model is proposed where policies are described by means of functions that implement import and export filters, similarly to real-world BGP configuration languages. Several important complexity results are proved: (*i*) checking if a BGP network has a stable routing (SOLVABILITY) is NP-complete, (*ii*) deciding whether a BGP network can be trapped in a permanent oscillation is NP-hard, and (*iii*) deciding whether a BGP network has a stable routing, i.e., it is solvable, under any combination of link failure is NP-hard. Observe that result (*iii*) is different from the ROBUSTNESS problem. In fact, checking if a network admits at least one stable state is different from checking whether a network is safe. In fact, a network with two or more stable states is proved to be unsafe [SSZ09]. Also, both these results may identify positive instances even if they are unsafe. Namely, in (*i*), a network with one or more stable states may be unsafe. In (*ii*), a network without strong routing oscillation may have weak routing oscillations. In (*iii*), checking if a network admits at least one stable state under any possible link failures is different from checking whether a network is safe under any link failures, as in result (*i*). Hence, SAFETY remains an open problem.

In [GSW02] the SPP model is introduced. BGP policies are expressed by explicitly enumerating and ranking all the permitted paths. In this setting, it is shown that SOLVABILITY is NP-hard. This result could not be evinced from [GW99], as translation from one model to the other might take exponential time. The complexity of SAFETY and ROBUSTNESS is left open.

In [SSZ09] it is proved that the coexistence of two stable states implies the existence of an oscillation. Policies are modeled with SPP. Although the model for BGP dynamics is slightly different from SPVP, the result also holds in SPVP [SSZ09].

In [JR04] a model is used in which BGP policies are applied consistently network-wide based on a classification of neighbors into groups. In this setting, a polynomial time algorithm is given to check whether the structure of the classes can lead to specific BGP policies in which oscillations are possible. The 3-SPP model is similar to the one used in [JR04] in that it also limits the expressiveness of BGP policies, however it is more general since it allows each AS to preserve its autonomy.

The rest of the literature studied BGP from a game-theoretic perspective, where ASes act as players and BGP policies as players' strategies. Messages between players are still allowed to be arbitrarily (even if not indefinitely) delayed. However, these approaches fail to capture specific BGP features either

in the game (*e.g.*, by assuming that routers can directly receive routes from non-neighbors [FP08]) or in the strategies (*e.g.*, by considering impossible strategies in BGP [JSW11, EFSW13]). In particular, In [FP08] SAFETY is claimed to be PSPACE-complete. However, the adopted model assumes that ASes are omniscient, that is, upon activation they can immediately know the AS-paths that are being used by every other AS, without the need to exchange BGP messages. This assumption makes it very hard to apply the results to any realistic model of BGP.

2.7 Conclusions

The design of BGP as a protocol where ASes interact in full autonomy poses a fundamental trade-off between the expressiveness of routing policies and risks of routing oscillations. Restricting the expressiveness of routing policies can be done either dynamically, e.g., by extending the protocol with oscillation-detection capabilities, or statically, e.g., by limiting the expressive power of BGP configuration languages. Unfortunately, the first option is affected by severe deployment issues. Prior contributions that explored the second option (e.g., [GR00]) devised restrictions on BGP policies that guarantee convergence, but affect both the autonomy and the expressiveness, e.g., by forcing ASes to filter certain paths.

In this chapter, we take a different approach, which can be summarized by the following question: assuming that ASes preserve their autonomy, how expressive can policies be in order to allow an efficient static assessment of BGP stability?

Unfortunately, we find that the most interesting problems about BGP stability are computationally intractable if ASes fully preserve their autonomy and are allowed to specify policies as expressive as Local Transit policies. Table 2.1 summarizes the results. While such results are primarily related to BGP, they can be generalized to any policy-based path vector routing protocol. Our findings show that computational tractability of BGP stability can be achieved by restricting the expressiveness of the policies alone, preserving ASes' autonomy. Determining whether there exist restrictions that keep the policies expressive enough for practical uses remains an interesting open problem. Also, we want to stress the fact that all the results, but SAFETY in the 3-SPP model, can be proved in the SAFETY version of the problem where the initial state is not constrained to be the 0-state. We believe that studying SAFETY in the 3-SPP model for every initial state can lead to new insight into the analysis of BGP

stability.

Chapter 3

Routing Policies and Logic Gates *

Because of its practical relevance, BGP has been the target of a huge research effort since more than a decade. In particular, many contributions aimed at characterizing the computational complexity of BGP-related problems. We described out contributions in Chapter 2. In this chapter, we further investigate the inherent difficulty of analyzing BGP dynamics. We unveil a fundamental mapping between BGP configurations and logic circuits. Namely, we describe simple networks containing routers with elementary BGP configurations that simulate logic gates, clocks, and flip-flops, and we show how to interconnect them to simulate arbitrary logic circuits. We then investigate the implications of such a mapping on the feasibility of solving BGP fundamental problems, and prove that, under realistic assumptions, *BGP has the same computing power as a Turing Machine*. This result is stronger than the one in Chapter 2, but it is stated under different message timing assumptions. We also investigate the impact of restrictions on the expressiveness of BGP policies and route propagation (e.g., route propagation rules in iBGP and Local Transit Policies in eBGP) and the impact of different message timing models. Finally, we show that the mapping is not limited to BGP and can be applied to generic routing protocols that use several metrics.

*Part of the material presented in this chapter is based on the following publications: M. Chiesa, L. Cittadini, Laurent Vanbever, S. Vissicchio, G. Di Battista. Using Routers to Build Logic Circuits: How Powerful is BGP?. In *Proc. ICNP*, IEEE, 2013.

3.1 Introduction and Related Work

We recall from Chapter 1 that BGP enables each AS to apply routing policies in complete autonomy, i.e., enabling each AS to fully control the routes that it accepts, prefers, and propagates to its neighboring ASes. While such a rich policy expressiveness can support complex business relationships, it can also cause routing and forwarding anomalies both in eBGP [GSW02] and iBGP [GW02] configurations.

Because of its practical relevance for Internet operation and its lack of correctness guarantees, BGP has been the focus of many research and industrial efforts in the last 15 years. Results of such an effort encompass formal analyses of the protocol (*e.g.*, [GSW02, GW02]), experimental measurements of disruptions due to BGP (*e.g.*, [MWA02, KKK07]), proposal of configuration guidelines (*e.g.*, [GR00]) and of protocol modifications (*e.g.*, [FR09]), and practical approaches to check a given configuration for correctness (*e.g.*, [CRV$^+$11, FMS$^+$10]). Refer to Chapter 2 for a more detailed description of BGP stability related work. However, all previous studies missed a fundamental analogy: Basic BGP configurations can encode elementary logic gates but also, memory and clock components. As such, BGP is powerful enough to encode logic circuits of arbitrary complexity, as we show in Section 3.2. We build this mapping assuming a simplified model (i.e. 3-SPP) for BGP routing policies which does not include advanced BGP features like MED or conditional advertisement.

In this chapter, we investigate the theoretical consequences of the existence of such a mapping between BGP configurations and logic circuits. We make the following four contributions.

First, we leverage the mapping to characterize the computational complexity of several routing problems in a "bounded" asynchronous model. Contrary to the previous chapter and previous works on BGP complexity, in this model each network link is associated with a network delay bounded between finite minimum and maximum values. This effectively imposes a partial order on the exchange of BGP updates. Previous lower bounds for BGP related problems have been proved in models that allow BGP messages to be arbitrarily (even if not indefinitely) delayed (e.g., see Chapter 2 and related work). In Section 3.3, we show that *BGP configurations can simulate arbitrary Turing Machines* in the considered bounded asynchronous model. Two implications derive from this observation. First, policy-based protocols like BGP intrinsically have the same computational power of Turing Machines, even when simple policies are considered. Second, it enables us to assess the computational intractability of BGP routing problems, like routing convergence and correct route propagation.

Second, in Section 3.4, we use the mapping to investigate the impact of policy restrictions on the complexity of BGP problems. We analyze both iBGP networks and eBGP policy configuration paradigms like the well-known Gao-Rexford conditions [GR00] and the widely used Local Transit Policies [GGSS09]. Also, we discuss the extent to which the mapping holds when other message timings are considered.

Third, in Section 3.6, we show that our methodology can be applied in a routing framework that is different from BGP, and we investigate how difficult the analysis of a generic routing protocol using several metrics is.

Finally, we prove that our approach can be used in several message timing models in Section 3.5. In particular, we show that the complexity of unstudied routing problems can be assessed in the bounded asynchronous models [GSW02] by relying on the mapping between BGP configurations and logic gates.

3.2 BGP Configurations as Logic Circuits

The most prominent feature of BGP is the support for routing policies that can be independently defined on each BGP router. Routing policies are used to specify which routes should be accepted from (or announced to) which neighbors, and to assign different degrees of preference to different routes.

In this chapter, we rely on the well-known SPP formalism [GSW02] to model eBGP configurations (in Section 3.4 we use a similar model to represent iBGP configurations). In SPP, an eBGP configuration is represented as a graph where every node is an Autonomous System (AS) and every edge is an eBGP peering. Since BGP routers treat different destinations separately, we focus on one destination at the time. The destination is represented by a special node d, to which all other nodes try to establish a route. A route is a simple path on the graph. Each node can specify its own policy, which is modeled as a list of all the routes that the node accepts towards the destination. The order of the elements in the list corresponds to the preference of the node.

Despite the fact that SPP is a simplified model for BGP policies (it does not capture `MED`s and conditional route announcements, just to name a few), in this section we show that SPP instances representing eBGP configurations can emulate any logic circuit. To achieve this result, we show how elementary BGP configurations can replicate the behavior of AND, OR and NOT gates. In particular, we map the logic signals 1 and 0 to the absence or presence of a BGP route, respectively. Two elements are nonetheless missing to build a

Turing machine: the ability to store information in the BGP configuration (i.e., a memory) and a clock. To build those, we leverage two peculiar configurations that have troubled network operators and the IETF community for over a decade. The first one is a BGP wedgie [TG05], a BGP configuration that has two stable states. In our construction, we map one of its state to 0 and the other one to 1. Like a flip-flop memory, this BGP configuration may oscillates between these two stable states. The second configuration [MGWR02, WRCS13] is a perpetual BGP oscillation that emits a BGP route at a known frequency.

Elementary Logic Gates with eBGP

We now show how to build eBGP configurations that simulate the OR and NOT logic gates. We map the inputs (outputs, resp.) of a logic gate to a set of *input nodes* (*output nodes*, resp.) of the SPP instance. Also, we map the availability of a route to a 1 (true value) and the absence of a route to a d (false value). In particular, the availability (absence, resp.) of a route at an output node r at time t means that the output signal of r at time t is 1 (d, resp.).

The eBGP configurations simulating the OR and the NOT logic gates are shown in Fig. 3.1 and Fig. 3.2, respectively. The graphical convention we use in the figure is adopted throughout the chapter, unless differently specified. ASes are represented by circles, and solid edges represent eBGP peerings. A list of paths is specified beside each AS. Each list contains the paths that the AS accepts (i.e., paths that are not filtered out by the routing policy) in a descending order of preference. All routes refer to the same destination d. We use dots inside a path when we do not specify the entire path, so $(a\ b \dots d)$ represents a path that start at a, traverses b and ends at d. Incoming and outgoing dashed arrows indicate input and output nodes, respectively. For the sake of brevity, whenever it is clear from the context, we omit node d and its peerings. For example, in Fig. 3.2, d should be considered directly attached to a, b and c.

Fig. 3.1 represents an eBGP configuration corresponding to the OR gate. Nodes i_1 and i_2 correspond to the inputs and node o_1 corresponds to the output. Since node o_1 only accepts routes from r, it will have a route to d if and only if either i_1 or i_2 has a route to d. Similarly, Fig. 3.2 represents an eBGP configuration simulating the NOT gate, where i_1 is the input and o_1 is the output. In this configuration, o_1 has a route to d if and only if i_1 has no route to d. Indeed, if i_1 has a route to d, r receives and selects the route from i_1 instead of the route from b (that is always available at r) because of its preferences. Thus, o_1 will end up with no route, since o_1 does not accept

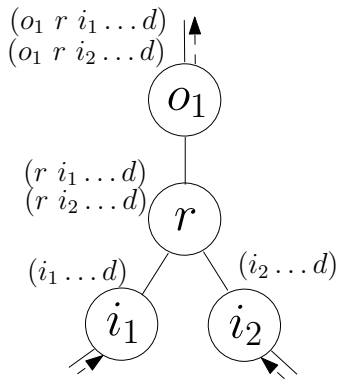

Figure 3.1: The OR gate.

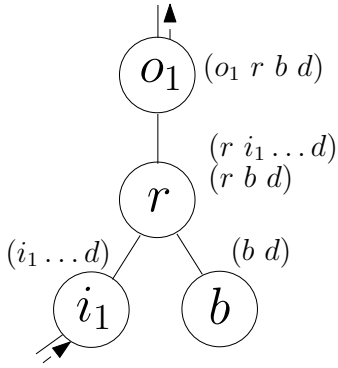

Figure 3.2: The NOT gate.

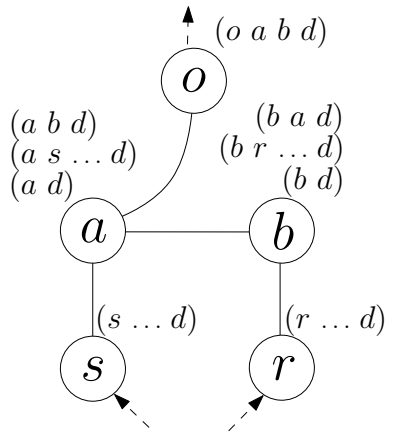

Figure 3.3: A DISAGREE with two extra nodes S and R behaves as a flip-flop.

path $(o_1\ r\ i_1 \dots d)$, as shown by the absence of the path in the list aside o_1 in Fig. 3.2). On the contrary, if i_1 has no route to d, then r selects $(r\ b\ d)$ and, consequently, o_1 selects $(o_1\ r\ b\ d)$.

Memory and Clock with Popular eBGP Gadgets

Besides encoding elementary gates, eBGP is powerful enough to simulate more complex logic components, like flip-flops and clock generators.

Fig. 3.3 shows an eBGP configuration that simulates an SR flip-flop. This flip-flop has two inputs S (set bit) and R (reset bit) and one output Q. The flip-flop stores and outputs a 1 (d, resp.) whenever the set (resp., reset) bit is set to 1 (d, resp.). If both set and reset bits are set to d, then output Q is the stored value. Setting both S and R to 1 is not allowed. The configuration in Fig. 3.3 simulates this behavior. It is based on the presence of a well-known BGP gadget, called DISAGREE [GSW02], that has two stable states. Indeed, nodes a and b form a DISAGREE. In one stable state, nodes a and b select paths $(a\ b\ d)$ and $(b\ d)$, respectively. In the other one, nodes a and b select $(a\ d)$ and $(b\ a\ d)$, respectively. Depending on whether nodes s and r receive a route, we have the following three cases. If s announces a route to a and r does not announce any route to b, then a never selects $(a\ d)$, since path $(a\ s \dots d)$ is available and more preferred than $(a\ d)$. Hence, b has to select $(b\ d)$ because none of $(b\ a\ d)$ and $(b\ r\ \dots\ d)$ is available. Since a receives $(b\ d)$, it can select

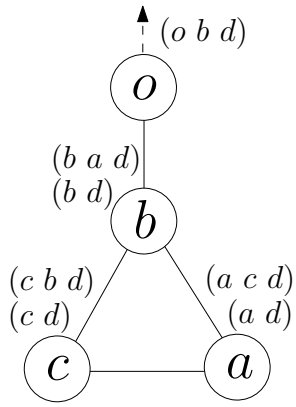

Figure 3.4: A BAD-GADGET plus output node o behaves as a clock.

its best path $(a\ b\ d)$. Symmetrically, if r announces a route to b and s does not announce any route to a, then a has to select $(a\ d)$. Finally, if neither s nor r receives a route, then the DISAGREE does not change its stable state. As a consequence, node o has an available path to d if and only if node a selects path $(a\ b\ d)$, hence mirroring the output of an SR flip-flop.

Further, the dynamics of eBGP configurations that admit no stable state are conceptually similar to those of clock generators. A *clock generator* is a logic circuit producing a signal that oscillates between 1 and d. The BAD-GADGET [GSW02], shown in Fig. 3.4, is a gadget that never converges to a stable state. It consists in a cycle of three nodes a, b, and c, in which each node prefers a route through its successor instead of a direct route to d. When the gadget oscillates, node a alternatively selects paths $(a\ c\ d)$ and $(a\ d)$. Since o does not accept path $(a\ c\ d)$ from a, o has a route only when node a selects $(a\ d)$. Therefore, the output node o will alternate forever between having a route and not having any route, as for a clock generator. Observe that the clock of Fig. 3.4 can be thought in terms of the circular interconnection of 3 NOT gates of Fig. 3.2.

Simulating Arbitrary Logic Circuits

Now that we have the elementary logic components, it would be tempting to simply interconnect them using eBGP peerings. Such an operation is needed

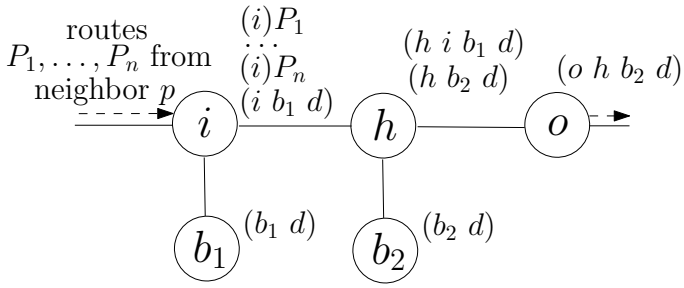

Figure 3.5: The HUB gadget we use to interconnect logic components.

for building: (i) the AND gate, using OR and NOT and applying the De Morgan's laws; (ii) arbitrary logic gates as a combination of AND, OR, and NOT; and (iii) arbitrary logic circuits starting from logic gates, flip-flops and clocks. Unfortunately, because of BGP peculiarities, arbitrary interconnections are not straightforward.

The first problem we face is that signal propagation in logic circuits has a direction, while routes may traverse an eBGP peering in both ways. We need to prevent routes from being propagated in unintended directions, e.g., "signals" traversing the gates from their output to their input. This can be accomplished by using eBGP policies to accept only routes in the intended direction.

A second and more subtle problem arises with loops. BGP has a built-in control plane loop prevention mechanism [RLH06] which mandates an AS to discard routes containing its own identifier. Because of this mechanism, we need an additional building block to be able to simulate logic circuits where the signal is propagated through a loop. In particular, we interpose a special gadget, called HUB gadget, between any pair of interconnected logic components.

The HUB gadget is in Fig. 3.5. Intuitively, it takes a route at its input node i and generates a new, completely different route at its output node o. It can be seen as the concatenation of two NOT gadgets, in which the first NOT gadget filters out the original route and the second NOT gadget generates the new one. No route is produced at the output if i receives no external route. In other words, the HUB gadget is able to correctly propagate both the presence of a route (a binary 1) and the absence thereof (a binary d). Nodes h, b_1 and b_2 are different for each HUB gadget and therefore cannot appear in any external route received by i. This guarantees that the output route cannot share any node (besides d) with the input route, which in turn keeps BGP's

loop prevention mechanism from being triggered. More precisely, if i receives no route from its neighboring node p, i selects route $(i\ b_1\ d)$. This allows h to select its preferred path $(h\ i\ b_1\ d)$, which in turn makes o unable to select any valid route to d. Otherwise, if p advertises a route $(p \dots d)$ to i, then i selects $(i\ p \dots d)$. As a consequence, h selects $(h\ b_2\ d)$, and o selects $(o\ h\ b_2\ d)$.

Observe that the output node of the HUB gadget can either announce no route, or it can announce a single route which does not depend on the route received by the input node. For this reason, when connecting the output node o of the HUB gadget to the input node i' of another gadget, i' can receive only one route. Given a logic circuit, w.l.o.g. composed by NOT and OR gates, we can replace each gate with the gadgets of Sect. 3.2 and replace each wire with a HUB gadget obtaining a BGP network computing the same function. Since the gadgets and the HUB have a constant number of routers, each requiring a constant size routing configuration, we have that the construction is done in polynomial time.

Restricting our attention to combinational logic circuits, we have a first interesting consequence of our constructions. Since a combinational logic circuit can encode any logic formula, the fact that eBGP can be used to construct combinational logic circuits gives new intuition of why most problems related to BGP are NP-hard. In fact, by encoding a logic formula in BGP, it is typically possible to obtain a polynomial reduction from SAT [Pap94], a well-known NP-complete problem.

3.3 Understanding the Complexity of BGP Using Logic Gates

The fact that eBGP configurations can simulate logic circuits has several implications in terms of the computational complexity of routing problems. To deep out investigation in this direction we have to model BGP dynamics. We consider a *bounded* asynchronous model, where messages traversing a link have a propagation delay between a minimum and a maximum value. A link propagation delay encompasses many delay elements of a BGP network (e.g., physical delays, congestion of a link, MRAI timer) whose values range between a minimum and a maximum. We recall that since BGP updates travel into TCP connections, packet loss and out-of-order packets are not an issue.

Building a Turing Machine with Logic Gates

In the bounded BGP model, each link l is associated with a propagation delay that can take any value within range (m_l, M_l), where m_l (M_l, resp.) is the minimum (maximum, resp.) delay value for l. Both m_l and M_l are finite values different from d. Observe that, if $m_l = M_l$ all BGP message exchanges are completely synchronized. In general, however, we assume $m_l \neq M_l$. Analogously, we can define a bounded model for logic circuits associating a minimum and a maximum propagation delays on each wire.

We recall from Chapter 1 that a BGP network is *safe* if, for each possible execution, the network converges to a stable state. SAFETY is defined as the problem of checking if a BGP network is safe. A BGP network *oscillates* if it is not safe. A logic circuit *halts* if, for each timing, there is a time instant when (i) for each link its endpoints have the same value and (ii) for each gate, its output value is the correct output with respect to the current gate inputs. A logic circuit *oscillates* if it does not halt.

Using standard circuit design methodologies for the bounded model (e.g., [Fri01]) we can use logic gates in the bounded model to construct a *Finite Turing Machine* (FTM) [Lin06], i.e., a Turing Machine where the size of the tape is finite. More details on such a construction are reported at the end of the chapter in Section 3.8. An FTM is a simple device that reads and writes symbols on a finite tape according to a table of rules. This enables us to show that most BGP routing problems are at least as difficult as the "halting" problem for an FTM, which is known to be PSPACE-hard [GJ79]. Hence, we have the following lemma.

Lemma 3.1 *Given a Finite Turing Machine M, it is possible to construct in polynomial time a logic circuit C in the bounded model such that C halts iff M halts.*

The discussion of Section 3.2 shows that using eBGP we can construct logic circuits. Since we are using a BGP model with bounded delays, then we can simply assign the desired delays to BGP peerings. Hence, exploiting Lemma 3.1, we have the following theorem.

Theorem 3.2 *Given a Finite Turing Machine M, it is possible to construct in polynomial time an eBGP network N in the BGP bounded model such that N converges to a stable state iff M halts.*

The ability to simulate FTMs with eBGP configurations enables us to prove PSPACE-hardness results for BGP problems. We reduce those problems from

the LINEAR SPACE ACCEPTANCE problem, which is known to be PSPACE-complete [GJ79]. An instance of LINEAR SPACE ACCEPTANCE consists of a FTM M and a finite string x, where the size of the tape of M is linear with respect to the size of x. The problem is to verify if M accepts x. We say that an FTM M accepts a string x if M halts on an acceptance state given that x is initially written on its tape.

In the following, we prove that both SAFETY [GSW02] and REACHABILITY [GW99] are PSPACE-hard. The PSPACE class contains all problems that can be solved by a TM using a tape of polynomial length w.r.t. the size of the input string [Pap94] and, in turn, it contains all problems in the NP class. PSPACE-hard problems are the representative problems of the PSPACE class, i.e., if a PSPACE-hard problem can be solved in polynomial time, then every problem in PSPACE (and in NP) can be solved in polynomial time. Since NP is believed to be a proper subset of PSPACE, PSPACE-hard problems are considered to be harder than NP-hard problems. For instance, SAT solvers [PBG05], which are a practical tool used to deal with NP-hard problems, cannot be used for PSPACE-hard problems.

Theorem 3.3 SAFETY *is PSPACE-hard.*

Proof: We reduce SAFETY from the LINEAR SPACE ACCEPTANCE problem. A similar construction with respect to that described above enables to build an eBGP configuration that simulates an arbitrary FTM M in such a way that the network converges to a stable state if and only if M reaches an acceptance state. This polynomial-time reduction directly yields the statement. □

In [FP08] BGP SAFETY is proved to be PSPACE-complete in an unrealistic game-theoretical model in which BGP speakers are assumed to be omniscient and BGP messages are not passed router by router (i.e., router receives routing update as in a link-state protocol). We stress the fact that this unrealistic assumption fails to capture the communication model of BGP in which messages are exchanged as in a distance-vector protocol.

A very similar reduction from LINEAR SPACE ACCEPTANCE can be leveraged to show the complexity of the REACHABILITY problem [GW99], that is, deciding whether a BGP configuration admits a stable state in which a given node s has a route to a given destination d. Namely, it is sufficient to build an BGP configuration that simulates the FTM M as shown in the proof of Theorem 3.3 and modify it such that node s is guaranteed to have a route if and only if the BGP gadget simulating the clock of the FTM has stopped oscillating.

Observe that, theoretically, an infinite BGP network would be able to simulate a Turing Machine, where each cell of the infinite tape is modeled by a certain number of routers. In a sense this means that, despite the simplifications listed in Section 3.2, unrestricted BGP policies have the same expressive power as Turing Machines. As a consequence, since the halting problem for a TM is undecidable [Wol02], also SAFETY would be an undecidable problem for an infinite BGP network.

3.4 The Impact of Policy Restrictions

As in Chapter 2, we study the impact of this mapping between logical gates and router configurations under different restrictions to the routing policy expressiveness. Intuitively, the fact that BGP configurations can encode arbitrary logic circuits suggests that the complexity of BGP related problems stem out of the intrinsic complexity of BGP semantics, which ultimately maps to the expressiveness of BGP policies. One might argue that it is not surprising that completely unrestricted policies yield complex semantics. It is therefore interesting to study whether restricting BGP policies significantly simplifies the analysis of a BGP configuration.

In this section, we consider iBGP configurations, where policies are dictated by the iBGP route propagation rules and distances derived from the intradomain routing protocol, called IGP distances. We also consider Local Transit policies, where policies depend solely on the ingress and egress AS and Gao-Rexford conditions, where policies are tied to commercial relationships among ASes.

Restricting to iBGP

BGP [RLH06] comes in two flavors: external BGP (eBGP) and internal BGP (iBGP). eBGP is used to exchange reachability information between neighboring networks or Autonomous Systems (AS), while iBGP is used to distribute externally-learned routes within an AS. As opposed to eBGP, in iBGP all routers belong to the same AS. For this reason, routing policies are typically not applied on iBGP messages [GW02]. However, the specification of iBGP with route reflection [BCC06] imposes an implicit route ranking and an implicit route filtering. The ranking component is restricted in that it has to be consistent with the IGP graph. The filtering component is restricted in that it has to be consistent with the iBGP graph.

In particular, the BGP decision process has some tie breaking rules that only have local significance (e.g., IGP distance), therefore routing preferences are implicitly imposed by the BGP decision process itself. Further, when route reflection is used, the protocol requires certain routes to be filtered out at each iBGP router. More precisely, the iBGP neighbors of each router are split into three sets: *clients*, *peers* and *route-reflectors*. Best routes are always relayed to clients, but best routes learned from peers or route-reflectors are not propagated to other peers and route-reflectors.

We now show that, despite the restrictions above, the protocol still retains enough expressive power to encode arbitrary logic circuits. First of all, observe that we can consider just egress points preferences, disregarding the details of the IGP graph. In fact, in the following we show that for any given set of egress point preferences there exists an IGP graph which is consistent with those preferences. Consider a single destination d. Let $\mathcal{E}$ be the set of egress points to d. Let $\lambda^v : \mathcal{E} \to \mathbb{N}$ be the egress point ranking function of router v, such that $\lambda^v(e_j) = i$ if and only if e_j is the i-th most preferred egress point by v. Since the BGP process forces each router to deterministically select only one route, egress point preferences at each router are totally ordered, that is, $\forall e_i \neq e_j \ \lambda^v(e_i) \neq \lambda^v(e_j)$. Given the ranking functions of all the routers in the networks, the following algorithm, that we call IB, builds an IGP graph which is consistent with those ranking functions as follows:

(i) Create an empty IGP graph where the nodes are the routers and the egress points of the network.

(ii) For each pair of a router r and an egress point e, add a link (r, e) in the IGP graph.

(iii) To each link (r, e), assign a weight $w(r, e) = \lambda^r(e) + |\mathcal{E}|$.

We now prove that the IB algorithm is correct, that is, it builds an IGP graph consistent with the given egress point preferences at each router. Let $dist(v, u)$ be the length of the shortest path from v to u. We say that an IGP topology *realizes* the given ranking functions if for each router $v \notin \mathcal{E}$ and each arbitrary pair of distinct egress points e_1 and e_2, $\lambda^v(e_1) < \lambda^v(e_2)$ implies that $dist(v, e_1) < dist(v, e_2)$.

Lemma 3.1 *In the IGP topology built by the* IB *algorithm, the shortest path between any router r and any egress point e is $(r\ e)$.*

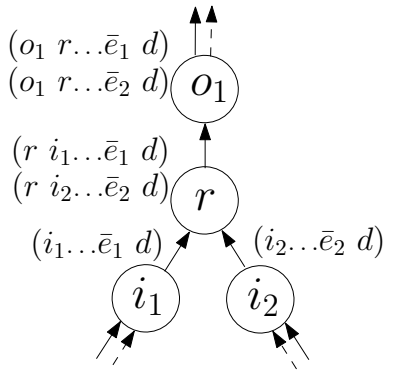

Figure 3.6: The OR gate in iBGP.

Proof: Let $G = (V, E)$ be the IGP topology built by the IB algorithm. Consider any router r and any egress point e. By construction, the weight of the path $(r\ e)$ is equal to $w(r, e) = \lambda^r(e) + |\mathcal{E}| \leq 2|\mathcal{E}|$. We now show that any path P from r to e, with $P \neq (r\ e)$, has a weight higher than $w(r, e)$. By definition of P, P contains at least two edges. By definition of the weight function adopted in the IB algorithm, the weight of P is equal or greater to $2 + 2|\mathcal{E}|$. Hence, the weight of any path $P \neq (r\ e)$ is higher with respect to $(r\ e)$, yielding the statement.

□

We are ready to prove the following theorem on the correctness of our construction.

Theorem 3.2 *Given a set Λ of ranking functions, the* IB *algorithm builds an IGP topology that realizes Λ.*

Proof: Let $G = (V, E)$ be the IGP topology built by the IB algorithm. Consider a router r and any pair of egress points e_1 and e_2, such that r prefers routes from e_1 over routes from e_2. By Lemma 3.1, $dist(r, e_1) = w(r, e_1)$ and $dist(r, e_2) = w(r, e_2)$. By definition of the weight function used in the algorithm, we have $w(r, e_1) < w(r, e_2)$, which proves the statement.

□

Exploiting the IB algorithm, we now show how to construct OR and NOT gates with iBGP configurations, as shown in Fig. 3.6 and Fig. 3.7, respectively.

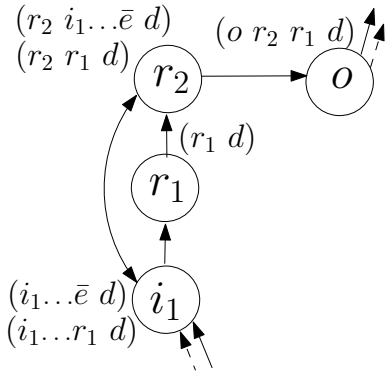

Figure 3.7: The NOT gate in iBGP.

In these figures, one-headed solid arrows represent sessions from a client to its route reflector, while double-headed solid arrows represent sessions between two peers. Inbound and outbound dashed arrows indicate input and output nodes, respectively. Paths aside each router represent the iBGP path towards egress points. The rest of the notation is consistent with the graphical convention introduced in Section 3.2.

The iBGP configuration in Fig. 3.6 simulates the behavior of an OR logic gate. The output router will receive a route to any of the egress points $\bar{e}_1$ and $\bar{e}_2$ if and only if a route is received by either i_1 or i_2 (or both). Similarly, Fig. 3.7 depicts an iBGP configuration corresponding to the NOT gate. If i_1 receives an eBGP route, it will propagate it to r_2. Because of egress point preferences, r_2 will select the route announced by i_1. Now, since this route was learned from a peer, iBGP route propagation rules require that r_2 do not relay the route to o, which therefore is unable to learn any feasible route. On the contrary, if i_1 receives no route, r_2 will select the route announced by r_1 and will propagate it to o. Note that the iBGP configuration corresponding to the NOT gate is based on the Over-Ride gadget introduced in [VCVB12].

Reference [GW02] shows examples of iBGP configurations realizing Disagree and Bad-Gadget structures. This enables us to build the memory and the clock components as we did for eBGP in Section 3.2.

Finally, to interconnect logic components, we use the iBGP-Hub gadget (see Fig. 3.8), which is the equivalent of the Hub gadget for iBGP. If i receives an iBGP path R towards any egress point $\bar{e}_j$, then i, c and x also select route

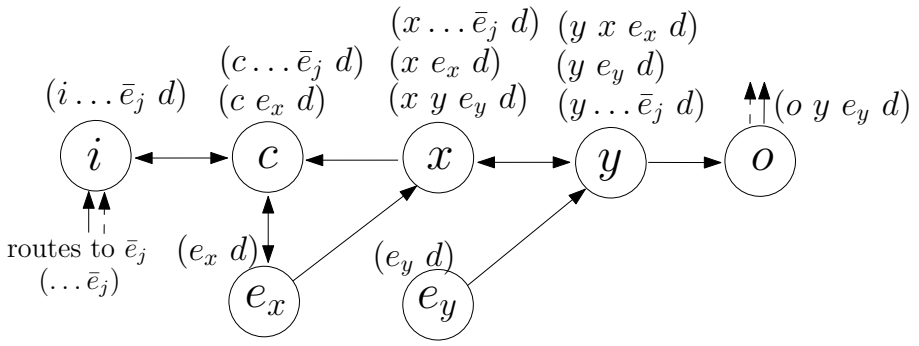

Figure 3.8: The iBGP-Hub gadget.

R. In this case, y selects path $(y\ e_y)$ because of its ranking function, and propagates it to o. Hence, o receives, selects and propagates one route which has no router in common with the original route R. Otherwise, if i receives no path towards any $\bar{e}_j$, then x selects route $(x\ e_x)$, enabling y to select its most preferred route $(y\ x\ e_x)$. However, y cannot propagate $(y\ x\ e_x)$ to o because of iBGP propagation rules that deny propagation of a path learned from an iBGP peer to a route reflector.

Observe that the iBGP-Hub gadget outputs at most one route, and has at most two routes in input. Also, node i cannot receive paths from c, which implies that routes can only flow from the left to the right part of the gadget, hence preventing propagation of routes in undesired directions. In fact, either i) node i selects a path $R = (i \ldots \bar{e}_j)$ which is learned over the client session itself; or ii) node i has no path to select as c's best route $(c\ e_x)$ is learned from an iBGP peer and cannot be propagated to another iBGP peer.

Having all the needed building blocks, an iBGP configuration that simulates a Finite Turing Machine can be done exactly as in Section 3.3 for eBGP configurations, again exploiting Lemma 3.1. As a consequence, we can derive new intractability results (PSPACE-hardness for the min-max model) of all correctness problems defined in iBGP, namely signaling, dissemination and forwarding correctness [GW02, VCVB12].

Local Transit Policies and Gao-Rexford Conditions

We now consider eBGP policy restrictions that have been proposed in the literature.

A common policy configuration practice consists in applying the so-called Local Transit policies [GGSS09]. Local Transit policies consist in defining routing policies as functions of the AS that announces the route and of the AS to which the route is announced only. Observe that all the policies used to build the gadgets presented in Section 3.2 are compliant with the definition of Local Transit policies. The same holds for the DISAGREE gadget and the BAD-GADGET. As a consequence, BGP problems remain hard (in the unbounded asynchronous model) or very hard (in the bounded asynchronous model) even for BGP networks in which only Local Transit policies are applied. These results extend the findings in [CCDV11].

A further restriction with respect to Local Transit Policies consists in imposing that the eBGP configuration satisfies the Gao-Rexford conditions introduced in [GR00]. These conditions are the most famous way to trade policy expressiveness for correctness guarantees without the need for global coordination among ASes. Gao-Rexford conditions assume that each AS classifies its eBGP neighbors as either customers, peers, or providers, and that: i) routes learned from customers are preferred over those learned from peers and providers; ii) there is no cycle such that each AS in the cycle is a customer of the next AS in the cycle; iii) an AS does not export routes learned from a peer or provider to its peers or providers. It has been proved [GR00] that the Gao-Rexford conditions guarantee that BGP always converges to a unique stable state and a greedy algorithm proposed in [GSW02] can be used to compute the stable state, and to solve BGP problems in polynomial time.

The BGP networks simulating the OR and NOT logic gates (Fig. 3.1 and Fig. 3.2, respectively) are compliant with Gao-Rexford conditions if the output node o_1 is set as provider of r, and r is a provider of all the other nodes. Similarly, the HUB gadget (Fig. 3.5) is compliant with the Gao-Rexford conditions if o is a provider of h, h is a provider of both i and b_2, and i is a provider of both p and b_1. This assignment of commercial relationships has the property that if a logic circuit does not contain cycles (as in combinational circuits) then it can be simulated by a BGP network that satisfies the Gao-Rexford conditions. Otherwise, cycles in the logic circuits translates to customer-provider cycles in the eBGP configuration, which violates the second Gao-Rexford condition. However, assuming Gao-Rexford conditions prevents us from building arbitrary logic circuits and configurations like a DISAGREE or a BAD-GADGET. This can

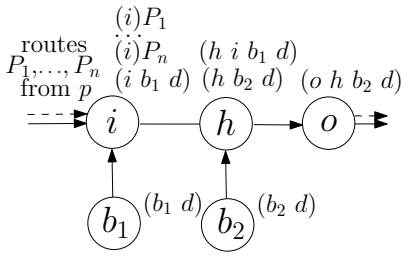

Figure 3.9: A HUB violating Condition i).

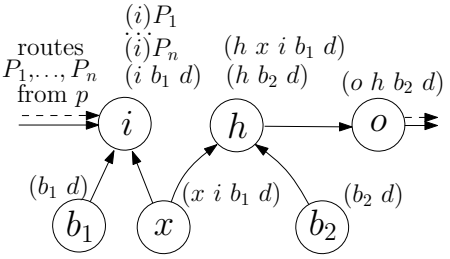

Figure 3.10: A HUB violating Condition iii). Variants of the HUB gadget obtained by violating one of the Gao-Rexford conditions. An oriented (unoriented) edge from a to b represents the fact that a is a customer (peer) of b.

be seen as an intuitive explanation of why most BGP problems turn out to be polynomial in such a setting.

However, violating any of the Gao-Rexford conditions enables us to build configurations that simulates arbitrary logic circuits, hence arbitrary Finite Turing Machines in the min-max model. We have already shown a customer-provider assignment such that a cycle in a logic circuit translates to a customer-provider cycle in the BGP network. Hence, if we violate condition ii) and customer-provider loops are allowed, then every interconnection between logic components is admitted. Otherwise, if conditions i) or iii) are violated, we modify the HUB gadget as shown in Fig. 3.10. In each of these two cases, cycles in the logic circuits are guaranteed not to translate to customer-provider cycles, and only one of the Gao-Rexford conditions is violated at the time. Hence, for any violation of the Gao-Rexford conditions, arbitrary logic circuit

can be simulated with BGP configurations. As a consequence, convergence and route propagation problems are still PSPACE-hard if any of the Gao-Rexford condition is violated.

3.5 Extending the Approach to Different Delay Models

In this section, we show that our mapping technique can be used in a delay model different from the bounded one. Namely, we use the same dynamic model used in Chapter 2 based on fair activation sequences, where there is no lower and upper bound on the link delays. In this section, we refer to this model as the *unbounded* model. Since Lemma 3.1 holds in the bounded model, all the proofs derived in Sect. 3.3, Sect. 3.4, and Sect. 3.6 for building an FTM do not directly extend to the unbounded one. In fact, as proved in [Mar90], it is not possible to construct a circuit with memory elements using logic gates in an unbounded model. Hence, it is not possible to build a FTM. Anyway, this limitation does not prevent our mapping to be used in order to simply prove new complexity result on BGP (Theorem. 3.1). Moreover, as a by-product of our study, we also discovered that many already known results from [GW02] and [GW99] can easily be proved with our technique (Appendix 3.5).

For instance, consider the following problem, called MOAS Reachability. Assume that a destination prefix is generated by multiple origin ASes in the eBGP network, as it happens when IP anycast is deployed in the Internet (e.g., for the DNS root name servers). The MOAS Reachability problem consists in determining if the destination prefix is reachable from a given source AS for any nonempty subset of origin ASes announcing the prefix. Such a problem aims at verifying that reachability of a given MOAS (Multiple Origin AS) destination is guaranteed in presence of failures or planned maintenance. Leveraging the mapping between BGP networks and logic gates, it is easy to prove the following theorem.

Theorem 3.1 MOAS Reachability *is coNP-hard.*

Proof: The scheme of the reduction from SAT COMPLEMENT [Pap94] is represented in Fig. 3.11, where nodes labeled as d_i, with $i = 1, \ldots, n$, represent the origin ASes announcing the destination prefix, and N, F, and s are defined as follows. Let F be the boolean formula in conjunctive normal form that represents an instance of SAT. Let s be a vertex that is the given source AS. We interpose between them a BGP network N which simulates the logic circuit corresponding to F. Now, considering the combination of origin ASes from

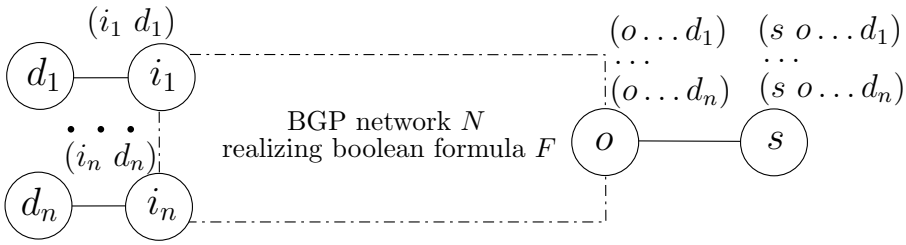

Figure 3.11: Scheme of the reduction from SAT COMPLEMENT to MOAS Reachability.

which the prefix is announced corresponds to providing all possible inputs to the network N. By construction of N, this translates to considering all boolean assignments to variables in the original boolean formula F. Hence, s receives a route for each combination of origin ASes announcing the destination prefix if and only if the boolean formula F is satisfied by any boolean assignment. The statement of the theorem follows from the coNP-hardness of SAT COMPLEMENT and the fact that the reduction can be built in polynomial time with respect to the size of F. Indeed, given that the number of clauses in F is C, the number of origin ASes d_i is equal to C, and each gate in N has a constant number of nodes, each accepting at most $2 * C$ paths, because of the presence of Hub gadgets at each interconnection between gates. □

Observe now that the above theorem holds also in the case Gao-Rexford conditions are enforced. In fact, as already discussed in Sect. 3.4, if there are no cycles in the logic circuit, it is possible to map it to a Gao-Rexford compliant BGP configuration. Hence, since the only circuit used in the proof of Thm. 3.1 is combinational, the MOAS Reachability problem remains NP-hard even when Gao-Rexford conditions are enforced. The NP-hardness of MOAS Reachability under Gao-Rexford conditions is especially interesting, because other BGP problems are polynomial in such a setting [GR00].

We stress that, by applying the same reduction technique, it is also straightforward to build reductions for problems like Reachability, Solvability, Trapped, Unique, and Multiple [GW99].

Reductions for the Unbounded Delay BGP Model

In the following, we show how to leverage the mapping between eBGP configurations and logic gates to prove the complexity of the problems studied in [GW99] in the unbounded asynchronous model. All the following reductions can be built in polynomial time with respect to F (see Section 3.2). Also note that a DISAGREE can be obtained as a loop of two NOT gadgets. As a consequence, the following complexity proofs remain valid whenever the OR and the NOT logic gates can be simulated and arbitrarily interconnected via the HUB gadget.

Reachability, Solvability, and Trapped

The REACHABILITY problem consists in deciding if a BGP network admits a stable state in which a given router s has a route to a given destination d. The problem has been already shown in [GW99] to be NP-hard. We now show an NP-hardness proof exploiting the mapping between BGP configurations and logic gates.

Theorem 3.2 REACHABILITY *is NP-hard.*

Proof: Let F be a boolean formula in conjunctive normal form that represents an instance of SAT. We build an instance of REACHABILITY as follows (see Fig. 3.12). Let o and d be the source and the destination vertices considered in REACHABILITY, respectively. We interpose between them a BGP network N which simulates the logic circuit corresponding to F. We also add a DISAGREE gadget between d and each input router i_j in N. Each DISAGREE gadget can converge to one of two distinct stable states. Each possible combination of these stable states is mapped to a boolean assignment M of the variables. For each of these combinations, by construction of N, node o will have a route to d if and only if F is satisfied by M. □

We now consider the SOLVABILITY and TRAPPED problems that deal with convergence guarantees of a BGP configuration. Namely, SOLVABILITY consists of deciding if a given BGP configuration admits at least one stable solution, while TRAPPED is the problem of deciding if the network can be trapped in permanent routing oscillations, like those occurring in a BAD-GADGET. For both problems, we use the reduction shown in Fig. 3.12, where a BAD-GADGET between nodes p_1, p_2, and p_3 is added. The BAD-GADGET does not oscillate if and only if p_2 steadily receives path $(o \dots d)$ from o. However, since this

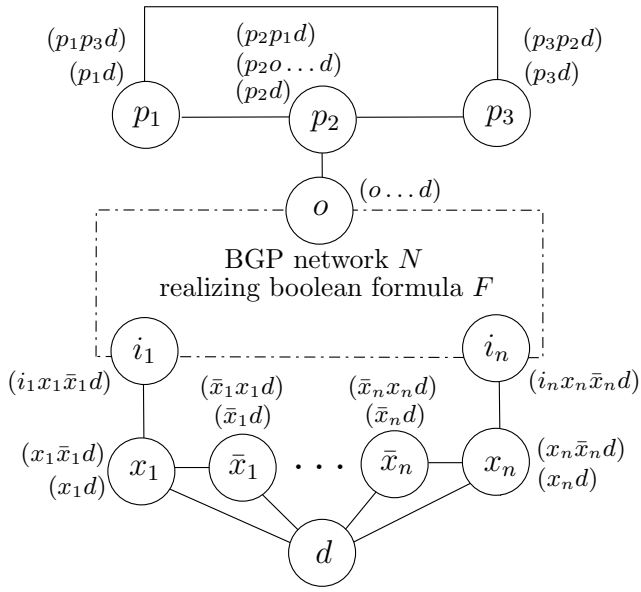

Figure 3.12: Construction for REACHABILITY, SOLVABILITY, and TRAPPED.

requires to solve the REACHABILITY problem, both SOLVABILITY and TRAPPED are proved to be NP-Hard.

Unique and Multiple

UNIQUE (MULTIPLE) is the problem of deciding if a single (more than one) stable state exists for a given BGP configuration. In the reduced instance in Fig. 3.13 the presence of a stable state is guaranteed in one of the two stable states of the DISAGREE between a and b. Indeed, if a steadily selects $(a\ b\ d)$ and b steadily selects $(b\ d)$, then no route is provided to c, which implies o having no route, and e selecting $(e\ b\ d)$. This, in turn, implies nodes f, g, and h having no route to d. However, if a steadily selects $(a\ d)$ and b steadily selects $(b\ a\ d)$, then the presence of additional stable states depends on the formula F. Indeed, if F is not satisfiable then o has no steady route. In this case, e selects $(e\ d)$ activating the BAD-GADGET between f, g, and h. As a result, no other stable state exists. Otherwise, if F is satisfiable, then o steadily selects a route for at least one combination of inputs to N. This implies that e selects

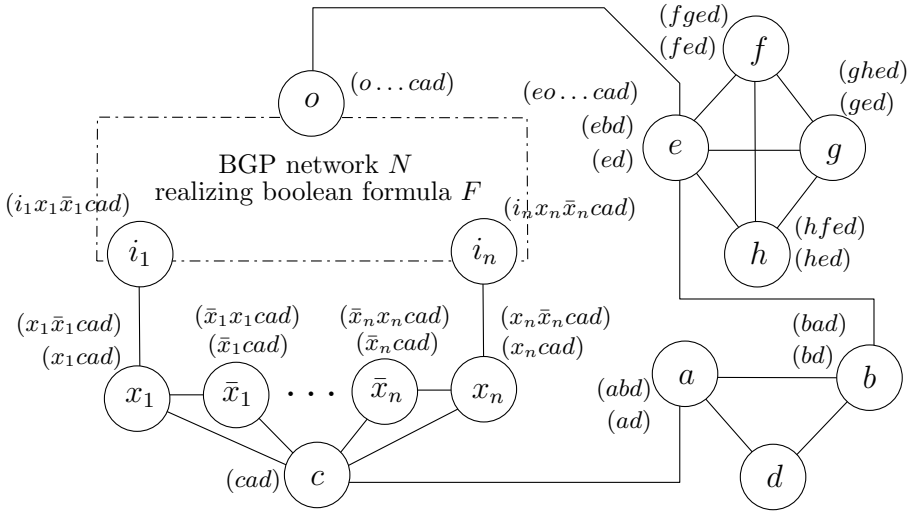

Figure 3.13: Scheme of the reduction from SAT to UNIQUE and MULTIPLE.

the route it receives from o, hence preventing the BAD-GADGET between f, g, and h from oscillating, and forcing a second stable state. This proves UNIQUE is NP-Hard, which directly implies the NP-hardness of MULTIPLE, as already noted in [GW99].

3.6 Combining Popular Metrics can be Hard to Analyze

In Sect. 3.2 and 3.4, we built a mapping from BGP to logic gates. This mapping exploits two specific BGP constructs: per-neighbor filtering and a ranking function that is based on the presence of a specific vertex in the path (i.e., per-neighbor and per-egress-point ranking in eBGP and iBGP, respectively). One may wonder whether this mapping technique can be applied to protocols different from BGP. In this section, we answer this question. We consider distance-vector routing protocols where each routing message contains a vector of metrics (as in EIGRP [SSN+13], where messages contain between five and seven different metrics) and where route filtering is not allowed. We suppose (notice that this is different from EIGRP) that each router make routing decisions based on any available routing metrics (e.g., path length, bandwidth,

reliability). Surprisingly, also in this setting, we found this mapping technique to be extremely powerful. We will show that, if a protocol handles more than two metrics, then it is possible to map router configurations to logic gates. As a consequence, we derive the computational intractability results as for BGP.

Consider an arbitrary distance-vector routing protocol, where messages associated to each route contain a vector of three metrics A, B, and C. We refer to this protocol as the METRIC-DV. To describe routing metrics, we use the standard terminology from routing algebras [BT10]. Each metric $S = A, B, C$ has a domain D_S and it is endowed with two binary operators $\oplus_S$ and $\otimes_S$. Given two paths, with metric values s_1 and s_2, from the same router, $s_1 \oplus_S s_2$ returns the value of the most preferred one. We assume that $\oplus$ is *transitive*, i.e., if s_1 is preferred over s_2 and s_2 is preferred over a value s_3, then s_1 is preferred over a third value s_3. Given a path p_n, with metric s_n, from a neighbor n of a router r, where the link joining r with n has metric value $s_{(r,n)}$, $s_{(r,n)} \otimes_S s_n$ returns the metric value of the path from r obtained by concatenating (r, n) with p_n. We denote by $\bar{1}_S$ the identity element of $\otimes_S$. For instance, a path length metric has domain $D_S = \mathbb{N}^\infty$ and operators $(\oplus, \otimes) = (\min, +)$. Each router r, for each path of length c learned from one of its neighbors n, adds ($\otimes$ is $+$) the cost of the link from r to n to c. Also, each router selects the shortest ($\oplus$ is min) path among the available ones. The identity element of $+$ is d. A bandwidth metric has domain $D_S = \mathbb{N}^\infty$ and it has operators $(\oplus, \otimes) = (\max, \min)$. In this case, the maximum-bandwidth available path is the most preferred and, since the capacity of a path is equal to the bottleneck capacity of that path (the smallest along the path), a minimum operator is used to compute the path bandwidth. The identity element of min is ∞. A most-reliable metric has domain $D_S = [0, 1]$, operators $(\oplus, \otimes) = (\max, \times)$, and the identity element of $\times$ is 1. When a router r receives a route R_n, with vector $\langle a, b, c\rangle$, from one of its neighbors n, r computes a new vector $\langle e_A \otimes a, e_B \otimes b, e_C \otimes c\rangle$ for route $(r\ n)R_n$, where $\langle e_A, e_B, e_C\rangle$ is the metric of the link between r and n.

In order to discuss the protocol, we define some generic metric values for metrics A, B, and C and a mapping from logic values to routes available at a router. For each metric $S = A, B, C$, let h_S be the most preferred value, m_S be the second-most preferred value, and l_S be the least preferred value. Also, we assume that each metric S is *monotonic*, i.e., for each pair of values x and y of S such that y is preferred over x, $(y \otimes x) \oplus x = x$. It basically means that the concatenation of two paths produces a path whose metric is at least as worse as the path with the lowest metric value. For example, this holds for metrics like path length, bandwidth, and reliability. Indeed, in the path length metric, for two paths of length 10 and 20, respectively, we have that

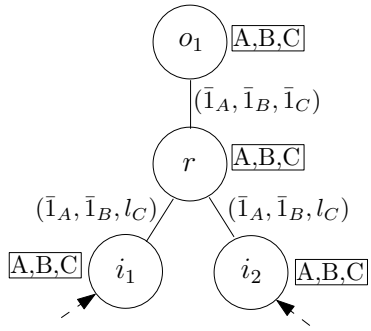

Figure 3.14: The OR gate in METRIC-DV.

$\min(10+20, 20) = 20$. As for the mapping, since no path filtering is allowed, we use a more sophisticated mapping between paths selected at a router and logic values. Namely, we map a router that selects a path with a vector $\langle h_A, l_B, \cdot \rangle$ ($\langle l_A, m_B, \cdot \rangle$) to a 1 (0) in the logic circuit, where $\cdot$ is an arbitrary value for the C value.

We now show that METRIC-DV configurations can simulate OR and NOT gates. To simplify their analysis, we allow identity elements to be used as link metric values (e.g., a link with bandwidth ∞). Even if this is not a realistic assumption, we stress that a more complex analysis would allow us to forbid identity elements as link metric values. In Fig. 3.14 and Fig. 3.15 we use the following graphical convention. Each edge has a label $\langle a, b, c \rangle$ that represents its values for metrics A, B, and C, respectively. Inside a box beside each router r, a label contains metrics in decreasing order of preferences for r. E.g., label $(A, B, C\rangle$ at router r means that r prefers metric A over B, which in turn is preferred over C.

Simulating an OR gate is easy (see Fig. 3.14). It has two input vertices i_1 and i_2, an output vertex o_1, and a vertex r that prefers paths with higher values of A. Observe that, since each edge has identity elements for metrics A and B, metrics A and B remain unchanged when a path is propagated through the gadget. If both i_1 and i_2 select a path with vector $\langle l_A, m_B, \cdot \rangle$, then o_1 also selects a path with vector $\langle l_A, m_B, \cdot \rangle$. Otherwise, if at least one router among i_1 and i_2 selects a path with vector $\langle h_A, l_B, \cdot \rangle$, then o_1 also selects a path with vector $\langle h_A, l_B, \cdot \rangle$. We discuss backward propagation of paths after having introduced the NOT and HUB gadgets.

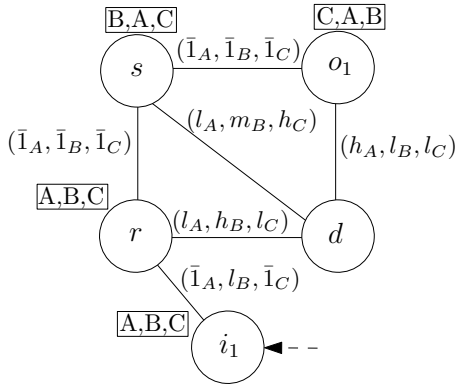

Figure 3.15: The NOT gate in METRIC-DV.

The NOT gadget is a more tricky (Fig. 3.15). Let d be the unique destination vertex of the network. We make several necessary observations. First, each edge of path $(r\ s\ o_1)$ has identity elements as its metric values, hence, if a path is propagated through $(r\ s\ o_1)$, then its metric values remain unchanged. Second, vertices r, s, o_1 never select paths with metrics $\langle l_A, m_B, h_C\rangle$, $\langle h_A, l_B, \cdot\rangle$, and $\langle l_A, \cdot, l_C\rangle$, respectively, because of their metric preferences. In fact, paths with metrics $\langle l_A, h_B, l_C\rangle$, $\langle l_A, m_B, h_C\rangle$, and $\langle h_A, l_B, l_C\rangle$ are steadily available at r, s, and o_1, respectively. For this reason, r (s) selects a path with vector $\langle h_A, l_B, \cdot\rangle$ ($\langle l_A, h_B, l_C\rangle$) only if it receives it from i_1 (s). Now, we are ready to analyze the behavior of the gadget. If i_1 selects a path with vector $\langle h_A, l_B, \cdot\rangle$, which is mapped to a 1 in the logic circuit, then r selects it instead of the direct route to d with vector $\langle l_A, h_B, l_C\rangle$ because it prefers metric A. In turn, s selects its direct route to d with vector $\langle l_A, m_B, h_C\rangle$ instead of the one learned from r because it prefers metric B. Finally, o_1 selects the route via s, which is mapped to a 0 in the logic circuit, instead of the direct one vector $\langle h_A, l_B, l_C\rangle$ because it prefers metric C. Otherwise, if i_1 selects a path with vector $\langle l_A, m_B, \cdot\rangle$, which is mapped to a 0 in the logic circuit, then r selects the direct route to d with vector $\langle l_A, h_B, l_C\rangle$ (in fact, by monotonicity and transitivity, h_B is preferred over l_B, which, in turn, is preferred over $\langle l_B \otimes m_B\rangle$) and propagates it to s, which, in turn, selects this route over the direct one. Finally, o_1 selects its direct route to d with vector $\langle h_A, l_B, l_C\rangle$, which is mapped to a 1 in the logic circuit, instead of the one learned from s with vector $\langle l_A, h_B, l_C\rangle$. In fact, since both routes have the same value for C, the direct route has a

better value for A. To construct an HUB gadget it suffices to connect two NOT gadgets in series, as we have already done in the BGP analysis.

We now discuss the issue of route backward propagations. We first analyze the NOT gadget. As we argued above, r selects a path with vector $\langle h_A, l_B, l_C\rangle$ only if it learns it from i_1. Hence, the only route that can be propagated back to i_1 is its direct path to d with vector $\langle l_A, h_B, l_C\rangle$. Two cases are possible. If i_1 selects a path with vector $\langle h_A, l_B, \cdot\rangle$, which is mapped to 1, then the direct path from r to d is not selected at i_1 and backward propagation is prevented. Otherwise, if i_1 selects a path with vector $\langle l_A, m_B, \cdot\rangle$, which is mapped to 0, then the direct path from r to d is not selected at i_1 because $l_B \otimes h_B$ is, by monotonicity, less preferred than l_B which, in turn, is less preferred than m_B. Hence, backward propagation is prevented in this case too. We now discuss the OR gadget. Since the output of an OR gadget is always connected to the input of an HUB gadget, which is the input of a NOT gadget, we are guaranteed, by the above discussion, that there is no backward propagation from the output router of the OR gadget to any of of its input routers. However, a route with vector $\langle l_A, m_B, h_C\rangle$ can be propagated from i_1 to i_2, which is connected to the output of an HUB gadget. When this route is propagated through path $(i_1\ r\ i_2)$, its metric value at o_1 becomes $\langle l_A, m_B, l_C \otimes (l_C \otimes h_C)\rangle$, which is less preferred than $\langle h_A, l_B, l_c\rangle$ (the metric of the directed path from o_1 to d). Hence, o_1 never selects a path from an input router of the OR gadget, which prevents any backward path propagation.

We draw several interesting considerations derived from this non-trivial construction of the NOT gadget. First, filtering is not a necessary construct for simulating logic circuits. This means that the SAFETY problem may be PSPACE-hard in the bounded model even if the analyzed protocol does not have filter capabilities. Second, per-neighbor and per-egress-point ranking functions, even in the absence of filters, are not necessary constructs for simulating logic circuits. In fact, our construction relies on simple popular metrics. Last, observe that if a protocol has exactly one "strict" monotonic metric, then it is guaranteed to converge to a stable state [GS05]. In this section, we proved that three metrics are enough to make SAFETY PSPACE-hard. We think that studying the complexity of SAFETY in a protocol with two metrics is an interesting non-trivial open problem.

3.7 Conclusions

Over the last 15 years, a routing theory has been developed to study problems on BGP convergence and route propagation. In this chapter, we described a mapping between BGP configurations and logic circuits that puts existing results in a new perspective. We showed how to leverage the mapping to devise reduction techniques and defined the computational complexity of several BGP routing problems under different assumptions. Most notably, by simulating Finite Turing Machines with BGP configurations, we proved the PSPACE-hardness of famous BGP problems like REACHABILITY and SAFETY in a model in which link delays are constrained into finite ranges. We also investigate the impact of restrictions to the expressiveness of the BGP policy language. We show that, under any restriction which does not guarantee convergence, BGP still retains enough expressive power to simulate arbitrary logic circuits, which in turn implies that several interesting BGP problems remain computationally intractable. Finally, we show that the mapping can be effectively used to investigate routing protocols that are very different from BGP. We found an interesting family of routing protocols, with no filter nor per-neighbor ranking constructs, for which the mapping to logic circuits is still possible.

We believe that this study raises many natural questions regarding the possibility of mapping routing configurations to logic circuits. In future work, we plan to investigate whether policy restrictions exist that do not guarantee convergence but allow efficient analysis of BGP configurations. We also plan to extend our analysis to other models and routing protocols.

3.8 Appendix: Building a Turing Machine with Logic Gates

The proof consists of two steps. In the first step we show that it is possible to construct an FTM starting from logic gates and a clock, and interconnecting them with links having a bounded delay. In the second step we use the building blocks in Section 3.2 to translate the logic circuit that simulates the FTM in an eBGP configuration.

Technically, an FTM is a device that processes symbols on a finite-length tape according to a history-less transition function δ. An FTM maintains a state during the computation and uses a head to read and write on specific cells of the tape. At the beginning, the tape is initialized with an initial string of symbols and the FTM is in an initial state. At each step, the FTM reads

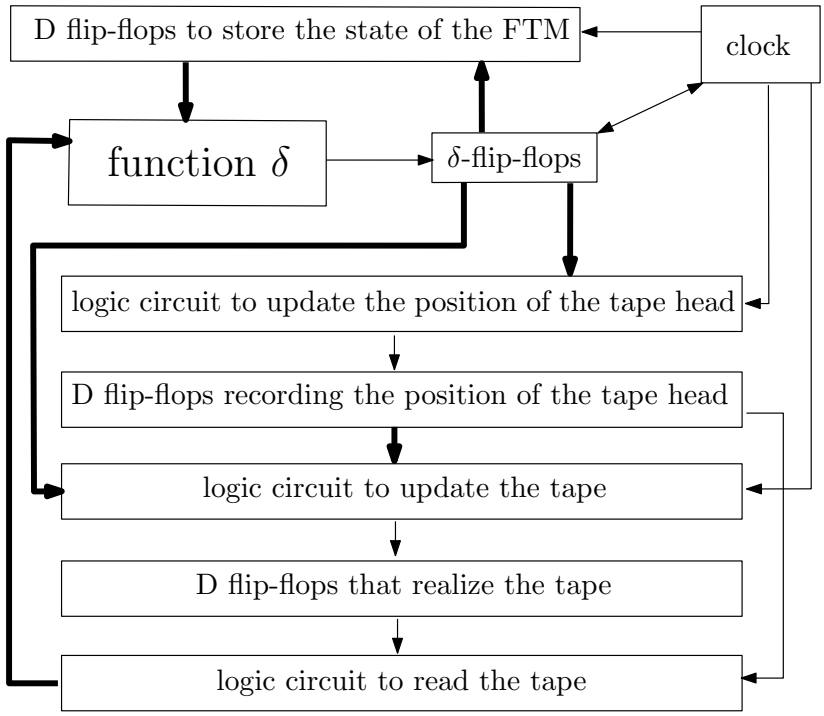

Figure 3.16: A Turing Machine. Delays with values $(\frac{2}{3}T, \frac{2}{3}T + \epsilon)$ and $(\epsilon, 2\epsilon)$ are associated with thick and thin lines, respectively, where T is the maximum time the clock holds the same output value.

the symbol stored in the cell pointed by the head of the tape. Next, according to its state and the read symbol, the transition function δ computes a new symbol to be written on the tape, a new state for the FTM, and eventually a movement of the head of the tape. The computation halts when the FTM reaches some special states.

Now, we build an FTM using only logic gates and a clock (see Fig. 3.16) and connecting the circuital components with links having a minimum and maximum delay. Each building block in the figure represents a logic circuit, hence it can be encoded with an eBGP configuration (see Section 3.2). Details of the circuits represented in Fig. 3.16 are provided in [CCD+12]. Observe

that a clock with the appropriate timing can be built interconnecting, as in Section 3.2, 3 NOT gates. It is not difficult to observe that 3 NOT gates correspond to a BAD-GADGET, which contains a strong routing oscillation.

The main issue is the synchronization. Indeed, on one hand, an FTM performs the computation in a centralized way, where all the operations are synchronized by the FTM itself. On the other hand, in a logic circuit the computation is distributed among the different logic gates and only partially synchronized since the signal propagation delay is variable. In a perfectly synchronized model where signal propagation delay is deterministic, it is easy to keep the circuit synchronized by conveniently assigning specific delays to the links. However, in the more general min-max model, the synchronization requires more attention since we need to be sure that the variability of link delays does not add up across multiple iterations. We build the logic circuit that simulates an FTM as follows. Function δ is simulated by a combinational logic circuit obtained by conveniently interconnecting simple logic gates. We call such a combinational circuit "δ block". A set of D flip-flops, called the δ-flip-flops, are interconnected to the output of the δ block in order to store its output value. The tape, the position of the head of the tape, and the state of the FTM, are simulated using D flip-flops. We build a logic circuit that updates the position of the head and we connect it to the clock (which acts as an enabler), to the δ-flip-flops that store the new position of the head (which is the input of the circuit), and to the D flip-flops that store the current position of the head (which is the output of the circuit). Similarly, we build a circuit that reads the tape and another one that updates it. The link delays within the clock are set in such a way that the clock's output node produces the same output value (either a 0 or a 1) for a time that is at most T and at least $T - \epsilon$, with $\epsilon << T$.

We now show that even if the clock period is variable, we can synchronize the circuit to simulate a FTM. Assume, for the sake of simplicity, that the clock switches to 1 at time 0 and that at time 0 the δ-flip-flops store the new state, the symbol that needs to be written on the tape, and the direction in which to move the head.

Consider a single clock period, which consists of two clock ticks (one for 1 and one for 0). The main intuition is to use the clock value 0 as a disabler, in order to block the propagation of signals until the clock switches to 1 again. During the first tick, the clock outputs 1 so all the δ flip-flops are enabled and signals can propagate. Within this tick, we need to

- propagate the new symbol to write on the tape to the logic circuit that

updates the tape;

- propagate the new tape head position to the logic circuit that updates the position;
- propagate the new state to the D flip-flops that store the state; and
- transfer the symbol which is currently on the tape to the input of the δ-block.

Observe that we need to make sure that the new symbol is written on the correct cell of the tape. In order to do so, we introduce a propagation delay between the D flip-flops that store the head position and the logic circuit that updates the tape. This ensures that when the circuit updates the tape, it will refer to the correct position, independent of whether the new position has been already written to the D flip-flops that store the head position.

On the other hand, we need to avoid the possibility that two or more symbols are read and elaborated within a single clock period. In particular, we want the δ-block to be disabled when the newly read symbol arrives, because we want this new symbol to be processed in the next clock period. This means that the propagation delay from the logic circuit that reads the tape to the δ-block must ensure that the symbol arrives when the clock has already switched to 0.

Now, if we can find an assignment of link delays that guarantees the above properties, then we can apply the same arguments to the next clock period and so on, yielding the ability to synchronize the signals to the clock periods and hence completing the definition of FTM. We show an example of such an assignment of link delays in Appendix 3.8.

The computation of an FTM finishes when a *final state* is reached. If and only if the FTM reaches a final state, the clock is stopped by the δ-flip-flops, hence the logic circuit stabilizes. Some final states are called *acceptance states* as in standard Turing Machine terminology.

Finally, we note that, since an FTM has a tape length which is polynomial in the size of the input string, the entire construction of the logic circuit takes polynomial time. More details about the initialization phase of the logic circuit can be found in [CCD$^+$12].

Analysis of the Simulation of a Finite Turing Machine

In Section 3.3, we claimed that, in the min-max model, delay can be assigned to links in such a way that each step of a Finite Turing Machine (FTM) can be

simulated by the corresponding BGP configuration in a separate clock period. We now prove our claim by showing a convenient link delay assignment.

Delays between the logic circuit blocks are shown in Fig. 3.16. In the figure, thick and thin lines represent delays of $(\frac{2}{3}T, \frac{2}{3}T + \epsilon)$ and $(\epsilon, 2\epsilon)$ respectively, where $2T$ is the period of the clock and $\epsilon << T$. All the internal delays of the various logic circuit blocks are set to $(\epsilon, 2\epsilon)$. Observe that, since $\epsilon << T$, the ordering of events occurring during one clock period, is not influenced by ϵ link delays or their sum. Hence, we disregard ϵ delays in the following.

We now show that the ordering of events occurring during one clock period is as defined in Section 3.3, allowing us to simulate any FTM in the min-max model. The computation starts at time $t = 0$, with the output of the clock at 0 until $t = T$. Before $t = T$, the only event that occurs is the transfer of the symbol in the current cell of the tape to function δ. At time $t = T$ the clock signal changes from 0 to 1, enabling the δ-flip-flops. As a consequence, the δ-flip-flops change their output values according to the output of the δ function. After $\frac{2}{3}T$, the new output of the δ-flip-flops is provided as input to the logic circuit that controls the tape head, to the logic circuit to update the tape, and to the flip-flops that store the state of the FTM. Hence, at time $t = T + \frac{2}{3}T = \frac{5}{3}T$, the state and the position of the head are updated. Moreover, at time $T + 2\frac{2}{3}T = \frac{7}{3}T$, the tape is updated, and the new symbol on the tape and the new state of the FTM are provided as input to the δ function. Thus, the new output of function δ is ready at time $\frac{7}{3}T$. However, since $\frac{7}{3}T > 2T$ and $\frac{7}{3}T < 3T$, the clock is at 0 and all flip-flops are disabled at that time, and nothing changes anymore until $t = 3T$, when the simulation of the next step of the FTM is performed.

Observe that if $\epsilon = 0$ the arguments above are still valid. This means that the above simulation works also when the minimum and maximum delay on each link are set to the same value, i.e., in the completely synchronous model.

Chapter 4

Traffic Hijacking in BGP and S-BGP *

In Chapters 2 and 3, we proved that, in the general case, the outcome of the routing process may be completely unpredictable. In this chapter, we therefore assume that routing is stable. We study how local routing changes (e.g., modifying export filters or announcing bogus routes) affects the global Internet routing. This problem is of great interest for the networking community.

Harmful Internet hijacking incidents put in evidence how fragile BGP is. As proved by recent research contributions, even S-BGP, the secure variant of BGP that is being deployed, is not fully able to blunt traffic attraction attacks. Given a traffic flow between two ASes, we study how difficult it is for a malicious AS to devise a strategy for hijacking or intercepting that flow. We show that this problem marks a sharp difference between BGP and S-BGP. Namely, while it is solvable, under reasonable assumptions, in polynomial time for the type of attacks that are usually performed in BGP, it is NP-hard for S-BGP. Our study has several by-products. E.g., we solve a problem left open in the literature, stating when performing a hijacking in S-BGP is equivalent to performing an interception.

*Part of the material presented in this chapter is based on the following publications: M. Chiesa, G. Di Battista, T. Erlebach, M. Patrignani. Computational Complexity of Traffic Hijacking under BGP and S-BGP. In *Proc. ICALP*, 2012.

4.1 Introduction and Overview

On 24th Feb. 2008, Pakistan Telecom started an unauthorized announcement of prefix 208.65.153.0/24 [Und08]. This announcement was propagated to the rest of the Internet, which resulted in the *hijacking* of YouTube traffic on a global scale. Incidents like this put in evidence how fragile BGP is, which is used to exchange routing information between Internet Service Providers (ISPs). Indeed, performing a hijacking attack is a relatively simple task. It suffices to issue a BGP announcement of a victim prefix from a border router of a malicious (or unaware) *Autonomous System (AS)*. Part of the traffic addressed to the prefix will be routed towards the malicious AS rather than to the intended destination. A mischievous variation of the hijacking is the *interception* when, after passing through the malicious AS, the traffic is forwarded to the correct destination. This allows the rogue AS to eavesdrop or even modify the transit packets.

In order to cope with this security vulnerability, a variant of BGP, called S-BGP [KLS00], has been proposed, that requires a Public-Key Infrastructure (PKI) both to validate the correctness of the AS that originates a prefix and to allow an AS to sign its announcements to other ASes. In this setting an AS cannot forge announcements that do not derive from announcements received from its neighbors. However, [GSHR10] contains surprising results: (i) simple hijacking strategies are tremendously effective and (ii) finding a strategy that maximizes the amount of traffic that is hijacked is NP-hard for both BGP and for S-BGP.

In this chapter we tackle the hijacking and interception problems from a new perspective. Namely, given a traffic flow between two ASes, how difficult is it for a malicious AS to devise a strategy for hijacking or intercepting at least that specific flow? We show that this problem marks a sharp difference between BGP and S-BGP. Namely, while it is polynomial time solvable, under reasonable assumptions, for typical BGP attacks, it is NP-hard for S-BGP. This gives new complexity related evidence of the effectiveness of the adoption of S-BGP. Also, we solve an open problem [GSHR10], showing when every hijack in S-BGP results in an interception. Tab. 4.1 summarizes our results. Rows correspond to different settings for a malicious AS m. The origin-spoofing setting (Sect. 4.2) corresponds to a scenario where m issues BGP announcements pretending to be the owner of a prefix. Its degree of freedom is to choose a subset of its neighbors for such a bogus announcement. This is the most common type of hijacking attack to BGP [wik12]. In S-BGP (Sect. 4.3) m must enforce the constraints imposed by S-BGP, which does not allow m to

Table 4.1: Complexity of finding a HIJACK strategy in different settings.

	AS-paths of any length	Bounded AS-path length	Bounded AS-path length and AS degree
Origin-spoofing	NP-hard (Thm. 4.1)	P(Thm. 4.5)	P
S-BGP	NP-hard	NP-hard (Thm. 4.1)	P(Thm. 4.2)

pretend to be the owner of a prefix that is assigned to another AS. Columns of Tab. 4.1 correspond to different assumptions about the Internet. In the first column we assume that the longest *valley-free* path (i.e. a path enforcing certain customer-provider constraints) in the Internet can be of arbitrary length. This column has a theoretical interest since the length of the longest path (and hence valley-free path) observed in the Internet remained constant even though the Internet has been growing in terms of active AS numbers during the last 15 years [Hus12]. Moreover, in today's Internet about 95% of the ASes is reached in 3 AS hops [Hus12]. Hence, the second column corresponds to a quite realistic Internet, where the AS-path length is bounded by a constant. In the third column we assume that the number of neighbors of m is bounded by a constant. This is typical in the periphery of the Internet. A "P" means that a Polynomial-time algorithm exists. Since moving from left to right the setting is more constrained, we prove only the rightmost NP-hardness results, since they imply the NP-hardness results to their left. Analogously, we prove only the leftmost "P" results.

Past work either attempt to optimize different objective measures (e.g., maximize the amount of attracted traffic, routing traffic throughout the most preferred path) or do not tackle the problem from an algorithmic perspective [BFZ07, LSZ08, GHJ+08, GSHR10, BG11, BG14].

Gao-Rexford Model

We use the 3-SPP model introduced in Chapter 1. Since checking for SAFETY in the 3-SPP model is NP-hard, we focus on a particular restriction of 3-SPP where policies must satisfy the so-called Gao-Rexford conditions. These conditions model those routing policies that are typically enforced by real-world economic relationships among ASes.

We recall that BGP allows each AS to autonomously specify which paths are forbidden (*import policy*), how to choose the best path among those available

to reach a destination (*selection policy*), and a subset of neighbors to whom the best path should be announced (*export policy*). Since BGP treats each prefix independently, we focus on a single prefix π, owned by a destination vertex d.

Policies are typically specified according to two types of relationships [Hus99]. In a *customer-provider* relationship, an AS that wants to access the Internet pays an AS which sells this service. In a *peer-peer* relationship two ASes exchange traffic without any money transfer between them. Such commercial relationships between ASes are represented by orienting a subset of the edges of E. Namely, edge $(u, v) \in E$ is directed from u to v if u is a customer of v, while it is undirected if u and v are peers. A path is *valley-free* if provider-customer and peer-peer edges are only followed by provider-customer edges.

The Gao-Rexford [GR00] Export-all (*GR-EA*) conditions are commonly assumed to hold in this setting [GSHR10].

- **GR1**: G has no directed cycles that would correspond to unclear customer-provider roles.
- **GR2**: Each vertex $v \in V$ sends an announcement containing a path P to a neighbor n only if path $(n\ v)P$ is valley-free. Otherwise, some AS would provide transit to either its peers or its providers without revenues.
- **GR3**: A vertex prefers paths through customers over those provided by peers and paths through peers over those provided by providers.
- **Shortest Paths**: Among paths received from neighbors of the same class (customers, peers, and provider), a vertex chooses the shortest ones.
- **Tie Break**: If there are multiple such paths, a vertex chooses according to some tie break rule. As in [GSHR10], we assume that the one whose next hop has lowest AS number is chosen. Also, as in [ES11], to break ties between equal class and equal length simple paths $P_1^u = (u\ v)P_1^v$ and $P_2^u = (u\ v)P_2^v$ at the same vertex u from the same neighbor v, if v prefers P_1^v over P_2^v, then u prefers P_1^u over P_2^u. This choice is called *policy consistent* in [ES11].
- **NE policy**: a vertex always exports a path except when GR2 forbids it to do so.

Since we assume that the GR-EA conditions are satisfied, then a (partially directed) graph is sufficient to fully specify the policies of the ASes. Hence, in the following a *BGP instance* is just a graph.

Understanding Hacking Strategies

We consider the following problem. A BGP instance with three specific vertices, d, s, and m are given, where such vertices are: the AS originating a prefix π, a source of traffic for π, and an attacker, respectively. All vertices, but m, behave correctly, i.e., according to the BGP protocol and GR-EA conditions. Vertex m is interested in two types of attacks: *hijacking* and *interception*. In the hijacking attack m's goal is to attract to itself at least the traffic from s to d. In the interception attack m's goal is to be traversed by at least the traffic from s to d.

It is worth to observe that computing an hijacking strategy in a shortest-path routing protocol is an easy task. Since a network prefers shorter routes, an attacker can attract traffic from all the networks that are "closer" to him than to the correct destination by simply announcing that it is the rightful destination. In BGP, we will show that the problem is more tricky.

In Fig. 4.1 $(2, 6)$ is peer-to-peer and the other edges are customer-provider. Prefix π is owned and announced by d. According to BGP, the traffic from s to d follows (s 6 2 1 d). In fact, 2 selects (1 d). Vertex 6 receives a unique announcement from d (it cannot receive an announcement with (5 4 3 m 2 1 d) since it is not valley-free). By cheating, (**Example 1**) m can deviate the traffic from s to d attracting traffic from s. In fact, if m pretends to be the owner of π and announces it to 2, then 2 prefers, for the shortest path criterion, (2 m) over (2 1 d). Hence, the traffic from s to d is received by m following (s 6 2 m). A hijack!

Observe that m could be smarter (**Example 2**). Violating GR2, it can announce (2 1 d) to 3. Since each of 3, 4 and 5 prefers paths announced by customers (GR3), the propagation of this path is guaranteed. Therefore, 6 has two available paths, namely, (2 1 d) and (5 4 3 m 2 1 d). The second one is preferred because 5 is a customer of 6, while 2 is a peer of 6. Hence, the traffic from s to d is received by m following path (s 6 5 4 3 m). Since after passing through m the traffic reaches d following (m 2 1 d) this is an interception.

Fig. 4.2 allows us to show a negative example (**Example 3**). According to BGP, the traffic from s to d follows (s 4 d). In fact, s receives only paths (4 d) and (1 2 3 d), both from a provider, and prefers the shortest one. Suppose that m wants to hijack and starts just announcing π to 6. Since all the neighbors of s are providers, s prefers, for shortest path, (4 d) over (5 6 m) (over (1 2 3 d) over (4 9 8 7 m)) and the hijack fails. But m can use another strategy. Since (s 5 6 m) is shorter than (s 1 2 3 d), m can attract traffic if (4 d) is "disrupted" and becomes not available at s. This happens if 4 selects, instead of (d), a path

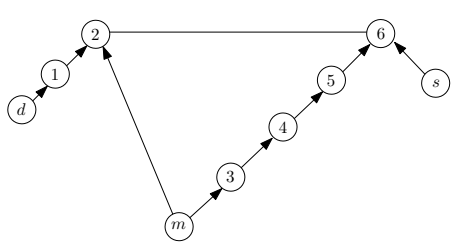

Figure 4.1: A network for Examples 1 and 2.

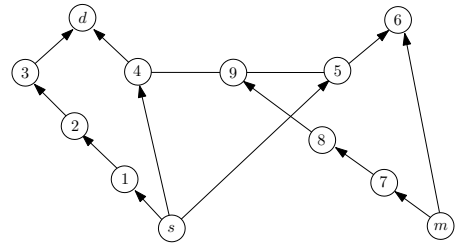

Figure 4.2: A network for Example 3.

received from its peer neighbor 9 (m may announce that it is the originator of π also to 7). However, observe that if 4 selects path (4 9 8 7 m) then 5 selects path (5 9 8 7 m) since it is received from a peer and stops the propagation of (s 5 6 m). Hence, s still selects path (s 1 2 3 d) and the hijack fails.

In order to cope with the lack of any security mechanism in BGP, several variations of the protocol have been proposed by the Internet community. One of the most famous, S-BGP, uses both origin authentication and cryptographically-signed announcements in order to guarantee that an AS announces a path only if it has received this path in the past.

The attacker m has more or less constrained *cheating capabilities*. (i) With the *origin-spoofing* cheating capabilities m can do the typical BGP announcement manipulation. I.e., m can pretend to be the origin of prefix π owned by d, announcing this to a subset of its neighbors. (ii) With the *S-BGP* cheating capabilities m must comply with the S-BGP constraints. I.e.: (a) m cannot pretend to be the origin of prefix π; and (b) m can announce a path $(m\ u)P$ only if u announced P to m in the past. However, m can still announce paths that are not the best to reach d and can decide to announce different paths to different neighbors. In Example 2, m has S-BGP cheating capabilities.

In this chapter, we study the computational complexity of the HIJACK and of the INTERCEPTION problems. The HIJACK problem is formally defined as follows.

Instance: A BGP instance G, a source vertex s, a destination vertex d, a manipulator vertex m, and a cheating capability for m.

Question: Does there exist a set of announcements that m can simultaneously send to its neighbors, according to its cheating capability, that produces a stable state for G where the traffic from s to d goes to m?

The INTERCEPTION problem is defined in the same way but changing "the traffic from s to d goes to m" to "the traffic from s to d passes through m before reaching d".

Path Ranking

We introduce some technical notation in order to prove our lemmas and theorems. Let P be a valley-free path from vertex v. We say that P *is of class* 3, 2, or 1 if its first edge connects v with a customer, a peer, or a provider of v, respectively. We also define a function f^v for each vertex v, that maps each path from v to the integer of its class. Given two paths P and P' available at v if $f^v(P) > f^v(P')$ we say that the class of P *is better than* the class of P'.

Path Disruption

Consider a stable routing state and two vertices v_1 and v_n connected in G by path $P = (v_1 \ \ldots \ v_n)$. Suppose that v_n owns prefix π. Path P is *disrupted at vertex* v_i by a path P' if there exists a vertex v_i of P such that v_i selects path P' different from $(v_i \ \ldots \ v_n)$ (see Example 2).

Vertex v_i may select P' over $(v_i \ \ldots \ v_n)$ for several reasons. As an example, $(v_i \ \ldots \ v_n)$ could not be available at v_i. If P' is preferred because of the GR3 condition, we say that P *is disrupted by a path of a better class.* If P' is preferred because of the shortest-paths criterion, we say that P *is disrupted by a path of the same class.*

Routing Stability under Manipulator Attacks

BGP policies can be so complex that there exist configurations that do not allow the network to reach any stable routing state (see, e.g., [GSW02]). A routing state is *stable* if there exists a time t such that after t no AS changes its selected path. If the GR-EA conditions are satisfied, then a BGP network always converges to a stable state [GR00]. However, there is a subtle issue to consider in attacks. As we have seen in the previous examples, m can deliberately ignore the GR-EA conditions. Anyway, the following lemma makes it possible, in our setting, to study the HIJACK and the INTERCEPTION problem ignoring stability related issues. The same result has been independently obtained in [LGS12].

Lemma 4.1 *Let G be a BGP instance and suppose that at a certain time one or more manipulators start announcing steadily any set of arbitrary paths to its or their neighbors. Routing in G converges to a stable state.*

Proof: Suppose, for a contradiction, that, after all manipulators start their announcements, routing in G is not stable. In [GSW99], it has been proved that (with no manipulators) the presence of a specific circular structure, known in the literature as a *dispute-wheel*, is a necessary condition for routing instability. We first improve this result, proving that, if routing is not stable, such a structure must necessarily exist also in the presence of one or more manipulators that steadily announce possibly different bogus paths. Then, as a consequence of this, we show that the existence of a dispute-wheel in a BGP instance implies a violation of the GR-EA conditions, which leads to a contradiction. Hence, a GR-EA instance always converges to a stable state even in the presence of one or more manipulators.

Let $U = u_0, \ldots, u_n$ be a circular sequence of vertices of G such that for each u_i: (1) u_i does not steadily announce a path; and (2) the most preferred path $P^{u_i} = R_i Q_i$ that is selected infinitely many times at u_i is such that R_i ends at u_{i+1}, Q_i starts with u_{i+1}, and each vertex in Q_i but u_{i+1} steadily announces a path, where i has to be interpreted modulo $n+1$. This circular sequence is a specific instance of a dispute-wheel. We show that if such a circular sequence U does not exist, then at least one vertex of U is stable, which is a contradiction. Suppose by contradiction that such a circular sequence $U = u_0, \ldots, u_n$ does not exist. Let u_0 be a vertex that does not steadily select a path and U be a maximal (non-circular) sequence of vertices such that $u_0, \ldots, u_{l-1}$ satisfies (1) and (2) and u_l only satifies (1). Since u_l does not satisfy (2), it means that all vertices in P^{u_l} but u_l, are stable. This means that u_l should steadily select P^{u_l} as its best path, which leads to a contradiction since we assumed that u_l does not steadily select a path. Hence, if routing is not stable, there exists a circular sequence U as described above.

We now prove that the presence of a dispute-wheel in a BGP instance implies a violation of the GR-EA conditions. We first make some basic observations about U. For each $i = 0, \ldots, n$: (i) path R_i contains at least two vertices (i.e., u_i and u_{i+1}); (ii) vertex u_i is not a manipulator, since it does not steadily select a path; (iii) $P^{u_i} \neq Q_{i-1}$, otherwise, since u_i is the only vertex in Q_{i-1} that is not stable, it would imply that u_{i+1} is stable, which is a contradiction. (iv) $\lambda^{u_i}(P^{u_i}) < \lambda^{u_i}(Q_{i-1})$ (i.e., P^{u_i} is preferred over Q_{i-1}).

Now, consider observation (iv). Two cases are possible. Either each path P^{u_i} is preferred over Q_{i-1} for same class or there exists a vertex u_j that prefers P^{u_j} over Q_{j-1} for better class. In the first case, it means that, for each u_i, path P^{u_i} is preferred over Q_{i-1} either by shortest path or by tie-break criterion. In both cases, it implies that $f^{u_i}(Q_{i-1}) = f^{u_i}(P^{u_i})$ and $|Q_{i-1}| \geq |P^{u_i}|$. Since we know from observation (ii) that $|R_i| \geq 2$, we obtain $|P^{u_i}| > |Q_i|$, which implies

$|Q_{i-1}| \geq |P^{u_i}| > |Q_i| \geq |P^{u_{i+1}}|$. Following the cycle of inequalities we have a contradiction as we obtain $|Q_i| > |Q_i|$. In the second case, let u_j be a vertex that prefers P^{u_j} over Q_{j-1} for better class, that is $f^{u_j}(P^{u_j}) > f^{u_j}(Q_{j-1})$. For each $i = 0, \ldots, j-1, j+1, \ldots, n$ we have $f^{u_i}(P^{u_i}) \geq f^{u_i}(Q_{i-1}) \geq f^{u_{i-1}}(P^{u_{i-1}})$ because of the GR2 and GR3 conditions. Following the cycle of inequalities we have a contradiction as we obtain $f^{u_j}(P^{u_j}) > f^{u_j}(P^{u_j})$.

□

The existence of a stable state (pure Nash equilibrium) in a game where one player can deviate from a standard behavior has been proved, in a different setting in [ES11]. Such a result and Lemma 4.1 are somehow complementary since the export policies they consider are more general than Export-All, while the convergence to the stable state is not guaranteed (even if such a stable state is always reachable from any initial state).

4.2 Checking if an Origin-Spoofing BGP Attack Exists

In this section, we show that, in general, it is hard to find an attack strategy if m has an origin-spoofing cheating capability (Theorem 4.1), while the problem turns out to be easier in a realistic setting (Theorem 4.5).

A hijacking can be obviously found in exponential time by a simple brute force approach which simulates every possible attack strategy and verifies its effectiveness. The following result in the case the Internet graph has no degree constraints may be somehow expected.

Theorem 4.1 *If the manipulator has origin-spoofing cheating capabilities, then problem* HIJACK *is NP-hard.*

Proof: We prove that HIJACK is NP-hard by a reduction from the 3-SAT problem, which is known to be NP-complete [Pap94]. Let F be a logical formula in conjunctive normal form with variables $X_1 \ldots X_n$ and clauses $C_1 \ldots C_h$ where each clause C_i contains three literals. We construct a GR-EA compliant BGP instance G as follows.

Graph G consists of 4 structures: the INTERMEDIATE structure, the SHORT structure, the LONG structure, and the DISRUPTIVE structure. See Fig. 4.3.

The INTERMEDIATE structure is the only portion of G containing valley-free paths joining s and m that are shorter than the one contained in the LONG structure. It is composed by edge (m, q_1) and two directed paths from s to q_1 of length $2n$: the first path is composed by edges (s, t_n), (t_n, q_n), (q_n, t_{n-1}), (t_{n-1}, q_{n-1}), (q_{n-1}, t_{n-2}), $\ldots$, (t_2, q_2), (q_2, t_1), and (t_1, q_1) while the second

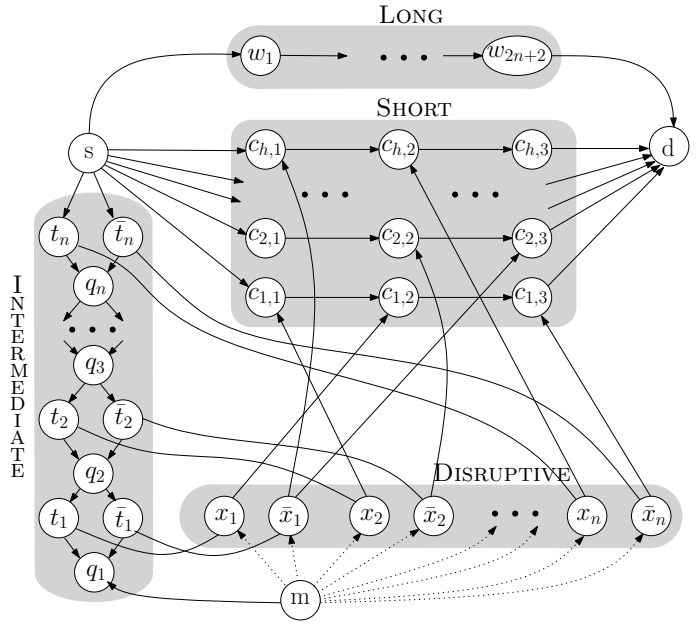

Figure 4.3: Reduction of the 3-SAT problem to the HIJACK problem when m has origin-spoofing capabilities. Dotted lines from m to vertices x_i and $\bar{x}_i$ have length $2n + 2$.

path is composed by edges $(s, \bar{t}_n)$, $(\bar{t}_n, q_n)$, $(q_n, \bar{t}_{n-1})$, $(\bar{t}_{n-1}, q_{n-1})$, $(q_{n-1}, \bar{t}_{n-2})$, $\ldots$, $(\bar{t}_2, q_2)$, $(q_2, \bar{t}_1)$, and $(\bar{t}_1, q_1)$. Obviously, these two paths can be used to construct an exponential number of other paths. We say that a path traverses the INTERMEDIATE structure if it passes through vertices s and q_1.

The SHORT structure consists of h paths joining s and d. Each path has length 4 and has edges $(s, c_{i,1})$, $(c_{i,1}, c_{i,2})$, $(c_{i,2}, c_{i,3})$, and $(c_{i,3}, d)$ $(1 \leq i \leq h)$. The LONG structure is a directed path of length $2n + 3$ with edges (s, w_1), (w_1, w_2), $\ldots$, (w_{2n+1}, w_{2n+2}), and (w_{2n+2}, d). The DISRUPTIVE structure is composed by $2n$ paths plus $3h$ edges. The $2n$ paths are defined as follows. For $1 \leq i \leq n$ we define two paths. The first path contains a directed subpath of length $2n + 2$ from m to x_i (dotted lines in Fig. 4.3), plus the undirected edge (x_i, t_i). The second path contains a directed subpath of length $2n+2$ from m to $\bar{x}_i$ (dotted lines in Fig. 4.3) plus the undirected edge $(\bar{x}_i, \bar{t}_i)$. The $3h$ edges are

added to G as follows. For each clause C_i and each literal $L_{i,j}$ of C_i, which is associated to a variable x_k, if $L_{i,j}$ is positive, then we add $(x_k, c_{i,j})$, otherwise we add $(\bar{x}_k, c_{i,j})$. We say that a path traverses the DISRUPTIVE structure if it traverses it from m to s.

Vertices s, d, and m have source, destination, and manipulator roles, respectively.

Intuitively, the proof works as follows. The paths that allow traffic to go from s to m are only those passing through the DISRUPTIVE structure and the INTERMEDIATE structure. Also, the paths through the INTERMEDIATE structure are shorter than the one through the LONG structure, which is shorter than those through the DISRUPTIVE structure.

If m does not behave maliciously, s receives only paths that traverse the SHORT structure and the LONG structure. In this case s selects one of the paths in the SHORT structure according to its tie break policy.

Observe that if m wants to attract traffic from s, then: (i) a path from m traversing entirely the INTERMEDIATE structure has to reach s and (ii) all paths contained in the SHORT structure have to be disrupted by a path announced by m.

Observe that only valley-free paths contained in the INTERMEDIATE structure, which have length at least $2n+2$, can be used to attract traffic from s. If (i) does not hold, then s selects the path contained in the LONG structure or a path contained in the SHORT structure. If (ii) does not hold, then s selects a path contained in the SHORT structure.

Our construction is such that the 3-SAT formula is satisfiable iff m can attract the traffic from s to d. To understand the interplay between our construction and the 3-SAT problem, consider (see Fig. 4.3) the behavior of m with respect to neighbors x_2 and $\bar{x}_2$. If m wants to disrupt path (s $c_{1,1}$ $c_{1,2}$ $c_{1,3}$ d) (which corresponds to making clause C_1 true) it might announce the prefix to x_2. This would have the effect of disrupting (s $c_{1,1}$ $c_{1,2}$ $c_{1,3}$ d) by better class. Observe that at the same time this would disrupt all the paths through t_2. If m is able to disrupt all the paths in the SHORT structure, then s has to select a path in the INTERMEDIATE structure. However, m has to be careful for two reasons. First, m has to announce the prefix to q_1 (otherwise no path can traverse the INTERMEDIATE structure). Second, m cannot announce the prefix both to x_2 and to $\bar{x}_2$ (variable X_2 cannot be true and false at the same time). In this case, all the paths through t_2 and $\bar{t}_2$ are disrupted. Also, consider that the paths that reach s through t_2 and x_2 ($\bar{t}_2$ and $\bar{x}_2$) and that remain available are longer than the one in the LONG structure.

Now we show that if F is satisfiable, then m can attract traffic from s. Let

M be a truth assignment to variables $X_1, \ldots, X_n$ satisfying formula F. Let m announce to its neighbors paths as follows: if X_i $(i = 1, \ldots, n)$ is true then m announces the prefix to x_i and does not announce anything to $\bar{x}_i$; otherwise m does the opposite. Also, the prefix is announced to q_1 in all cases.

We have that: (i) all paths (one for each clause) in the SHORT structure are disrupted by better class from the paths in the DISRUPTIVE structure; (ii) one path belonging to the INTERMEDIATE structure is available at s; (iii) the path in the LONG structure, available at s, is longer than the path in the INTERMEDIATE structure. Hence, m can attract traffic from s.

Now we prove that if manipulator m can attract traffic from s, then F is satisfiable.

We already know from the above discussion that m can attract traffic from s only using paths that traverse the INTERMEDIATE structure entirely. We also know that these paths are longer than paths contained in the SHORT structure and therefore, every path contained in the SHORT structure has to be disrupted. We have that paths contained in the SHORT structure can be disrupted only by using paths contained in the DISRUPTIVE structure. Let V^* be the set of neighbors of m different from q_1 that receive an announcement of the prefix from m. Observe that s, to attract traffic from m, has to announce the prefix to q_1. From the above discussion we have that for $i = 1, \ldots, n$ it is not possible both for x_i and for $\bar{x}_i$ to receive the announcement. Also, since all paths in the SHORT structure have been disrupted, for $j = 1, \ldots, h$ at least one of the $c_{j,k}$ $(k = 1, 2, 3)$ receives an announcement of the prefix from m. Hence, we define an assignment M, which satisfies formula F, as follows: for each $i = 1, \ldots, n$, if $x_i \in V^*$, then $M(X_i) = \top$, otherwise $M(X_i) = \bot$. □

Surprisingly, in a more realistic scenario, where the length of valley-free paths is bounded by a constant k, we have that in the origin-spoofing setting an attack strategy can be found in polynomial time ($n^{O(k)}$, where n is the number of vertices of G). Let N be the set of neighbors of m. Indeed, the difficulty of the HIJACK problem in the origin-spoofing setting depends on the fact that m has to decide to which of the vertices in N it announces the attacked prefix π, which leads to an exponential number of possibilities. However, when the longest valley-free path in the graph is bounded by a constant k, it is possible to design a polynomial-time algorithm based on the following intuition, that will be formalized below. Suppose m is announcing π to a subset $A \subseteq N$ of its neighbors and path $p = (z \ \ldots \ n \ m)$ is available at an arbitrary vertex z of the graph. Let n_1, n_2 be two vertices of $N \setminus A$. If p is disrupted (is not disrupted) by better class both when π is announced either to n_1 or to n_2, then

p is disrupted (is not disrupted) by better class when π is announced to both n_1 and n_2. This implies that once m has a candidate path p^* for attracting traffic from s, it can check independently to which of its neighbors it can announce π without disrupting p^* by better class, which guarantees that a path from m to z longer than p cannot be selected at z.

In order to prove Theorem 4.5, we introduce the following lemmata that relate attacks to the structure of the Internet.

Lemma 4.2 *Consider a valley-free path $p = (v_n \ \dots \ v_1)$ and consider an attack of m such that v_1 announces a path p_{v_1} to v_2 to reach prefix π and p is not disrupted by better class. Vertex v_n selects a path $\lambda^{v_n}(p_n) \leq \lambda^{v_n}(pp_{v_1})$.*

Proof: We prove inductively that each vertex v_i in p selects a path $\lambda^{v_i}(p_i) \leq \lambda^{v_i}((v_i \ \dots \ v_1)p_{v_1})$ such that $|p_i| \leq |(v_i \ \dots \ v_1)p_{v_1}|$. In the base case ($n = 1$), the statement holds since v_1 selects p_{v_1}. In the inductive step ($n > 1$), by induction hypothesis and NE policy, vertex v_i receives a path p_{i-1} from vertex v_{i-1} such that $\lambda^{v_{i-1}}(p_{i-1}) \leq \lambda^{v_{i-1}}((v_{i-1} \ \dots \ v_1)p_{v_1})$ and $|p_{i-1}| \leq |(v_{i-1} \ \dots \ v_1)p_{v_1}|$. Two cases are possible: p_{i-1} contains v_i, or not. In the second case, v_i selects a path $\lambda^{v_i}(p_i) \leq \lambda^{v_i}((v_i \ v_{i-1})p_{i-1})$ and since path $(v_i \ \dots \ v_1)$ is not disrupted by better class, we have also $|p_i| \leq |(v_i \ v_{i-1})p_{i-1}| \leq |(v_i \ \dots \ v_1)|$. In the first case, let p' be the sub-path of p_{i-1} from v_i. Observe that, since m has only origin-spoofing cheating capabilities, it cannot invalidate the acyclicity of G (the same does not hold if m has S-BGP cheating capabilities). As a consequence, since $(v_i \ v_{i-1})p_{i-1}$ is a valley-free path, vertex v_i is repeated in that path, and G is acyclic, we have that the class of p' must be higher than the class of $(v_i \ v_{i-1})p_{i-1}$. Hence, by inductive hypothesis, we have $f^{v_i}(p') > f^{v_i}((v_i \ v_{i-1})p_{i-1}) \geq f^{v_i}(v_i \ \dots \ v_1)$, which is a contradiction since $(v_i \ \dots \ v_1)$ cannot be disrupted by better class. Hence, the statement is proved. □

Lemma 4.3 *Consider a successful attack for m and let p_{sm} be the path selected at s. Let p_{sd} be a valley-free path from s to d that does not traverse m and that satisfies $\lambda^s(p_{sd}) < \lambda^s(p_{sm})$. Path p_{sd} is disrupted by a path of better class.*

Proof: Suppose by contradiction that there exists a valley-free path p_{sd} from s to d such that $\lambda^s(p_{sd}) < \lambda^s(p_{sm})$ and p_{sd} is not disrupted by a path of better class. If p_{sd} is not disrupted, then it is available at vertex s. It implies that s selects p_{sd} as its best path, which leads to a contradiction. Otherwise, suppose p_{sd} is disrupted only by same class. By Lemma 4.2 we

have a contradiction since s selects a path $\lambda^s(p) \leq \lambda^s(p_{sd}) < \lambda^s(p_{sm})$ different from p_{sm}. □

Lemma 4.4 *Let $p = (v_n \ \ldots \ v_1)$ be a valley-free path. Consider an attack where v_1 announces a path p_1 to v_2. Vertex v_n selects a path of class at least $f^{v_n}(p)$.*

Proof: We prove that each vertex v_i, with $i > 1$, in p selects a path p_i such that $f^{v_i}(p_i) \geq f^{v_i}(v_i \ \ldots \ v_1)$. In the base case ($n = 2$), the statement holds since $f^{v_2}(p_2) \geq f^{v_2}(v_2 \ v_1)$. In the inductive step ($n > 2$), by induction hypothesis and NE policy, vertex v_i receives a path p_{i-1} from vertex v_{i-1} such that $f^{v_{i-1}}(p_{i-1}) \geq f^{v_{i-1}}(v_{i-1} \ \ldots \ v_1)$. Two cases are possible: p_{i-1} contains v_i or not. In the second case, v_i selects a path $\lambda^{v_i}(p_i) \leq \lambda^{v_i}((v_i \ v_{i-1})p_{i-1})$ which implies that $f^{v_i}(p_i) \geq f^{v_i}(v_i \ \ldots \ v_1)$. In the first case, let p' be the sub-path of p_{i-1} from v_i. Observe that, since m has only origin-spoofing cheating capabilities, it cannot invalidate the acyclicity of G (the same does not hold if m has S-BGP cheating capabilities). As a consequence, since $(v_i \ v_{i-1})p_{i-1}$ is a valley-free path, vertex v_i is repeated in that path, and G is acyclic, we have that the class of p' must be higher than the class of $(v_i \ v_{i-1})p_{i-1}$. Hence, by inductive hypothesis, we have $f^{v_i}(p') > f^{v_i}((v_i \ v_{i-1})p_{i-1}) \geq f^{v_i}(v_i \ \ldots \ v_1)$, which implies that the statement holds also in this case. □

Theorem 4.5 *If the manipulator has origin-spoofing cheating capabilities and the length of the longest valley-free path is bounded by a constant, then problem* HIJACK *is in P.*

Proof: We tackle the problem with Alg. 1. First, observe that line 9 tests if a certain set of announcements causes a successful attack and, in that case, it returns the corresponding set of neighbors to whom m announces prefix π. Such test can be performed in polynomial time using the algorithm in [SSZ09]. Hence, if Alg. 1 returns without failure it is trivial to see that it found a successful attack in polynomial time. Suppose now that there exists a successful attack a^* from m that is not found by Alg. 1. Let p^*_{sm} be the path selected by s in attack a^*. Let A^* be the set of neighbors of m that receives prefix π from m in the successful attack.

Consider the iteration of the Alg. 1 where path p^*_{sm} is analyzed in the outer loop. At the end of the iteration Alg. 1 constructs a set A of neighbors of m. Let a be an attack from m where m announces π only to the vertices in A.

Algorithm 1 Algorithm for the HIJACK problem where m has origin-spoofing capabilities and the longest valley-free path in the graph is bounded.

```
1: Input: instance of HIJACK problem, m has origin-spoofing cheating capabilities;
2: Output: an attack pattern if the attack exists, fail otherwise;
3: let P_sm be the set of all valley-free paths from s to m;
4: for all p_sm in P_sm do
5:     let w be the vertex of p_sm adjacent to m; let A be a set of vertices and
       initialize A to {w}; let N be the set of the neighbors of m;
6:     for all n in N \ {w} do
7:         if there is no path p through (m, n) to a vertex x of p_sm such that
           f^x(p) > f^x(p_xm), where p_xm is the subpath of p_sm from x to m
           then
8:             insert n into A
9:         end if
10:    end for
11:    if the attack succeeds when m announces π only to the vertices in A
       then
12:        return A
13:    end if
14: end for
15: return fail
```

First, we prove that $A^* \subseteq A$. Suppose by contradiction that there exists a vertex $n \in A^*$ that is not contained in A. It implies that there exists a valley-free path p through (m, n) to a vertex x of p^*_{sm} such that $f^x(p) > f^x(p_{xm})$, where p_{xm} is the subpath of p^*_{sm} from x to m. Since m announces π to n, by Lemma 4.4, we have that x selects a path p' of class at least $f^x(p)$, that is a contradiction since p^*_{sm} would be disrupted by better class. Hence, $A^* \subseteq A$.

Now, we prove that attack a is a successful attack for m. Consider a valley-free path p_{sd} from s to d that does not traverse m and is preferred over p^*_{sm}. By Lemma 4.3 it is disrupted by better class in attack a^*. By Lemma 4.4, since $A^* \subseteq A$, we have that also in a path p_{sd} is disrupted by better class. Let x be the vertex adjacent to s in p_{sd}. Observe that vertex s cannot have an available path $(s\ x)p$ to d such that $\lambda^s((s\ x)p) < \lambda^s(p^*_{sm})$, because $(s\ x)p$ must be disrupted by better class.

Moreover, consider path p^*_{sm}. Since in a^* path p^*_{sm} is not disrupted by better class by a path to d, by Lemma 4.4, there does not exist a path p'_{xd} from

a vertex x of p^*_{sm} to d of class higher than p_{xm}, where p_{xm} is the subpath of p^*_{sm} from x to m. Hence, path p^*_{sm} cannot be disrupted by better class by a path to d. Also, observe that for each $n \in A$ there is no path p through (m, n) to a vertex x of p^*_{sm} such that $f^x(p) > f^x(p_{xm})$, where p_{xm} is the subpath of p^*_{sm} from x to m. Hence, p^*_{sm} can be disrupted only by same class. By Lemma 4.2, we have that s selects a path p such that $\lambda^s(p) \leq \lambda^s(p^*_{sm})$. Since path p cannot be a path to d, attack a is successful. This is a contradiction since we assumed that Alg. 1 failed.

Finally, since the length of the valley-free paths is bounded, the iterations of the algorithm where paths in P_{sm} are considered require a number of steps that is polynomial in the number of vertices of the graph. □

4.3 S-BGP Gives Hackers Hard Times

We open this section by strengthening the role of S-BGP as a security protocol. Indeed, S-BGP adds more complexity to the problem of finding an attack strategy (Theorem 4.1). After that we also provide an answer to a conjecture posed in [GSHR10] about hijacking and interception attacks in S-BGP when a single path is announced by the manipulator. In this case, we prove that every successful hijacking attack is also an interception attack (Theorem 4.4).

Theorem 4.1 *If the manipulator has S-BGP cheating capabilities and the length of the longest valley-free path is bounded by a constant, then problem* HIJACK *is NP-hard.*

Proof: We reduce from a version of 3-SAT where each variable appears at most three times and each positive literal at most once [Pap94]. Let F be a logical formula in conjunctive normal form with variables $X_1 \dots X_n$ and clauses $C_1 \dots C_h$. We build a BGP instance G (see Fig. 4.4) consisting of 4 structures: INTERMEDIATE, SHORT, LONG, and DISRUPTIVE.

The LONG structure is a directed path of length 6 with edges (s, w_1), (w_1, w_2), $\dots$, (w_4, w_5), and (w_5, d). The INTERMEDIATE structure consists of a valley-free path joining m and s. It has length 4 and it is composed by a directed path $(s\ j_3\ j_2\ j_1)$, and a directed edge (m, j_1). The SHORT structure has h directed paths from s to d. Each path has length at most 4 and has edges $(s, c_{i,1})$, $(c_{i,1}, c_{i,2})$, $\dots$, $(c_{i,v(C_i)}, d)$ $(1 \leq i \leq h)$, where $v(C_i)$ is the size of C_i. The DISRUPTIVE structure contains, for each variable X_i vertices, r_i, t_i, x_i, p_i and p'_i. Vertices, r_i, t_i, and x_i, are reached via long directed paths from m and are connected by (t_i, p_i), (x_i, p_i), (x_i, p'_i), (r_i, j_3), (t_i, j_3), and (p_i, d).

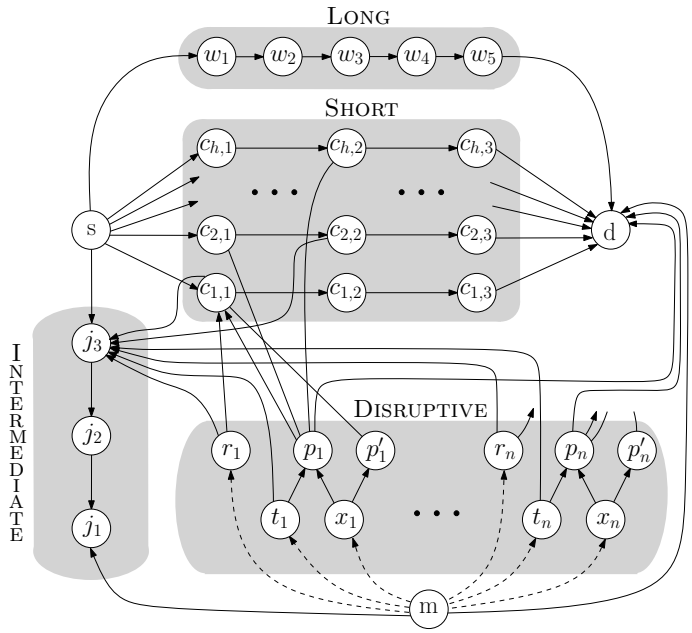

Figure 4.4: Reduction of a constrained 3-SAT problem to the HIJACK problem when m has S-BGP cheating capabilities.

Finally, suppose X_i occurs in clause C_j with a literal in position l. If the literal is negative the undirected edge $(p_i, c_{j,l})$ is added, otherwise, edges $(p_i, c_{j,l})$, $(r_i, c_{j,l})$, $(c_{j,l}, j_3)$, and undirected edge $(p'_i, c_{j,l})$ are added. An edge connects m to d. Vertices s, d, and m have source, destination, and manipulator roles, respectively.

Intuitively, the proof works as follows. The paths that allow traffic to go from s to m are only those passing through the DISRUPTIVE structure and the one in the INTERMEDIATE structure. Also, the path through the INTERMEDIATE structure is shorter than the one through the LONG structure, which is shorter than those through the DISRUPTIVE structure. If m does not behave maliciously, s receives only paths traversing the SHORT structure and the LONG structure. In this case s selects one of the paths in the SHORT structure according to its tie break policy. If m wants to attract traffic from s, then: (i) path (j_3 j_2 j_1 m d) must be available at s and (ii) all paths contained in

the SHORT structure must be disrupted by a path announced by m. If (i) does not hold, then s selects the path contained in either the LONG structure or the SHORT structure. If (ii) does not hold, then s selects a path contained in the SHORT structure.

Our construction is such that the 3-SAT formula is true iff m can attract the traffic from s to d. To understand the relationship with the 3-SAT problem, consider the behavior of m with respect to variable X_1 (see Fig. 4.4) that appears with a positive literal in the first position of clause C_1, a negative literal in the first position of C_2 and a negative literal in the second position of C_h.

First, we explore the possible actions that m can perform in order to disrupt paths in the SHORT structure. Since m has S-BGP cheating capabilities, m is constrained to propagate only the announcements it receives. If m does not behave maliciously, m receives path (d) from d and paths P_{r_1}, P_{t_1}, and P_{x_1} from r_1, t_1, and x_1, respectively. These paths have the following properties: P_{r_1} contains vertex $c_{1,1}$ that is contained in the path of the SHORT structure that corresponds to clause C_1; paths P_{t_1} and P_{x_1} both contain vertex p_1 and do not contain vertex $c_{1,1}$ since p_1 prefers $(p_1\ d)$ over $(p_1\ c_{1,1}\ c_{1,2}\ c_{1,3}\ d)$.

Now, we analyze what actions are not useful for m to perform an attack. If m issues any announcement towards t_1 or r_1 the path traversing the INTERMEDIATE structure is disrupted by better class. Also, if m sends a path P_{r_1}, P_{t_1}, or P_{x_1} towards r_j, t_j, or x_j, with $j = 2, \ldots, n$, the path traversing the INTERMEDIATE structure is disrupted by better class. Also, if m sends $(m\ d)$ to x_1, then the path traversing the INTERMEDIATE structure is disrupted from $c_{1,1}$ by better class. If m sends P_{x_1} to x_1, then it is discarded by x_1 because of loop detection. In each of these cases m cannot disrupt any path traversing the SHORT structure without disrupting the path traversing the INTERMEDIATE structure. Hence, m can disrupt a path in the SHORT structure without disrupting the path traversing the INTERMEDIATE structure only by announcing P_{r_1} and P_{t_1} from m towards x_1.

If path P_{t_1} is announced to x_1, then p_1 discards that announcement because of loop detection and path $(s\ c_{1,1}\ c_{1,2}\ c_{1,3}\ d)$ is disrupted from p'_1 by better class. Also, the path through the INTERMEDIATE structure remains available because the announcement through p'_1 cannot reach j_3 from $c_{1,1}$, otherwise valley-freeness would be violated. Hence, announcing path P_{t_1} corresponds to assigning the true value to variable X_1, since the only path in the SHORT structure that is disrupted is the one that corresponds to the clause that contains the positive literal of X_1.

If path P_{r_1} is announced to x_1, then $c_{1,1}$ discards that announcement because of loop detection and both paths ($s\ c_{2,1}\ c_{2,2}\ c_{2,3}\ d$) and ($s\ c_{h,1}\ c_{h,2}\ c_{h,3}\ d$) are disrupted by better class from p_1. Also, the path through the INTERMEDIATE structure remains available because the announcement through p_1 cannot reach j_3 from $c_{2,1}$ or $c_{h,2}$, otherwise valley-freeness would be violated. Hence, announcing path P_{r_1} corresponds to assigning the false value to variable X_1, since the only paths in the SHORT structure that are disrupted are the ones that correspond to the clauses that contain a negative literal of X_1.

Hence, announcing path P_{t_1} (P_{r_1}) from m to x_1 corresponds to assigning the true (false) value to variable X_1. As a consequence, m can disrupt every path in the SHORT structure without disrupting the path in the INTERMEDIATE structure iff formula F is satisfiable.

□

Theorem 4.2 *If the manipulator has S-BGP cheating capabilities and its degree is bounded by a constant, then problem* HIJACK *is in P.*

Proof: Observe that if the manipulator m has S-BGP cheating capabilities and the degree of the manipulator's vertex is bounded by a constant k, then problem HIJACK is in P. In fact, since m has at most k available paths plus the empty path, a brute force approach needs to explore $(k+1)^k$ possible cases.

□

To study the relationship between hijacking and interception we introduce the following technical lemma.

Lemma 4.3 *Let G be a BGP instance, let m be a vertex with S-BGP cheating capabilities, and let $d \neq m$ be any vertex of G. All vertices that admit a class c valley-free path to d not containing m have an available path of class c or better to d, irrespective of the paths propagated by m to its neighbors.*

Proof: Let $p = (v_n\ \ldots\ v_1)$ be a valley-free path to d not containing m. We prove by induction on vertices $v_1, \ldots, v_n$ that each vertex v_i has an available path of class $f^{v_i}(v_i\ \ldots\ v_1)$ or better. In the base case $i = 2$, v_2 is directly connected to d and the statements trivially holds. Suppose that vertex v_i, with $i > 2$, has an available path of class $f^{v_i}(v_i\ \ldots\ v_1)$. Hence, v_i selects a path p^* such that $f^{v_i}(p^*) \geq f^{v_i}(v_i\ \ldots\ v_1)$. Also, since $(v_{i+1}\ v_i\ \ldots\ v_1)$ is valley-free even $(v_{i+1}\ v_i)p^*$ is valley-free. Then, v_i announces (because of the NE policy) its best path p^* to v_{i+1}. There are two possible cases: either p^* does not contain v_{i+1} or it does. In the first case, path $(v_{i+1}\ v_i)p^*$ is available at v_{i+1} and the

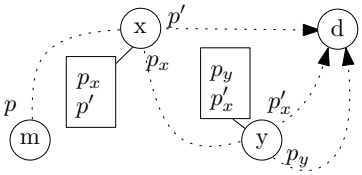

Figure 4.5: Proof of Theorem 4.4. (a) The order of paths in the boxes represents the preference of the vertices.

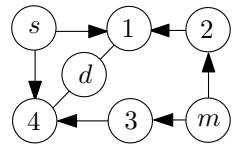

Figure 4.6: An instance where m cannot intercept traffic to d but it can hijack it.

statement holds. In the second case, consider the subpath $p^*_{v_{i+1}}$ of p^* from v_{i+1} to d. The statement easily follows because $f^{v_{i+1}}(p^*_{v_{i+1}}) \geq f^{v_{i+1}}((v_{i+1}\ v_i)p^*$. □

Theorem 4.4 *Let m be a manipulator with S-BGP cheating capabilities. If m announces the same path to any arbitrary set of its neighbors, then every successful hijacking attack is also a successful interception attack. If m announces different paths to different vertices, then the hijacking may not be an interception.*

Proof: We prove the following more technical statement that implies the first part of the theorem. Let G be a BGP instance, let m be a vertex with S-BGP cheating capabilities. Let p be a path available at m in the stable state S that is reached when m behaves correctly. Suppose that m starts announcing p to any subset of its neighbors. Let S' be the corresponding routing state. Path p remains available at vertex m in S'. The truth of the statement implies that m can forward the traffic to d by exploiting p.

Suppose for a contradiction that path p is disrupted in S' when m propagates it to a subset of its neighbors. Let x be the vertex of p closest to d that prefers a different path p_x (p is disrupted by p_x) in S' and let p' be the subpath of p from vertex d to x (see Fig. 4.5).

Observe that p is not a subpath of p_x as x cannot select a path that passes through itself. Since p_x is not available at x in S, let y be the vertex of p_x closest to d that selects a path p_y that is preferred over p'_x in S, where p'_x is the subpath of p_x from y to d.

We have two cases: either $f^x(p_x) > f^x(p')$ or $f^x(p_x) = f^x(p')$ (i.e., p_x is preferred to p' by better or by same class).

Suppose that $f^x(p_x) > f^x(p')$. By Lemma 4.3, since there exists a valley-free path p_x from x to d that does not traverse m, then x has an available path of class at least $f^x(p_x)$. Hence, x cannot select path p' in S, a contradiction.

Suppose that $f^x(p_x) = f^x(p')$. Two cases are possible: either p_y contains x or not. In the first case either $f^y(p_y) > f^y(p'_x)$ or $f^y(p_y) = f^y(p'_x)$. If $f^y(p_y) > f^y(p'_x)$, then we have that $f^y(p_y) \leq f^x(p') = f^x(p_x) \leq f^y(p'_x)$, a contradiction. If $f^y(p_y) = f^y(p'_x)$, we have that $|p'_x| < |p_x| \leq |p'| < |p_y|$. A contradiction since a longer path is preferred.

The second case ($f^x(p_x) = f^x(p')$ and p_y does not contain x) is more complex. We have that $|p'| \geq |p_x|$. Also, by Lemma 4.3, since p_y and p'_x do not pass through m, then y has an available path of class at least $\max\{f^y(p_y), f^y(p'_x)\}$. As y alternatively chooses p_y and p'_x we have that $f^y(p_y) = f^y(p'_x)$, which implies that $|p'_x| \geq |p_y|$. Denote by p_{xy} the subpath $(v_k \ \ldots v_0)$ of p_x, where $v_0 = y$ and $v_k = x$. Consider routing in state S. Two cases are possible: either $p_{xy}p_y$ is available at x or not. In the first case, since $|p'| \geq |p_x| = |p_{xy}p'_x| \geq |p_{xy}p_y|$, we have a contradiction because p' would not be selected in S. In the second case, we will prove that for each vertex $v_h \neq x$ in p_{xy} we have that $|p_h| \leq |(v_h \ \ldots \ v_0)p_y|$, where p_h is the path selected by v_h in S. This implies that $|(v_k \ v_{k-1})p_{k-1}| \leq |p_{xy}p_y| \leq |p_x| \leq |p'|$ and this leads to a contradiction. In fact, if $|(v_k \ v_{k-1})p_{k-1}| < |p'|$, then we have a contradiction because p' would not be selected in S. Otherwise, if $|(v_k \ v_{k-1})p_{k-1}| = |p'|$, we have that $|p_x| = |p'|$. Then, x prefers p_x over p' because of tie break. We have a contradiction since also $(v_k \ v_{k-1})p_{k-1}$ is preferred over p' because of tie break in S.

Finally, we prove that for each vertex $v_h \neq x$ in p_{xy} we have that $|p_h| \leq |(v_h \ \ldots v_0)p_y|$. This trivially holds for $v_0 = y$. We prove that if it holds for v_i then it also holds for v_{i+1}. If v_{i+1} selects $(v_{i+1} \ v_i)p_i$, then the property holds. Otherwise, $(v_{i+1} \ v_i)p_i$ is disrupted either by better class or by same class by a path p_{i+1}. In the first case, we have that either p_{i+1} traverses m or not. Suppose p_{i+1} traverses m and let q' be the neighbor of v_{i+1} on p_{i+1}. Observe that p is not necessarily a subpath of p_{i+1}. In fact, in the statement of the lemma, we only assumed that p is available at m, which does not imply that it is also selected at m. Now, since p_{i+1} disrupts $(v_{i+1} \ v_i)p_i$ by better class, then p_{i+1} is composed by a directed path from d to q' (that traverses m) and an edge (q', v_{i+1}) that can be either an oriented edge from q' to v_{i+1} or an unoriented edge. Let n be the neighbor of m on p and n' be the neighbor of n on p different from m. Consider the relationship between n and n'. Suppose n is a customer or a peer of n'. If m is a provider or a peer of n, then p is not valley-free and p cannot be available at m in S, which leads to a contradiction.

Otherwise, consider the case where m is a customer of n. In this case p is not a subpath of p_{i+1}. Otherwise, since m is a customer of n, we have that p_{i+1}, which passes through n and m, is not a directed path from d to q', which is a contradiction. Hence, since p_{i+1} does not traverse n and p_{i+1} is a directed path from d to q', m has to announce it to its provider n, which, in turn, would have preferred it over the subpath of p learnt from its peer or provider n'. This implies that p would not be available at m in S, that is a contradiction. Hence, n is a provider of n' and the subpath of p from d to n is a directed path. Since $f^x(p_x) = f^x(p')$, we have that also p_x is a directed path from d to x. Therefore, v_{i+1} is a provider of v_i and so $(v_{i+1}\ v_i)p_i$ would not be disrupted by better class in S, which is a contradiction. Hence, p_{i+1} does not traverse m. By Lemma 4.3, a path of a class better than $(v_{i+1}\ \ldots\ v_0)p'_x$ is available at v_{i+1} and so v_{i+1} cannot select $(v_{i+1}\ \ldots\ v_0)p'_x$ in S', a contradiction. In the second case ($(v_{i+1}\ v_i)p_i$ is disrupted by same class by a path p_{i+1}) we have that $|p_{i+1}| \leq |(v_{i+1}\ v_i)p_i| \leq |(v_{i+1}\ \ldots\ v_0)p_y|$. The second inequality comes from the induction hypothesis.

This concludes the first part of the proof. For proving the second part we show an example where m announces different paths to different neighbors and the resulting hijacking is not an interception. Consider the BGP instance in Fig. 4.6. In order to hijack traffic from s, vertices 1 and 4 must be hijacked. Hence, m must announce $(m\ 3\ 4\ d)$ to 2 and $(m\ 2\ 1\ d)$ to 3. However, since $(3\ 4\ d)$ and $(2\ 1\ d)$ are no longer available at m the interception fails.

□

4.4 Conclusions and Open Problems

Given a communication flow between two ASes we studied how difficult it is for a malicious AS m to devise a strategy for hijacking or intercepting that flow. This problem marks a sharp difference between BGP and S-BGP. Namely, while in a realistic scenario the problem is computationally tractable for typical BGP attacks it is NP-hard for S-BGP. This gives new evidence of the effectiveness of the adoption of S-BGP. It is easy to see that all the NP-hardness results that we obtained for the hijacking problem easily extend to the interception problem. Further, we solved a problem left open in [GSHR10], showing when performing a hijacking in S-BGP is equivalent to performing an interception.

Several problems remain open: (*i*) We focused on a unique m. How difficult is it to find a strategy involving several malicious ASes [GSHR10]? (*ii*) In [SZR10] it has been proposed to disregard the AS-paths length in the

BGP decision process. How difficult is it to find an attack strategy in this different model?

Chapter 5

Equal-Split Load-Balancing *

In this chapter, we switch our focus from route propagation predictability to the problem of selecting routing paths that make an efficient usage of network resources. To efficiently exploit network resources operators do traffic engineering (TE), i.e., adapt the routing of traffic to the prevailing demands. TE in large IP networks typically relies on configuring static link weights and splitting traffic between the resulting shortest-paths via the Equal-Cost-MultiPath (ECMP) mechanism. Yet, despite its vast popularity, crucial operational aspects of TE via ECMP are still little-understood from an algorithmic viewpoint. We embark upon a systematic algorithmic study of TE with ECMP. We consider the standard model of TE with ECMP and prove that, in general, even *approximating* the optimal link-weight configuration for ECMP within *any* constant ratio is an intractable feat, settling a long-standing open question. We establish, in contrast, that ECMP can provably achieve optimal traffic flow for the important category of Clos datacenter networks. We last consider a well-documented shortcoming of ECMP: suboptimal routing of large ("elephant") flows. We present algorithms for scheduling "elephant" flows on top of ECMP (as in, e.g., Hedera [AFRR$^+$10]) with *provable* approximation guarantees. Our results complement and shed new light on past experimental and empirical studies of the performance of TE with ECMP.

*Part of the material presented in this chapter is based on the following publications: M. Chiesa, G. Kindler, M. Schapira. Traffic Engineering with ECMP: an Algorithmic Perspective. In *Proc. INFOCOM*, IEEE, 2014.

5.1 Introduction

The rapid growth of online services (from video streaming to 3D games and virtual worlds) is placing tremendous demands on the underlying networks. To make efficient use of network resources, adapt to network conditions, and satisfy user demands, network operators do traffic engineering (TE), i.e., tune routing-protocol parameters to control how traffic is routed across the network. Our focus in this chapter is on the prevalent mechanism for engineering the flow of traffic within a single administrative domain (e.g., company, university campus, Internet Service Provider, and datacenter): TE with Equal-Cost-MultiPath (ECMP) [Hop00] via static link-weight configuration.

Most large IP networks run Interior Gateway Protocols, e.g., Open Shortest Path First (OSPF) [Moy98], to compute all-pairs shortest-paths between routers based on configurable static link weights (where a link's weight specifies its distance in the shortest-path computation). The ECMP feature was introduced to exploit shortest-path diversity by enabling the "split" of traffic between multiple shortest-paths via per-flow static hashing [CWZ00]. See Figure 5.1 for an illustration of shortest-path routing and ECMP traffic splitting on a simple network topology. Hence, today's TE often constrains the flow of traffic in two important respects: (1) traffic from a source to a destination in the network can only flow along the shortest paths between them (for the given configuration of link weights); and (2) traffic can only be split between multiple shortest paths (if multiple shortest paths exist) in a very specific manner (as illustrated in Figure 5.1).

Despite many proposals for alternative TE protocols and techniques, "traditional" TE with ECMP remains the prevalent mechanism for engineering the (intradomain) flow of traffic in today's Internet because, alongside its limitations, TE with ECMP has many advantages over other, more sophisticated schemes: stable and predictable paths, relatively low protocol overhead, implementation in existing hardware, simple configuration language, scalability, a built-in failure recovery mechanism, and more. Still, while ECMP is the subject of much empirical and experimental study (e.g., for ISP networks [FRT02] and for datacenter networks [GHJ+11]), even crucial operational aspects of TE with ECMP are little-understood from an algorithmic perspective: Can the configuration of link weights be done in a *provably* good manner? What conditions on network topologies lead to desirable TE guarantees? Can algorithmic insights aid in "fixing" ECMP's documented shortcomings, e.g., the suboptimal routing of large ("elephant") flows? We embark on a systematic algorithmic study of TE with ECMP. Our main contributions are discussed below.

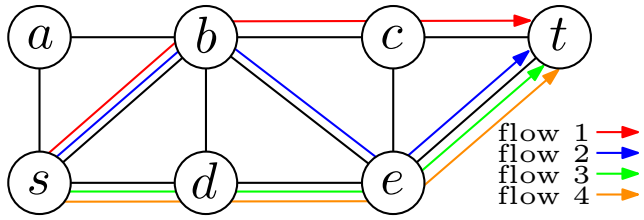

Figure 5.1: An illustration of Equal-Cost-MultiPath routing: 4 TCP connections, called "flows 1-4", originate at (source) router s and are destined for (target) router t. All link weights are 1. Observe that (s, b, c, t), (s, b, e, t) and (s, d, e, t) are (all) the induced shortest-paths from s to t. Each router now uses a static hash function on packet headers to map every connection to an outgoing link on a shortest-path to its destination, e.g., router s can map each of the flows 1-4 to the link (s, b) or the link (s, d) according to its hash function. The figure describes a possible mapping of flows to outgoing links.

Optimizing link-weight configuration? In practice, link weight configuration often relies on heuristics, such as setting link weights to be inversely proportional to capacity [Cis11]. While reasonable, these heuristics come with no guarantees. Can link-weight configuration be executed in a *provably* good manner? We consider the standard "splittable-flow model" of TE with ECMP, put forth by Fortz and Thorup [FT00, FT06, FT04], and the standard objective of minimizing the maximum link utilization. We settle a long-standing open question by proving a devastating impossibility result: No computationally-efficient algorithm can approximate the optimal link-weight configuration within *any* constant ratio. We show that this inapproximability result extends to other metrics of interest, e.g., maximizing total throughput and minimizing the sum of (exponentially-increasing) link costs (introduced in [FT00]). Our proof utilizes a new ("graph-power") technique for amplifying an inapproximability factor. We believe that this technique (somewhat inspired by the "diamond graph" in [LN04]) is of independent interest and may prove useful in other TE (and flow optimization, in general) contexts.

Optimizing ECMP performance on specific (datacenter) network topologies. The above negative result establishes that without imposing any restrictions on the network topology, TE with ECMP comes with no reasonable (provable) guarantees whatsoever. What about *specific* network topologies of

interest? What conditions on network topology imply good guarantees? We take the first steps in this research direction. We consider two recent proposals for datacenter network topologies: folded Clos networks (VL2 [GHJ+11]) and hypercubes (BCube [GLL+09], MDCube [WLL+09],). Our main positive result establishes that in the splittable-flow model, TE with ECMP is optimal for the important category of folded Clos networks. We show, in contrast, that for hypercubes, computing the optimal link weights for ECMP is NP-hard.

Our optimality result for folded Clos networks supports past experimental studied of ECMP in environments with fine-grained traffic splitting. [AFRR+10] shows that ECMP routing of small ("mice") flows in Clos networks leads to good network performance. To avoid TCP packet reordering, ECMP routing splits traffic across multiple paths at an (IP-)flow-level granularity, that is, packets belonging to the same IP flow traverse the same path. [DPHK13] advocates replacing today's ECMP traffic splitting scheme with packet-level traffic splitting (i.e., allowing the "spraying" of packets belonging to the same flow across multiple paths). [DPHK13] shows, via extensive simulations, that "ECMP-like" traffic splitting at packet-level granularity leads to significantly better load-balancing of traffic in folded Clos networks. Our optimality result provides a strong theoretical justification for this claim.

Optimizing the routing of elephant flows. As explained above, ECMP splits traffic across multiple paths at an (IP-)flow-level granularity. Consequently, a key limitation of ECMP is that large, long-lived ("elephant") flows traversing a router can be mapped to the same output port. Such "collisions" can cause load imbalances across multiple paths and network bottlenecks, resulting in substantial bandwidth losses [DPHK13, AFRR+10]. Beyond transitioning to ECMP traffic splitting at packet-level, researchers have also examined other possible approaches to alleviating this. Recent studies, e.g., Hedera [AFRR+10] and DevoFlow [CMT+11], call for dynamically scheduling elephant flows in datacenter (folded Clos) networks so as to minimize traffic imbalances (while still routing mice flows with ECMP). We now focus on the unsplittable-flow model, which captures the requirement that all packets in a flow (be it long-lived or short-lived) traverse the same path, and investigate the approximability of elephant flow routing. We show that this task is intractable and devise algorithms for approximating the (unattainable) optimum. We discuss the connections between our algorithmic results and past experimental studies along these lines.

Organization. We present the standard ECMP routing model in Section 5.2.

Our inapproximability result for optimizing link-weight configuration is presented in Section 5.3. We discuss our results for TE with ECMP for specific (datacenter) network topologies (folded Clos networks and hypercubes) in Section 5.6. Our results for scheduling elephant flows in folded Clos networks appear in Section 5.7. We conclude and present directions for future research in Section 5.9.

5.2 EMCP Routing Model

We now present the standard model of TE with ECMP from [FT04]. We refer the reader to [FT06, FT00, FT04] for a more thorough explanation of the model and its underlying motivations. We shall revisit some of the premises of this model in Section 5.7.

Network and traffic demands. The network is modeled as an undirected graph $G = (V, E)$, where each edge $e \in E$ has fixed capacity c_e. Vertices in V represent routers and edges (links) in E represent physical communication links between routers. We are given a $|V| \times |V|$ demand matrix D such that, for each pair $s, t \in V$, the entry D_{st} specifies the volume of traffic, in terms of units of flow, that (source) vertex s sends to (target) vertex t.

Flow assignments. A flow assignment is a mapping $f : V \times V \times E \to \mathbb{R}^+ \setminus \{0\}$. $f(s, t, e)$ represents the amount of flow from source s to target t traversing edge e. Let $f_e = \Sigma_{s,t \in V} f(s, t, e)$, that is, f_e denotes the total amount of flow traversing edge e . We restrict out attention (unless stated otherwise) to flow assignments that obey two conventional constraints: (1) flow conservation: $\forall v \in V$, $\forall s, t \in V$ such that $v \neq s$ and $v \neq t$, $\Sigma_{e \in E_v} f(s, t, e) = 0$, where E_v is the set of v's incident edges in E; (2) demand satisfaction: for all $s, t \in V$ $\Sigma_{e \in O_s} f(s, t, e) = \Sigma_{e \in O_t} f(s, t, e) = D_{s,t}$. (Observe that in some scenarios a flow satisfying the two above conditions must exceed the capacity of some link, i.e., $f_e > c_e$ for some edge e).

Link-weight configurations and routing. A link-weight configuration is a mapping from edges to nonnegative "weights" $w : E \to \mathbb{R}^+ \setminus \{0\}$. Every such link-weight configuration w induces the unique flow assignment that adheres to the following two conditions:

- **Shortest-path routing.** Link weights in w induce shortest paths between all pairs of vertices, where a path's length is simply the sum of its link weights. All units of flow sent from source s to target t must be

routes along the resulting shortest-paths between them. We next explain how traffic is split between multiple shortest paths.

- **Equal splitting.** All units of flow traversing a vertex v en route to a given target vertex t are equally split across all of v's outgoing links on shortest-paths from v to target t.

Optimizing link weight configuration. We study the optimization of link-weight configuration for ECMP routing. We consider 3 optimization goals:

- MIN-ECMP-CONGESTION (MEC). A natural and well-studied optimization goal is to minimize the maximum link utilization, that is, to engineer a flow assignment f (via link-weight configuration) so that $max_{e \in E} \frac{f_e}{c_e}$ is minimized.
- MIN-SUM-COST. Another optimization goal that has been studied in the context of TE with ECMP is MIN-SUM-COST [FT00, FT06, FT04, SGD05]: minimizing the sum of edge-costs under a given flow $\Sigma_e \phi(\frac{f_e}{c_e})$, where ϕ is an exponentially-increasing cost function, e.g., $\phi(x) = 2^x$.
- MAX-ECMP-FLOW (MEF). MEF can be regarded as the straightforward generalization of classical max-flow objective to the multiple sources / multiple targets (i.e., multicommodity flow) setting. Here the goal is to send as much traffic through the network while (i) not exceeding the demands in D (i.e., possibly violating "demand satisfaction", as defined above) and (ii) not exceeding the link capacities.

Approximating the optimum. While in some scenarios computing the optimal solution with respect to the above optimization goals is tractable, in other scenarios this task is NP-hard. We therefore also explore the *approximability* of these goals. We use the following standard terminology. Let $\mathcal{A}$ be an algorithm for a minimization problem P. For every instance I of P, let $\mathcal{A}(I)$ denote the value of $\mathcal{A}$'s outcome for I and $OPT(I)$ denote the value of the optimal solution for I. $\mathcal{A}$ is a polynomial-time α-*approximation* algorithm for P for $\alpha \geq 1$ if $\mathcal{A}$ runs in polynomial time and for any instance I of P, $\mathcal{A}(I) \leq \alpha \cdot OPT(I)$. Similarly an algorithm $\mathcal{A}$ is a polynomial-time α-*approximation* algorithm for a maximization problem P, for $\alpha \geq 1$, if $\mathcal{A}$ runs in polynomial time and, for any instance I of P, $\mathcal{A}(I) \geq \frac{OPT(I)}{\alpha}$.

5.3 TE with ECMP is Inapproximable!

We settle a long-standing question by showing that optimizing link-weight configuration for ECMP is not only NP-hard but cannot, in fact, be approximated within any "reasonable" factor (unless P=NP) with respect to all 3 optimization goals discussed in Section 5.2: MIN-ECMP-CONGESTION, MIN-SUM-COST, and MAX-ECMP-FLOW. Remarkably, these inapproximability results hold even when the demand matrix has a single nonzero entry, i.e., when only a single router aims to send traffic to another router. Hence, in general, configuring link-weights for ECMP cannot be done in a provably good manner.

Theorem 5.1 *No computationally-efficient algorithm can approximate the optimum with respect to* MIN-ECMP-CONGESTION*,* MIN-SUM-COST *or* MAX-ECMP-FLOW*, within any constant factor $\alpha \geq 1$ unless $P = NP$, even when the demand matrix has a single nonzero entry.*

The remainder of the section provides a proof of Theorem 5.1 for the MIN-ECMP-CONGESTION, MAX-ECMP-FLOW, and MIN-SUM-COST objectives.

We henceforth focus on the scenario that the demand matrix has a single nonzero entry. Below, we discuss the three main ingredients of the proof of Theorem 5.1: (1) a new graph-theoretic problem called "MAX-ECMP-DAG", which we prove is inapproximable within a small constant factor (Section 5.3); (2) amplifying this inapproximability result for MAX-ECMP-DAG via a new technique to establish that MAX-ECMP-DAG is not approximable within *any* constant factor (Section 5.3); and (3) showing that our inapproximability result for MAX-ECMP-DAG implies similar results for both MIN-ECMP-CONGESTION and MAX-ECMP-FLOW (Section 5.3).

Max-ecmp-DAG

Max-ecmp-DAG. We present the following graph-theoretic problem called "MAX-ECMP-DAG" (MED). In MAX-ECMP-DAG, the input is a capacitated directed acyclic graph (DAG) H and a single source-target pair of vertices (s, t) in H. We associate with every sub-DAG $\bar{H}$ of H that contains s and t a flow assignment $f_{\bar{H}}$ as follows. Given $\bar{H}$, the flow assignment $f_{\bar{H}}$ is the max-flow from s to t in $\bar{H}$ subject to the constraint that every vertex in $\bar{H}$ split outgoing flow equally between all of its outgoing edges in $\bar{H}$. The objective in MAX-ECMP-DAG is to find the sub-DAG of H for which the induced flow is maximized, i.e., $\max_{\bar{H}} |f_{\bar{H}}|$.

Inapproximability result for Max-ecmp-DAG. We prove that MAX-ECMP-DAG is inapproximable within a (small) constant factor via a reduction from a hardness result for MIN-ECMP-CONGESTION in [FT04]. We shall later amplify this inapproximability ratio in Section 5.3 using our new amplification technique.

Theorem 5.2 *Given a* MAX-ECMP-DAG *instance I, distinguishing between the following two scenarios is NP-hard:*

- $OPT(I) = 1$
- $OPT(I) = \frac{2}{3}$

where $OPT(I)$ is the value of the optimal solution for I.

Observe that Theorem 5.2 implies that MAX-ECMP-DAG cannot be approximated within a factor of $\frac{3}{2}$ (unless P=NP).

To prove it, we first show that, in the single source-target pair case, an optimal solution for a MIN-ECMP-CONGESTION instance is also an optimal solution for a MAX-ECMP-FLOW instance and vice versa. We then show that every solution for a MAX-ECMP-DAG instancecan be translated into an "equivalent" solution for a MAX-ECMP-FLOW instance of the same graph. The following theorem has been proved by Fortz and Thorup [FT04].

Theorem 5.3 *Given a* MIN-ECMP-CONGESTION *instance I, with multiple unit flow demands in D, distinguishing between the following two scenarios is NP-hard:*

- $OPT_{MEC}(I) = 1$
- $OPT_{MEC}(I) = \frac{3}{2}$

Based on Theorem 5.3, we first prove that the same result holds even in the more restrictive case where there only exists a single source-target pair.

Lemma 5.4 *Given a* MIN-ECMP-CONGESTION *instance I with a single source-target pair, distinguishing between the following two scenarios is NP-hard:*

- $OPT_{MEC}(I) = 1$.
- $OPT_{MEC}(I) = \frac{3}{2}$.

Proof: Let $I = (G, D)$ be a MEC instance such that OPT_{MEC} is either 1 or $\frac{3}{2}$, where $F = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ is the set of source-target pairs of vertices of G with $D_{s_i,t_i} = 1$. For sake of simplicity, we assume that k is a power of 2. Create a copy G' of G and D' of D. Add a new source vertex s into G' and connect it to all vertices $s_1, \ldots, s_k$ with a binary tree rooted at s. Add a new target vertex t and connect it with an edge to all vertices $t_1, \ldots, t_k$. Let $D_{s,t} = |F|$, $D_{x,y} = 0$ for $x \neq s$ and $y \neq t$, and set the capacity of each edge of the binary tree incident to a source (target) vertex s_i (t_i) to 1 and all the remaining edges of both binary trees to infinite. We now show that $OPT_{MEC}((G, D)) = OPT_{MEC}((G', D'))$. It is easy to see that $OPT_{MEC}((G, D)) \geq OPT_{MEC}((G', D'))$. In fact, if (i) flow demand $D_{s,t}$ is split among every edge in the binary tree that join s to all vertices $s_1, \ldots, s_k$, (ii) each flow from s_i is routed as in the optimal solution for I, and (iii) each flow is routed from each t_i directly to t, then the value of this solution will be equal to $OPT_{MEC}((G, D))$. By observing that an unequal splitting through the binary tree from s to vertices $s_1, \ldots, s_k$ causes a congestion of 2, our lemma easily derives from Theorem 5.3.

□

We denote by $I = (G, s, t)$ an instance of both MAX-ECMP-FLOW (MIN-ECMP-CONGESTION) with a single source-target (unit) flow demand.

Lemma 5.5 *A link weight assignment for a graph G is optimal for an instance (G, s, t) of* MIN-ECMP-CONGESTION *if and only if it is optimal for an instance (G, s, t) of* MAX-ECMP-FLOW.

Proof: It is easy to see that, given an optimal solution for an instance $I = (G, s, t)$ of MIN-ECMP-CONGESTION, by scaling the amount of flow sent from s to t by a factor of $\frac{1}{OPT_{MEC}(I)}$, each edge will have congestion at most 1. Hence, $OPT_{MEF}(I) \geq \frac{1}{OPT_{MEC}(I)}$. Vice versa, given an optimal solution for an instance $I = (G, s, t)$ of MAX-ECMP-FLOW, by scaling the amount of flow sent from s to t by a factor of $\frac{1}{OPT_{MEF(I)}}$, each edge will have congestion at most $\frac{1}{OPT_{MEF(I)}}$ and a unit of flow will be routed from s to t. Hence, $OPT_{MEC}(I) \leq \frac{1}{OPT_{MEF}(I)}$.

□

Corollary 5.6 *Given a* MAX-ECMP-FLOW *instance I with a single source-target pair, distinguishing between the following two scenarios is NP-hard:*

- $OPT_{MEF}(I) = 1$

- $OPT_{MEF}(I) = \frac{2}{3}$

Proof: It easily follows by Lemma 5.5 and Theorem 5.4.

□

We now show that every solution for a MAX-ECMP-DAG instance can be translated in a link weight assignment that is associated with the same flow assignment on the graph. Hence, finding an optimal (s,t)-DAG is equivalent to compute an optimal link weight assignment, which justifies our reduction to the MAX-ECMP-DAG problem. We now introduce some useful notation. We say that a flow assignment f on a graph G is *realized* by a weight assignment w, if setting link weights of G as in w, flows are routed according to f. Given a link weight assignment w, let $\mathcal{B}(w)$ denote the oriented subgraph of G containing the edges that are traversed by a flow, which are oriented according to the direction of the flow. Since flows are routed according to the shortest-path criterium, $\mathcal{B}(w)$ is a directed acyclic graph (DAG) with a single source s and a single sink t.

Lemma 5.7 *For any arbitrary sub-DAG A with a single source s and a single sink t of a graph G, there exists a link weight assignment w such that $\mathcal{B}(w)$ is equal to A.*

Proof: We denote by $out(v, A)$ the set of vertices of A that have an ingoing edge from vertex v. and by $sp(v, t, w)$ the length of the shortest-paths from vertex v to vertex t of G according to a link weight assignment w. Consider a topological order $(v_1, \ldots, v_n)$ of the vertices of A, where $v_1 = t$ and $v_n = s$. We compute a link weight assignment w by the following procedure that processes vertices from v_2 to v_n. For each vertex v_i, with $i = 2, \ldots, n$, let M be the length of the longest shortest path from any neighbor of $v_i \in out(v_i, A)$,i.e., $M = \max_{v_l \in out(v_i,A)}\{sp(v_l, t, w)\}$. For each vertex $v_k \in out(v_i, G)$, we set $w((v_k, v_i)) = M + 1 - sp(v_k, t, w)$. This guarantees that all the edges in $out(v_i, A)$ belong to at least one shortest path from v_i to t. Observe that, since vertices are processed in topological order, we have that $l < i$ and $k < i$, which implies that both $sp(v_l, t, w)$ and $sp(v_k, t, w)$ has already been recursively computed. Observe that this assignment implies that the shortest path in G from s to t is at most $|E(G)|$. For this reason, for each edge $e \in E(G) \setminus E(A)$, we set $w(e) = |E(G)| + 1$. This guarantees that every shortest path between s and t does not pass through an edge that is not contained in A.

□

This theorem implies that for any instance (G, s, t) of MAX-ECMP-FLOW with a single source-target unit flow from s to t, we have that $OPT_{MF}((G, s, t)) = OPT((G, s, t))$. Hence, combining this with Lemma 5.7 leads to our constant inapproximability result for MAX-ECMP-DAG.

Theorem 5.2. *Given a* MAX-ECMP-DAG *instance I, distinguishing between the following two scenarios is NP-hard:*

- $OPT(I) = 1$
- $OPT(I) = \frac{2}{3}$

where $OPT(I)$ is the value of the optimal solution for I.

Amplifying the Inapproximability Gap

We can now leverage Theorem 5.2 to prove that that MAX-ECMP-DAG is not approximable within any constant factor.

Amplifying the inapproximability gap: a new technique. Our proof relies on a new technique for amplifying an inapproximability gap. Roughly speaking, we show how to create, given an instance I_0 of MAX-ECMP-DAG, a new, polynomially-bigger, instance I_1 of MAX-ECMP-DAG such that $OPT(I_1) = (OPT(I_0))^2$. Observe that as distinguishing between the scenario that $OPT(I_0) = 1$ and the scenario that $OPT(I_0) = \frac{2}{3}$ is NP-hard, distinguishing between the scenario that $OPT(I_1) = 1$ and the scenario that $OPT(I_1) = (\frac{2}{3})^2$ is also NP-hard. By applying this idea multiple times the inapproximability gap can be further amplified to an arbitrary (constant) factor.

The $\otimes$ operator: intuition. We now sketch the key tool used in our proof technique. We define the "$\otimes$ operator" that, given two MAX-ECMP-DAG instances, constructs a new MAX-ECMP-DAG instance. Before formally defining the $\otimes$ operator, we illustrate its use via the example in Figure 5.2. Consider the MAX-ECMP-DAG instance I_0 in Figure 5.2. The numbers in black are edge capacities and the orange arrows indicate the direction of the edges. Observe that the optimal solution for I_0 is the sub-DAG that contains the edges $(s, a), (a, b), (b, t)$, and (a, t) and that the value of this solution is 9. Specifically, the optimal solution routes 9 units of flow through (s, a), which are then equally split between(a, b) and (a, t), and the 4.5 units of flow entering vertex b are then sent directly to t. Now, consider the instance I_1 of MAX-ECMP-DAG, shown in Fig. 5.3, that is obtained from I_0 as follows. Let G_0 be

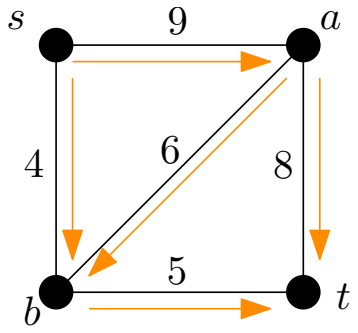

Figure 5.2: Graph G_0.

the network graph in I_0. We replace each edge (u, v) in G_0 with an exact copy of G_0. We connect vertex u to the source vertex in this copy of G_0 and vertex v to the target vertex. The capacity of each edge in this copy of G_0 is set to be its original capacity in G_0 multiplied by the capacity of (u, v). The capacities of the edges connecting vertices u and v to this copy of G_0 are set to be ∞.

We argue that the optimal solution for I_1, $OPT(I_1)$ is $f^* = 9^2 = 81$. We now provide some intuition for this claim. Let G_1 be the network graph in I_1. Consider $G_{(u,v)}$, the copy of G_0 that was used in the construction of I_1 to replace the edge (u, v) in G_0. Specifically, consider $G_{(a,b)}$, with $V(G_{(a,b)} = \{s_{a,b}, a_{a,b}, b_{a,b}, t_{a,b},\}$ and $E(G_{(a,b)}) = \{(s_{(a,b)}, a_{(a,b)}), (s_{(a,b)}, b_{(a,b)}), (a_{(a,b)}, b_{(a,b)}), (a_{(a,b)}, t_{(a,b)}), (b_{(a,b)}, t_{(a,b)})\}$. Observe that the optimal sub-DAG of $G_{(a,b)}$ in terms of maximizing the flow from $s_{(a,b)}$ to $t_{(a,b)}$ is precisely as in the optimal solution for I_0. Observe also that the value of the optimal solution within $G_{(a,b)}$ is 9×6, that is, f^* multiplied by the capacity of the edge (a, b) in G_0. Similarly, every subgraph $G_{(u,v)}$ can route a flow of $f^* \times c_{G_0}((u, v))$, where $c_{G_0}((u, v))$ is the capacity of the edge (u, v) in G_0. Hence, the network graph G_1 can be abstracted as in Figure 5.4 (replacing each copy of G_0 by a single edge with the appropriate capacity). A simple argument shows that the optimal solution in this instance of MAX-ECMP-DAG has value $(f^*)^2$, the value of the optimal solution in I_0 multiplied by a scaling factor of f^*.

The $\otimes$ operator: formal definition. Let I_1 and I_2 be two MAX-ECMP-DAG instances. We now define the operation $I_1 \otimes I_2$. Let G_1 and G_2 be the network graphs in I_1 and I_2, respectively. $I = I_1 \otimes I_2$ is an instance of MAX-ECMP-DAG with network graph G constructed as follows. We create,

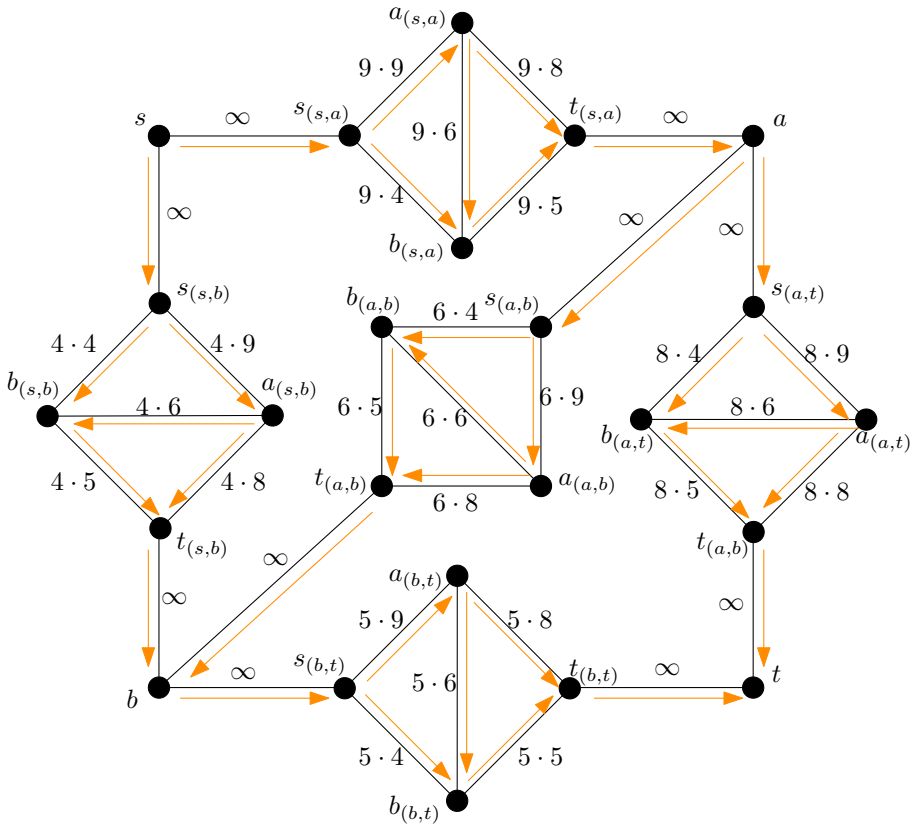

Figure 5.3: Graph G_1.

for every edge $e \in E(G_1)$, a copy of G_2, G_e. Let s_e and t_e denote the source and target vertices in G_e, respectively. The set of vertices in G consists of the vertices in $V(G_1)$ and also of the vertices in all $V(G_e)$'s, i.e., $V(G) = V(G_1) \bigcup_{e \in E(G_1)} V(G_e)$. The set of edges in G contains all the edges in the different $E(G_e)$'s, and also the edges (u, s_e) and (t_e, v) for every edge $e = (u, v) \in E(G_1)$, i.e., $E(G) = \bigcup_{e=(u,v) \in E(G_1)} (\{(u, s_e)(t_e, v)\} \cup E(G_e))$. The capacity of every edge in G_e is set to be the capacity of the corresponding edge in G_2 multiplied by the capacity of e in I_1. The capacity of every edge of the form (u, s_e) or (t_e, v) is set to ∞.

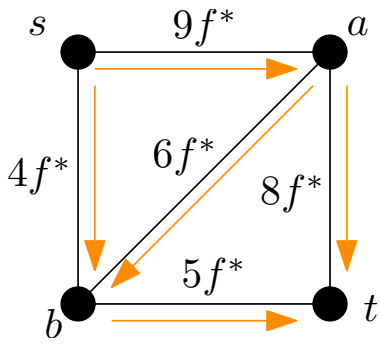

Figure 5.4: Abstraction of G_1.

Gap amplification via the $\otimes$ operator. We prove a crucial property of the $\otimes$ operator: applying the $\otimes$ operator to an instance I of MAX-ECMP-DAG k times increases the value of the optimal solution from $OPT(I)$ in the original instance I to $(OPT(I))^k$ in the resulting new instance of MAX-ECMP-DAG.

Lemma 5.8 *Let I be an instance of* MAX-ECMP-DAG. $OPT(\otimes^k I) = (OPT(I))^k$ *for any integer $k > 0$.*

Proof: Let $I = \otimes^0 I$ be a MAX-ECMP-DAG instance. Recall that I is a DAG with a single source s and sink t. We prove this lemma by induction on k. Let $\bar{H}_0$ be an optimal solution for $\otimes^0 I$, that is, a sub-DAG of I.

In the base case $k = 0$, we have that $OPT(\otimes^0 I) = OPT(I)^1$, which is true since $\otimes^0 I = I$.

In the inductive case $k > 0$, let $I_k = \otimes^k I$ and $I_{k+1} = \otimes^{k+1} I$. Let $\bar{H}_k$ be an optimal solution for I_k. We prove that there exists a sub-DAG $\bar{H}_{k+1}$ of I_{k+1} such that $OPT(I_{k+1}) = OPT(I)^{k+2}$. First, we prove that $OPT(I_{k+1}) \geq OPT(I)^{k+2}$. Recall that, each edge e of I_k with capacity $c_{I_k}(e) \neq \infty$ is replaced in I_{k+1} by a DAG H_e, where the capacity of each edge of H_e is multiplied by a factor $c_{I_k}(e)$. Consider a solution $\bar{H}_{k+1}$ of I_{k+1} constructed as follows. For each vertex of I_{k+1} that is not contained in any graph H_e (i.e, each vertex in common with I_k), we split the traffic according to the optimal solution in I_k, i.e., for each edge $(x, y) \in E(\bar{H}_k)$ add $(x, s_{(x,y)})$ and $(t_{(x,y)}, y)$ into $\bar{H}_{k+1}$. Moreover,for each subgraph H_e, with $e \in E(I_k)$, we split traffic as $\bar{H}_0$ does in I. Namely, for each subgraph H_e, where $e \in E(\bar{H}_k)$, for each edge $(x, y) \in E(\bar{H}_0)$ add (w_e, y_e) into $E(\bar{H}_{i+1})$. Observe that we can route through H_e a flow that

is $OPT(I_0)$ times larger than $c_{I_k}(e)$. Therefore, the maximum flow in I_{k+1} is $OPT(I_0) \cdot OPT(I_k) = OPT(I_0 =) \cdot OPT(I_0)^{k+1} = OPT(I_0)^{k+2}$, which implies $OPT(I_{k+1}) \geq OPT(I)^{k+2}$.

Now, we prove that $OPT(I_{k+1}) \leq OPT(I)^{k+2}$. Suppose, by contradiction, that there exists a sub-DAG $\bar{H}_{k+1}$ of I_{k+1} such that $f_{\bar{H}_{k+1}} > OPT(I)^{k+2}$. Construct a sub-DAG $\bar{H}_k$ of I_k as follows. For each directed edge $(v, u_{(x,y)}) \in E(\bar{H}_{k+1})$, where v is a vertex of I_{k+1} in common with I_k and $u_{(x,y)}$ is the source or target vertex of any subgraph $H_{(x,y)}$, add (v, y) into $E(\bar{H}_k)$ if $y \neq v$, otherwise add (v, x). Since each edge e of I_k can route a flow $OPT(I)$ times smaller than its corresponding subgraph H_e of I_{k+1}, we have that the maximum flow through $\bar{H}_k$ is at least $\frac{f_{\bar{H}_{k+1}}}{OPT(I)} > \frac{OPT(I)^{k+2}}{OPT(I)} = OPT(I)^{k+1}$, which is a contradiction, since, by induction hypothesis, we have that $OPT(I_k) = OPT(I)^{k+1}$. □

Lemma 5.8 can now be used to prove that no constant approximation ratio is achievable for Max-ecmp-DAG. Recall that, by Theorem 5.2, distinguishing, for a given a Max-ecmp-DAG instance I, between the following two scenarios in NP-hard: (1) $OPT(I) = 1$; and (2) $OPT(I) = \frac{2}{3}$. Observe that when combined with Lemma 5.8 this implies that distinguishing, for a given a Max-ecmp-DAG instance I, between the following two scenarios is also NP-hard: (1) $OPT(I) = 1$; and (2) $OPT(I) = (\frac{2}{3})^k$ for any constant integer $k > 0$.

Relating Max-ecmp-DAG to Min-ecmp-Congestion and Max-ecmp-Flow

Given an instance H_0 of Max-ecmp-DAG, let G_k be a copy of $\otimes^k H_0$ with undirected edges. Let s and t be the source and sink vertices of H_0. We denote by I_k an instance (G_k, s, t) of Max-ecmp-Flow. We introduce a property of a graph instance that will be used to exploit the amplification technique in Min-ecmp-Congestion and Max-ecmp-Flow.

Reversibility. We say that a mef instance $I = (G, s, t)$ is *non-reversible* if $OPT_{MEF}(I) \geq OPT_{MEF}((G, t, s))$. Similarly, a mec instance I is non-reversible if $OPT_{MEC}(I) \leq OPT_{MEC}((G, t, s))$.

Lemma 5.9 *Let $I = (G, s, t)$ be a* mef *instance with $OPT_{MEF} = \{k, \frac{2}{3}k\}$, with $k > 0$. It is possible to construct in polynomial time a non-reversible instance I' such that $OPT_{MEF}(I) = OPT_{MEF}(I')$.*

Proof: If G is non-reversible, we create $I' = (G', s', t)$ as a copy of I where s' is a new vertex added into $V(G')$. Moreover, we add four vertices $v_1, v_2.v_3$, and v_4 and connect them to s through a path (v_1, v_2, v_3, v_4, s), with capacity k. Then, we connect s' to each vertex v_1, v_2, v_3, v_4 with an edge of capacity $\frac{1}{4}k$. Observe that, by construction, the maximum flow from s to s' is $\frac{1}{4}k + \frac{1}{8}k + \frac{1}{16}k + \frac{1}{32}k$, while the maximum flow from s' to s is k since s' can send a flow of value $\frac{1}{4}k$ to v_1, v_2, v_3, and v_4, respectively. Hence,

$$\begin{aligned} &OPT_{MEF}(G, s', t) = \\ &= \min\{OPT_{MEF}(G', s', s), OPT_{MEF}(G', s, t)\} = \\ &= \min\{k, OPT_{MEF}(G, s, t)\} = OPT_{MEF}(G, s, t) \geq \\ &\geq \frac{2}{3}k \geq \frac{1}{4}k + \frac{1}{8}k + \frac{1}{16}k + \frac{1}{32}k = \\ &= OPT_{MEF}(G', s, s') \geq OPT_{MEF}(G', t, s'), \end{aligned}$$

which means that I' is non-reversible.

□

Corollary 5.10 *Let $I = (G, s, t)$ be a* MEC *instance with $OPT_{MEC} = \{k, \frac{3}{2}k\}$, with $k > 0$. It is possible to construct in polynomial time a non-reversible instance I' such that $OPT_{MEC}(I) = OPT_{MEC}(I')$.*

Proof: It easily follows by Lemma 5.5. □

Relating Max-ecmp-DAG to Max-ecmp-Flow. Given an instance H_0 of MAX-ECMP-DAG such that its undirected copy G_0 is non-reversible. Let s and t be the source and sink vertices of H_0. We say that a flow assignment f in G_0 is *compliant* with H_0 if for every edge $(x, y) \in E(G_0)$, if a flow is routed from x to y, then $(x, y) \in E(H_0)$. We say that a directed acyclic graph H' is an *orientation* of an undirected graph G_0 if at least an optimal flow assignment f of G is compliant with H. We denote $H_k = \otimes^k H_0$, by G_k the undirected copy of H_k, and by I_k an instance (G_k, s, t) of MAX-ECMP-FLOW.

Lemma 5.11 *Suppose that in at least one optimal solution for (G_0, s, t) the corresponding flow assignment is compliant with H_0. Then, $OPT(H_k) = OPT_{MEF}(I_k)$.*

Proof: We prove it by induction. In the base case $k = 0$, the statement of the lemma holds since instance I_0 is such that it has an optimal solution that is compliant with H_0. In the inductive case $k > 0$, by Lemma 5.7 we know that $OPT_{MEF}(I_k) = OPT(H_k) = OPT(H_0)^{k+1}$ and $OPT_{MEF}(I_{k+1})$ is at least $OPT(H_{k+1}) = OPT(H_0)^{k+2}$. We want to show that $OPT_{MEF}(I_{k+1}) \leq OPT(H_0)^{k+2}$. Suppose, by contradiction, that there exists a sub-DAG $\bar{H}_{k+1}$ of I_{k+1} such that $f_{\bar{H}_{k+1}} > OPT(I)^{k+2}$. Construct a sub-DAG $\bar{H}_k$ of I_k as follows. For each directed edge $(v, u_{(x,y)}) \in E(\bar{H}_{k+1})$, where v is a vertex of I_{k+1} in common with I_k and $u_{(x,y)}$ is the source or target vertex of any sub-graph $H_{(x,y)}$, add (v, y) into $E(\bar{H}_k)$ if $y \neq v$, otherwise add (v, x). Recall that, since I^0 is non-reversible, for each graph H_e, we have $OPT_{MEF}(H_e, t_e, s_e) \leq OPT_{MEF}(H_e, s_e, t_e) = OPT_{MEF}(H_0, s, t) \cdot c_{I_k}(e)$, which means that, for each edge e of H_k, we can route at least a flow $OPT_{MEF}(I_0)$ times smaller than in H_e. Hence, solution $\bar{H}_k$, induces a maximum flow through H_k of at least $\frac{OPT_{MEF}(I_{k+1})}{OPT_{MEF}(I_0)} > \frac{OPT_{MEF}(I_0)^{k+2}}{OPT_{MEF}(I_0)} = OPT_{MEF}(I_0)^{k+1}$ units, which is a contradiction, since, by induction hypothesis, we have that $OPT_{MEF}(I_k) = OPT_{MEF}(I_0)^{k+1}$. □

By Lemma 5.5. we have the following corollary.

Corollary 5.12 $OPT(H_k) = \frac{1}{OPT_{MEC}(I_k)}$.

We present the following lemma, which concludes the proof.

Lemma 5.13 *For any $\alpha > 1$, if* Max-ecmp-DAG *is NP-hard to approximate within a factor of α then*

- Min-ecmp-Congestion *is NP-hard to approximate within a factor of α in the single source-target pair setting;*
- Max-ecmp-Flow *is NP-hard to approximate within a factor of α in the single source-target pair setting.*

Proof: Suppose, by contradiction that there exists an $\alpha > 0$, such that Max-ecmp-Flow can be approximated within a factor of α, i.e., there exists a polynomial time algorithm $\mathcal{A}$ that, given an instance I of Max-ecmp-Flow, returns a solution of value $\mathcal{A}(I) \geq \frac{OPT_{MF}(I)}{\alpha}$. We can construct a α-approximation algorithm for Max-ecmp-DAG as follows. Let $I_0 = (H_0, s, t)$ be a Max-ecmp-DAG instance used in Lemma 5.6 such that its undirected copy G_0 is non-reversible. By Lemma 5.9, we know that such instance must

exists. Moreove, we have that $OPT(I_0)$ is either 1 or $\frac{2}{3}$. Further, it was proved in [FT00] that it is easy to compute an orientation of G_0 such that at least one optimal solution is compliant with it. Now, let c be an integer such that $\left(\frac{2}{3}\right)^c < \alpha$. Let $H_c = \otimes^c H_0$, denote by G_c the undirected copy of H_c, and by I_c an instance (G_c, s, t) of MAX-ECMP-FLOW. By Lemma 5.11, we have that $OPT(H_c) = OPT_{MF}(I_c)$. Now, if $OPT(H_0) = 1$, we have that $\mathcal{A}((G_c, s, t)) \geq \alpha$. Otherwise, if $OPT(H_0) = \frac{2}{3}$, since $OPT_{MF}(I_c) = OPT(H_c) = \left(\frac{2}{3}\right)^c$, we have that $\mathcal{A}(I_k) \leq \left(\frac{2}{3}\right)^c < \alpha$. Hence, $\mathcal{A}$ can be used to distinguish, in polynomial time, between MAX-ECMP-DAG instances with optimal value 1 or $\frac{2}{3}$, which is a contradiction to Theorem 5.2.

By Lemma 5.5, the same result also holds for MIN-ECMP-CONGESTION.

□

5.4 Sum of Link Costs inapproximability

As observed in [FT00], minimizing the maximum congested link may be an overly-pessimistic and incomplete measure of the network congestion state. In fact, even if just one link is congested, the value of the solution would not give any information about the state of all the other links in the network. For this reason, we now consider the well-studied MIN-SUM-COST problem. In this case each link has a cost that depends on the amount of flow that is routed through it. The goal is to minimize the sum of the costs of all the links in the network. Past works [FRT02, FT00, FT06] observed that the link cost increases progressively as the congestion approaches 1, and explodes when we go above 1. Hence, we model this behavior by an exponential function $\phi(x) = 2^x - 1$, where x is a measure of the congestion of the link. The objective goal is therefore:

$$\min \sum_{e \in E(G)} \phi\left(\frac{f_e}{c_e}\right)$$

.

We show that even in this case approximating the optimum within any constant factor is an NP-hard problem. We will again exploit our amplification technique based on the operator $\otimes$.

We first introduce and study a related problem called MIN-CONGESTED-EDGES. The goal is to minimize the number of edges that has congestion at least $\frac{3}{2}$. We then show how to leverage our $\otimes$ amplification technique in order to amplify the gap between two different classes of instances of MIN-CONGESTED-EDGES, in a similar manner as we did for the MAX-ECMP-DAG

problem in the previuos section. Finally, we relate MIN-CONGESTED-EDGES to MIN-SUM-COST by showing that the the latter is not approximable within any constant factor (Theorem 5.4).

Min-Congested-Edges problem. An instance I of this problem is a graph G, a source vertex s, and a target vertex t, exactly as in the MEC and the MEF problems.

In the reduction construction from 3-SAT used in [FT04] to prove that MIN-ECMP-CONGESTION is not approximable within a factor of $\frac{3}{2}$, a SAT formula F with n variables is transformed into an instance $I = ((G, s, t), \cdot)$ of MIN-ECMP-CONGESTION such that, if a variables assignment satisfies a clause c, then the edge e_c associated with clause c is such that $\frac{f_{e_c}}{c_{e_c}} \leq 1$, otherwise $\frac{f_{e_c}}{c_{e_c}} = \frac{3}{2}$. Since the reduction is from 3-SAT, we can use the following well-known inapproximability result (Theorem 5.1) to prove that also in a slightly modified Fortz and Thorup construction, at least a certain amount of edges must be congested (Lemma 5.2).

Theorem 5.1 *[Hås01]. For any $\epsilon > 0$, MAX-3-SAT is $(\frac{7}{8} + \epsilon)$-hard to approximate.*

We omit the proof of the following lemma, which is based on Theorem 5.1 and on a straightforward modification of the Fortz and Thorup construction.

Lemma 5.2 *There exists two constants $\alpha > 1$ and $p > 0$ such that, given a congestion threshold $C = \frac{3}{2}$, it is NP-hard to approximate* MIN-CONGESTED-EDGES *within a factor of α even if the input instance I is "non-reversible", in its optimal solution either all edges have congestion at most 1 or at least a fraction p of its edges have congestion at least C, and an orientation of I is given in input.*

In MIN-CONGESTED-EDGES, an instance $I = (G, s, t)$ is *non-reversible* if, in every optimal solution of (G, t, s) at least $p > 0$ edges have congestion at least $\frac{3}{2}$.

We now prove the following key lemma that, given an instance I, provide a lower bound on the number of edges that are "heavily" congested in $\otimes^k I$, with $k \in \mathbb{N}$.

Let $I = (G, s, t)$ be a non-reversible MIN-SUM-COST instance such that it only admits solutions where at least a fraction $p > 0$ of its edges have congestion at least C. Let H be a directed copy of G such that there exists at least an

optimal flow assignment for I that is compliant with H. We denote by G_k the undirected copy of $\otimes^k H$ and by $I_k = (G_k, s, t)$.

Lemma 5.3 *For every $k \geq 0$, every solution for (G_k, s, t) is such that at least a fraction p^{k+1} of the edges of G_k have congestion at least C^{k+1}.*

Proof: We prove it inductively on k. In the base case $k = 0$, the statement trivially holds. In the inductive step $k > 0$, by inductive hypothesis, in every solution of I^k at least $p^{k+1}|E(G_k)|$ edges have congestion at least C^{k+1}. We want to prove that in every solution of I_{k+1} at least $p^{k+2}|E(G_{k+1})|$ of the edges have congestion at least C^{k+2}. Suppose, by contradiction, that there exists an optimal solution $\bar{A}$ of I_{k+1} (i.e., a sub-DAG of $\otimes^k H$ with a source s and a sink t) such that less than $p^{k+2}|E(G_{k+1})|$ edges have congestion at least C^{k+2}. We now construct an optimal sub-DAG solution A_k of I_k from $\bar{A}$ exactly as we did in the proof of Lemma 5.8, i.e., for each vertex in common between G_k and G_{k+1}, we split traffic in the same way.

Recall that, each edge e of G_k with capacity $c_{G_k}(e) \neq \infty$ is replaced by a graph $G_e = G$ in G^{k+1}, where the capacity of each edge of G_e is multiplied by $c_{G_k}(e)$. Observe that, by definition of G, we have that at least a fraction p of the edges in $E(G)$ have a congestion of C, when one unit of traffic is routed from s to t in G. As a consequence, by definition of G_e, we know that at least a fraction p of its edges have congestion $C_e \geq Cf_e$, where f_e is the amount of flow routed from s_e to t_e in G_{k+1}. By construction of A_k, we have that edge $e \in E(G_k)$ is also traversed by a flow f_e, which means that its congestion is $\frac{C_e}{C}$. By a simple counting argument, there only exist at least $(1 - p^{k+1})|E(G_k)|$ subgraphs G_e such that for each of them less than $p|E(G)|$ edges have congestion at least C^{k+2}. This implies, that the flow assignment associated with A_k is such that at least $(1 - p^{k+1})|E(G_k)|$ edges have congestion at most $\frac{C_e}{C} < \frac{C^{k+2}}{C} = C^{k+1}$, which is a contradiction since, by inductive hypothesis, in any solution of I_k at least $p^{k+1}|E(G_k)|$ edges have congestion at least C^{k+1}.

□

We now prove that Min-Sum-Cost is inapproximable within any constant factor. We consider the two class of instances of Lemma 5.2. We then leverage our construction technique based on operator $\otimes$ on these instances. As a consequence, by Lemma 5.12 and Lemma 5.3, the gap between the optimal sum of link costs can be set arbitrary large.

Theorem 5.4 *It is NP-hard to approximate the* Min-Sum-Cost *problem within any constant factor.*

Proof: Suppose that there exists an α-approximation algorithm for a certain constant α. Let $I = (G, s, t)$ be a non-reversible instance of MIN-CONGESTED-EDGES used to prove the NP-hard-ness in Lemma 5.3, i.e., in any optimal solution of I either (i) all edges have congestion at most 1 or (ii) at least a fraction p of the edges have congestion at least $C = \left\{\frac{3}{2}\right\}$. Let H be a directed copy of G such that there exists at least an optimal flow assignment of I that is compliant with H.

We now leverage our result for MIN-CONGESTED-EDGES to get an estimate of the value of an optimal solution of the MIN-SUM-COST problem on an instance constructed based on I using our operator $\otimes$.

In case (i), by Lemma 5.12, each edge of G_k in the optimal solution of I^k has congestion at most 1. Hence, $\sum_{e \in E(G_k)} \phi\left(\frac{f_e}{c_e}\right) \leq \phi(1)|E(G_k)| = |E(G_k)|$. In case (ii), by Lemma 5.3, there exists at least a fraction p^{k+1} of the edges of G_k that have congestion at least C^{k+1}. Hence, $\sum_{e \in E(G_k)} \phi\left(\frac{f_e}{c_e}\right) \geq p^{k+1}|E(G_k)|\phi\left(\left(\frac{3}{2}\right)^{k+1}\right) = p^{k+1}|E(G_k)|2^{\left(\frac{3}{2}\right)^{k+1}-1}$. Hence, the value of an optimal solution in case (ii) is at least $2^{\left(\frac{3}{2}\right)^{k+1}-1}p^{k+1}$ times higher than the value of an optimal solution in case (i). This quantity can be made larger than α, for any $\alpha \geq 1$, by carefully selecting a certain $k > 0$. This implies that, an α-approximation algorithm for MIN-SUM-COST can be exploited to distinguish between the two class of instances, which is a contradiction because of Lemma 5.2.

□

5.5 Non-Constant (Almost Polynomial) Inapproximability Factors

Theorem 5.1 shows that both MIN-ECMP-CONGESTION and MAX-ECMP-FLOW cannot be approximated with any constant factor unless P=NP. However, if one is willing to use a slightly stronger assumption than $P \neq NP$, namely that NP is not contained in 'quasi-polynomial' time, then one can push further the technique of Lemma 5.8. Namely, assuming that not all problems in NP can be solved in "quasi-polynomial time", can lead to an even worse (i.e., higher) inapproximability factor: both MIN-ECMP-CONGESTION and MAX-ECMP-FLOW are hard to approximate within a non-constant factor that is "almost" a constant power of the size of the input instance. Again, this result is achieved via the repeated use of our gap-amplification technique (see, e.g., [BGS96] for a

similar approach).

Theorem 5.1 MIN-ECMP-CONGESTION *and* MAX-ECMP-FLOW *cannot be approximated within a factor of* $\left(\frac{3}{2}\right)^{(\log n)^{1-\epsilon}}$*, where* n *is the number of edges of the input graph, unless* NP *is in quasi polynomial time.*

To make our statement formal, we define the quasi-polynomial time family to be the set of decision problems that have an $n^{(\log n)^{\beta}}$-time solution, where n denotes the size of the instance and β is any positive constant.

Theorem 5.2 *For any* $\epsilon > 0$*,* MAX-ECMP-FLOW *is hard to approximate within factor* $\left(\frac{3}{2}\right)^{(\log n)^{1-\epsilon}}$*, where* n *is the number of edges of the input graph, unless* NP *is in quasi polynomial time.*

Note that if one assigns the value 0 to the term ϵ in the expression for the hardness-of-approximation factor above, then it becomes a constant power of n. But since the theorem requires that $\epsilon > 0$, one can interpret the hardness factor as being "almost-polynomial" in n – a power of n that slowly decreases to 0 as n grows. Before we prove Theorem 5.2 let us start with the following technical lemma.

Lemma 5.3 *Let* I *be a* MAX-ECMP-DAG *instance. Then* $|E(\otimes^k I)| \leq |E(I)|^{k+2}$.

Proof: Let $|E(I)|$ be the number of edges of I. The number of edges of $\otimes^k I$ is $|E(\otimes^k I)| = |E(I)|^{k+1} + 2(|E(I)|^k + \cdots + |E(I)|) \leq |E(I)|^{k+1} + 2|E(I)|^{k+1} \leq |E(I)|^{k+2}$, where in the last inequality we assumed that $|E(I)| \geq 2$.

□

We are now ready to prove Theorem 5.2.

Proof:[Proof of Theorem 5.2] We repeat the construction as in Lemma 5.8, except that we increase the value of k. Consider a given MAX-ECMP-DAG instance $I_0 = (G, s, t)$, whose optimal solution is either 1 or $\frac{3}{2}$. We now pick $k = \lceil (\log |E(G)|)^{\gamma} \rceil$, for some constant $\gamma > \frac{1-\epsilon}{\epsilon}$. By Lemma 5.3 we have that

$$|E(\otimes^k I)| \leq |E(I_0)|^{k+2}, \tag{5.1}$$

and thus $\otimes^k I$ can be constructed from I_0 in quasi-polynomial time. By Lemma 5.8, we have that $OPT(\otimes^k I)$ is either 1 or $\left(\frac{2}{3}\right)^{k+1}$, depending on the maximal flow in the original instance I. If we could get a polynomial time approximation for

MAX-ECMP-DAG within factor $\left(\frac{2}{3}\right)^{k+1}$ on $\otimes^k I$ (here we mean polynomial time in the size of $\otimes^k I$), we could determine whether $OPT(I)$ is 1 or $\frac{2}{3}$. Together with the construction of $\otimes^k I$ this would take quasi-polynomial time, and would be a contradiction of Theorem 5.2 if we assume that NP is not contained in quasi-polynomial time.

We thus have that it is hard to get a polynomial time approximation within $\left(\frac{2}{3}\right)^{k+1}$ for an instance the size of $\otimes^k I$. Let us now recompute the value of k as a function of the size of $\otimes^k I$. Using Equation 5.1, we have

$$|E(I_0)|^{k+2} \geq |E(\otimes^k I)|$$

$$(k+2)\log|E(I_0)| \geq \log|E(\otimes^k I)|.$$

Since $\log|E(I_0)| \leq k^{\frac{1}{\gamma}} \leq (k+2)^{\frac{1}{\gamma}}$, we have that

$$\begin{aligned}(k+2)(k+2)^{\frac{1}{\gamma}} &\geq \log|E(\otimes^k I)| \\ (k+2)^{\frac{\gamma+1}{\gamma}} &\geq \log|E(\otimes^k I)| \\ k &\geq (\log|E(\otimes^k I)|)^{\frac{\gamma}{\gamma+1}} - 2 \\ k &\geq (\log|E(\otimes^k I)|)^{1-\epsilon} - 2\end{aligned}$$

which implies that MAX-ECMP-FLOW is not approximable within a factor of

$$\left(\frac{3}{2}\right)^{k+1} \geq \left(\frac{3}{2}\right)^{(\log|E(\otimes^k I)|)^{1-\epsilon}},$$

unless NP is in quasi polynomial time, which proves the statement of the theorem.

□

5.6 TE with ECMP in Datacenter Networks

We now explore the guarantees of TE with ECMP in two specific network topologies, which have recently been studied in the context of datacenter networks: folded Clos networks and hypercubes. We prove that while in hypercubes optimal TE with ECMP remains intractable, ECMP routing easily achieves the optimal TE outcome in folded Clos networks. Our positive result for folded Clos networks implies that TE with ECMP is remarkably good when traffic consists of a large number of small (mice) flows (see Hedera [AFRR+10]),

or when traffic is split at a packet-level (instead of IP-flow-level, e.g., via Random Packet Spraying [DPHK13]), as in these contexts the splittable-flow model well-captures the network behavior. We discuss the handling of unsplittable large (elephant) flows in Section 5.7.

TE with ECMP is Optimal for Folded Clos Networks

We now present our optimality result for TE with ECMP in folded Clos networks (FCNs).

Folded Clos networks. An n-FCN is a graph whose vertices are partitioned into n sets, called stages, that is obtained via the following recursive construction:

- **A 1-FCN.** A 1-FCN consists of a single stage ("stage 1") that contains a single vertex.
- **Construction an n-FCN from an (n-1)-FCN.** Let F^{n-1} be an (n-1)-FCN. An n-FCN F^n is constructed as follows:
  - **Creating stages $1, \ldots, n-1$ of F^n:** Create, for some chosen $k > 0$, k duplicates of F^{n-1}: $F_1^{n-1}, \ldots, F_k^{n-1}$. Set stage $i = 1, \ldots, n-1$ of F^n to be the union of the i'th stages of $F_1^{n-1}, \ldots, F_k^{n-1}$. Create an edge between two vertices in stages $1, \ldots, n-1$ of F^n iff the two vertices belong to the same F_t^{n-1} and there is an edge between the two vertices in F_t^{n-1}.
  - **Creating stage n of F^n:** Create, for a chosen $r > 0$, r new vertices $v_{i,1}, \ldots, v_{i,r}$ for every vertex i in the $n-1$'th stage of F^{n-1}. Set the n'th stage of F^n to be the union $\bigcup_i \{v_{i,1} \ldots, v_{i,r}\}$. Create, for every vertex i in the $n-1$'th stage of F^{n-1} an edge between each of the k vertices in the $n-1$'th stage of F^n that correspond to vertex i and each of the vertices in $\{v_{i,1}, \ldots, v_{i,r}\}$.

Figure 5.5 shows a 3-FCN constructed by interconnecting six 2-FCNs. Past work focused on the scenario that all link capacities in an FCN are equal (as in [AFLV08, GHJ+11, YNDM07]). Our positive result below extends to the scenario that only links in the same "layer", that is, that all links that connect the same two stages in the FCN, must have equal capacity.

TE with ECMP is optimal for Clos networks even when all link weights are 1. We investigate the complexity of Min-ecmp-Congestion,

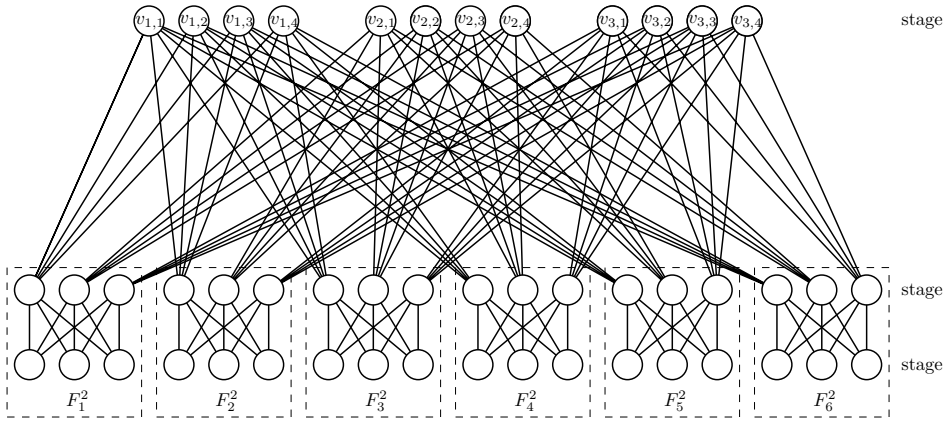

Figure 5.5: A 3-FCN constructed by interconnecting four 2-FCNs.

Min-Sum-Cost, and Max-ecmp-Flow, for FCNs. We call a demand matrix for an FCN "inter-leaf" if the source and targets of traffic are all vertices in stage 1 of the FCN (i.e., the leaves of the multi-rooted tree). Inter-leaf demand matrices capture realistic traffic patterns in datacenters, as most traffic in a datacenter flows between the top-of-rack switches at the lowest level of the datacenter topology. We present a surprising positive result: Setting all links weights to be 1 (i.e, the default in datacenters) results in the optimum traffic flow for *a*ny inter-leaf demand matrix for all three optimization objectives.

Theorem 5.1 *When all link weights in an FCN network are* 1 *ECMP routing achieves the optimum flow with respect to* Min-ecmp-Congestion*,* Min-Sum-Cost*, and* Max-ecmp-Flow*.*

We now prove this result with respect to Min-ecmp-Congestion, and for the scenario that all edge capacities are equal.

Proof: Let F be an n-FCN network such that $n \geq 2$ and all link weights are 1. An *l-sub-FCN* of F, for $1 \leq l \leq n$ is the subgraph of F that is induced by all vertices in stages $1, \ldots, l$ (i.e., the graph consisting of these vertices and edges between them only).

Now, let S be any sub-FCN of F with $l \leq n$ stages of F and let $F_1^{l-1}, \ldots, F_m^{l-1}$ be all the $(l-1)$-sub-FCNs of S that used in the recursive construction of S (see above). $\bar{V}(S)$ denotes the set of vertices in the last stage of S and $\bar{V}(F_i^{l-1})$,

with $i = 1, \ldots, m$ denotes the set of vertices in the last stage of F_i^{l-1}. The following claims easily follow from the construction of F and S.

Claim 5.2 *If $l > 1$ (S has more than one stage), then for every two vertices $v \in \bar{V}(S)$ and $u \in \bar{V}(F_i^{l-1})$ for $i = 1, \ldots, m$, (u, v) is on a shortest path from v to any vertex in the first stage of $V(F_i^{l-1})$.*

Proof: We prove by induction on l that the length of the shortest path from v to any vertex z in the first stage of F_i^{l-1} is $|l-1|$. Clearly, if $l = 2$, then there is a unique path of length 1 between v and every vertex in the first stage. If $l > 2$, then by the induction hypothesis there exists a shortest path of length $l-2$ from any vertex in $\bar{V}(F_i^{l-1})$ to any vertex z in the first stage of F_i^{l-1}. As v is directly connected to a vertex in $\bar{V}(F_i^{l-1})$, and every path to z must cross a vertex in $\bar{V}(F_i^{l-1})$, the claim follows.

□

Claim 5.3 *If $l > 1$ (S has more than one stage), then for every two vertices $v \in \bar{V}(S)$ and $u \in \bar{V}(F_i^{l-1})$ for $i = 1, \ldots, m$, (u, v) is on a shortest path from v to any vertex in the first stage of F that is not in $V(F_i^{l-1})$.*

Proof: We prove by induction on $j = n - l$ that the length of the shortest path from any $u \in \bar{V}(F_i^l)$ to any vertex z in the first stage of F that is not in $V(F_i^{l-1})$ is the same. Observe that if $j = 0$, then, by Claim 5.2, the shortest path between a vertex in $\bar{V}(S)$ and a vertex z in the first stage of F that is not in $V(F_i^{l-1})$ is $n - 1$. As every vertex $u \in \bar{V}(F_i^l)$ is directly connected to a vertex in $\bar{V}(S)$, and as all shortest paths from y must cross a vertex in $\bar{V}(S)$, the claim follows. Now, if $j > 1$, then by induction hypothesis and by Claim 5.2, from every vertex in $\bar{V}(S)$ there exists a shortest path to z (with nonnegative length). Since every vertex $u \in \bar{V}(F_i^l)$ is directly connected to a vertex in $\bar{V}(S)$ and every shortest path from u must cross a vertex in $\bar{V}(S)$, the claim again follows.

□

Let $\mathcal{F}_S$ be the set of flows such that (i) the source vertex is in S and the target vertex is not in S; or (ii) the sources vertex is in F_i^l for some $i = 1, \ldots, m$ and the target vertex is in some F_j^l for $j \neq i$.

Claim 5.4 *Each vertex in $\bar{V}(S)$ receives an equal fraction of every flow $f \in \mathcal{F}_S$.*

Proof: We prove the claim by induction on l, that is, the number of stages of S. When $l = 1$, S is simply a 1-FCN and the claim trivially follows. Now, suppose that $l > 1$. By the induction hypothesis, each vertex $v \in \bar{V}(F_i^{l-1})$ receives the same fraction of any flow $f \in \mathcal{F}_S$ whose source is contained in $V(F_i^{l-1})$. Since every vertex in $\bar{V}(F_i^{l-1})$ is connected to the same number of vertices in $\bar{V}(S)$, each vertex $v \in \bar{V}(S)$ must be (directly) connected to precisely one vertex $m_v \in \bar{V}(F_i^{l-1})$. By Claim 5.3, v is contained in a shortest path from m_v to the target vertex of f, and so each vertex in $\bar{V}(S)$ receives an equal fraction of f.

□

Let $\bar{\mathcal{F}}_S$ be the set of flows such that the target vertex is in S and the source vertex is not in S.

Claim 5.5 *Each vertex in $\bar{V}(S)$ receives an equal fraction of every flow $\bar{\mathcal{F}}_S$.*

Proof: We prove the claim by induction on the number of stages $l = n, \ldots, 1$ of F. When $l = n$, $\bar{\mathcal{F}}_S = \oslash$ and the statement holds. Otherwise, if $l < n$, let T be a $(l+1)$-sub-FCN of F that contains S as a subgraph. Consider any flow $f \in \bar{\mathcal{F}}_S$. If the source vertex of f is in (not in) T, then, by Claim 5.4 (by the induction hypothesis), each vertex in $\bar{V}(T)$ receives an equal fraction of every flow $f \in \bar{\mathcal{F}}_S$. Since each vertex in $v \in \bar{V}(T)$ is connected to exactly one vertex in $\bar{V}(S)$, each vertex $m_v \in \bar{V}(S)$ is connected to the same number of vertices in $\bar{V}(T)$, and, by Lemma 5.2, m_v is contained in a shortest path from v to the target vertex of f, we have that each vertex in $\bar{V}(S)$ receives an equal fraction of f.

□

Let E_S be the set of edges between vertices in $\bar{V}(S)$ and vertices in stage $l-1$ of F. Observe that, by the definition of FCN, the set of vertices in $\bar{V}(S)$ is a vertex-cut of F for all pairs in $\mathcal{F}_S$. Hence, each flow in $\mathcal{F}_S$ and $\bar{\mathcal{F}}_S$ must traverse at least one vertex in $\bar{V}(S)$ and through at least one edge in E_S. Let $\mathcal{F}_S^*$ be the sum of all the flows in $\mathcal{F}_S$ and in $\bar{\mathcal{F}}_S$. We have that $\frac{\mathcal{F}_S^*}{c_l|E_S|}$, where c_l is the capacity of edges between vertices in the l'th and in the $(l-1)$'th stages, is a lower bound on the amount of flow that is routed through the most loaded edge in E_S. We will now prove that when all link weights are 1, this lower bound is achieved (and the theorem follows).

Edges in E_S connect vertices in $\bar{V}(S)$ to vertices in stage $l-1$ of S. Since each vertex in $\bar{V}(S)$ is connected to the same number of vertices in stage $l-1$ of S and each vertex in stage $l-1$ of S is connected to the same number of

vertices in $\bar{V}(S)$, Claim 5.4 and Claim 5.5 imply that each edge carries an equal fraction of each flow in $\mathcal{F}^*_S$.

□

TE with ECMP is NP-hard for Hypercubes

We now investigate MIN-ECMP-CONGESTION in hypercubes. We show that, in contrast to folded Clos networks, MIN-ECMP-CONGESTION in hypercubes is NP-hard.

Hypercubes. A *k-hypercube* is a graph in which the set of vertices is $\{0,1\}^k$ and an edge between two vertices $u = (u_1, \ldots, u_k)$ and $v = (v_1, \ldots, v_n)$ exists iff the hamming distance between u and v is 1 (that is, the two vertices differ in just a single coordinate).

Optimizing TE with ECMP is intractable for hypercubes. We present the following hardness result for hypercubes.

Theorem 5.6 *Computing the optimal flow with respect to* MIN-ECMP-CONGESTION *in hypercubes is NP-hard.*

Proof: we leverage instances used in Theorem 5.3 to prove the hardness result for hypercube topologies. In particular, we consider an instance $I = (G, s, t)$ such that either $OPT_{MC}(I) = 1$ or $OPT_{MC}(I) = \frac{3}{2}$. We "embed" I into an hypercube H, with a logarithmic dimension w.r.t. to the size of I, and we carefully construct a demand matrix D for vertices in $V(H)$ in such a way that $OPT_{MC}(H, D) = 1$ iff $OPT_{MC}(I) = 1$, where $OPT_{MC}(H, D) = 1$ is the value of an optimal weight assignment for a graph H with demand matrix D.

Embedding sketch idea. Let $I = ((G, s, t), f)$ be an instance of MIN-ECMP-CONGESTION, where a vertex s of G wants to sends a flow of f units to a vertex t of G. Consider an hypercube H that contains a subgraph G' that it is a subdivision of G, i.e., a *subdivision* G' of a graph G can be obtained from G by replacing each edge of G with a single simple path. We can show that such hypercube exists and has size polynomial w.r.t. the maximum degree of a vertex of G and the size of G. We then construct a demand matrix among vertices of H. We add a flow between the endpoints of each edge $e \in E(H)$ in such a way that we saturate the capacity of e only if e is not in $E(G')$. In order to enforce capacity constraints of edges of G to paths in H, for each path p of G' that corresponds to an edge e of G, for each edge e' of p, we assign a certain flow

between the endpoints of e' that limits the capacity of that path. Namely, the higher the value of the capacity of e, the lower the size of the flow that we assign between the endpoints of e'. These flows are used to force a flow from s to t to flow exactly through G'. We then refine our mapping by removing "chords" from the subgraph induced by vertices of G'. Since G' is a subdivision of G and the available capacity of each edge of G' is properly scaled by these extra-flows, we can prove that if $OPT_{MC}(I) = 1$, then $OPT_{MC}(H, D) = 1$. Otherwise, if $OPT_{MC}(I) > 1$, then $OPT_{MC}(H, D) > 1$. Since MIN-ECMP-CONGESTION is NP-hard even in the case G has degree at most 3 and $OPT_{MC}(I)$ is either 1 or $\frac{3}{2}$ [FT04], then also MIN-ECMP-CONGESTION is NP-hard even if we restrict our attention to hypercubes.

We now show how to find a subgraph G' of an hypercube such that G' is a subdivision of G.

Embedding a graph instance into an hypercube. Consider an instance (G, D) of MIN-ECMP-CONGESTION, where D contains only a flow demand f from s to t. We first map G into a k-hypercube H, with $k > 0$. Let ϕ be an injective function that maps each vertex v of G to a vertex $\phi(v)$ of H, each edge $e = (v, u)$ of I to a simple path $\phi(e)$ of H from $\phi(v)$ to $\phi(u)$. Let G' be the subgraph of H such that $e \in E(G')$ iff there exists an edge $e' \in E(G)$ such that $\phi(e')$ traverses e. We say that ϕ is an *embedding* of G into H if G' is a subdivision of G. In other words, for each pair of edges e_1 and e_2 of G, paths $\phi(e_1)$ and $\phi(e_2)$ are internal-vertex-disjoint.

We construct H from G with the following recursive procedure. Let $M = \max\{|V(G)|2^{\Delta(G)-1}, 2|E(G)|\}$, where $\Delta(G)$ is the degree of the vertex with the maximum degree, and $d = 2\lceil \lg_2 M \rceil$. We first construct a d-hypercube H using the following notation to denote its vertices. H contains 2^d vertices $v_{(x,y)}$, with $0 \leq x \leq 2^{\frac{d}{2}} - 1$ and $0 \leq y \leq 2^{\frac{d}{2}} - 1$. There exists an edge between two vertices v_{xy} and v_{wz} iff there exists a $n \geq 0$ such that either $(x = w) \wedge (|z - y| = 2^n) \wedge \left(\lfloor \frac{y}{2^{n+1}} \rfloor = \lfloor \frac{z}{2^{n+1}} \rfloor\right)$, or $(y = z) \wedge (|x - w| = 2^n) \wedge \left(\lfloor \frac{x}{2^{n+1}} \rfloor = \lfloor \frac{w}{2^{n+1}} \rfloor\right)$.

We now create a mapping ϕ from G to H. Let $u_0, \ldots, u_n$ be all the vertices of G. Let $\phi(u_i) = v_{(0,2^{\Delta(G)-1}i)}$. Let A be the set of edges of G that has been mapped to a path of H. Initially, $A = \emptyset$. Each vertex v order its incident edges into a sequence E_v. For each edge $e = (a, b) \in E(G)$, we compute a path p_i from $\phi(a) = v_{(0,y)}$ to $\phi(b) = v_{(0,z)}$, where $y = \phi(a)$ and $z = \phi(b)$, as a concatenation of 5 paths p_1, p_2, p_3, p_4 and p_5. Suppose e is the i-th (j-th) edge in E_a (E_b). If $i \neq 1$, $p_1 = (v_{(0,y)}, v_{(0,y+2^{i-1})})$, otherwise $p_1 = (v_{(0,y)})$. If $j \neq 1$, $p_5 = (v_{(0,z)}, v_{(0,z+2^{j-1})})$, otherwise $p_5 = (v_{(0,z)})$. p_2 is a shortest path from $v_{(0,y+2^{i-1})}$ to $v_{(2|A|+1,y+2^{i-1})}$. p_4 is a shortest path from $v_{(0,z+2^{j-1})}$ to

$v_{(2|A|+1,z+2^{j-1})}$. p_3 is a shortest path from $v_{(2|A|+1,y+2^{i-1})}$ to $v_{(2|A|+1,z+2^{j-1})}$. Add e into A.

See an example of embedding a clique K_4 of size 4 into an hypercube H of dimension $d = 2\lceil \lg_2 M \rceil = \lceil \lg_2(\max\{|V(G)|2^{\Delta(G)-1}, 2|E(G)|\})\rceil = 2\lceil \lg_2(\max\{4 \cdot 2^{3-1}, 12\})\rceil = 2\lceil \lg_2(12)\rceil = 8$ in Fig. 5.6. Each vertex $v_{(x,y)}$ of H is represented as a circle in column x and row y. Edges of the hypercube are not depicted. In order to better understand where an edge between two vertices of H exists, we drew several dashed lines, each of them with a number beside it. An edge of the hypercube is such that, if it intersects any dashed line, then its two endpoints have a difference in one of their coordinate that is exactly equal to the highest number associated to any of the intersected dashed lines. Let u_0, u_1, u_2, u_3 be all the vertices of K_4. We have that $\phi(u_0) = v_{(0,0)}, \phi(u_1) = v_{(0,4)}, \phi(u_2) = v_{(0,8)}$, and $\phi(u_3) = v_{(0,12)}$. These vertices are colored gray in the picture. Let (u_0, u_1) be the first edge of K_4 to be analyzed and let (u_0, u_1) be the first edge in both E_{u_0} and E_{u_1}. We have that path $p_1 = (v_{(0,0)})$, path $p_2 = (v_{(0,0)}, v_{(1,0)})$, path $p_3 = (v_{(1,0)}, v_{(1,4)})$, path $p_4 = (v_{(1,4)}, v_{(0,4)})$, and $p_5 = (v_{(0,4)})$. Now, consider edge (u_0, u_2) and suppose that it is the second (first) edge in E_{u_0} (E_{u_2}). We have that path $p_1 = (v_{(0,0)}, v_{(0,1)})$, path $p_2 = (v_{(0,1)}, v_{(2,1)}, v_{(3,1)})$, path $p_3 = (v_{(3,1)}, v_{(3,9)}, v_{(3,8)})$, path $p_4 = (v_{(3,8)}, v_{(2,8)}, v_{(0,8)})$, and $p_5 = (v_{(0,8)})$. Intuitively, for each edge (x, y), path p_1 (p_5) is used to connect $\phi(x)$ ($\phi(y)$) to a vertex z_x (z_y) in the first column in the first "available" row below $\phi(x)$ ($\phi(y)$). Path p_1 and p_5 may be empty in the case (x, y) is the first edge in E_x and E_y, respectively. Then, path p_2 (p_4) is used to connect z_x (z_y) to a vertex w_x (w_y) of a column with index $2|A| + 1$. Observe that, since p_2 and p_4 are shortest paths, they never change row. Also, observe that each of these paths has only one vertex that lies on an even column, namely on column $2|A| + 1$. Finally, path p_3 is used to interconnect these two vertices w_x and w_y. Also in this case, path p_3 never leaves column $2|A| + 1$. Because of these properties, it is easy to see that for each pair of edges e_1 and e_2 of G, paths $\phi(e_1)$ and $\phi(e_2)$ are internally vertex-disjoint.

Assigning flow demands in H. We construct set D from $((G, s, t), f)$, H, and ϕ. For each edge $e \in E(G)$, for each edge (x, y) of $\phi(e)$, add flow $((x, y), \frac{2}{5}\left(\frac{c_{max} - c_G((x,y))}{4c_{max}} + 1\right)$ and $((y, x), \frac{2}{5}\left(\frac{c_{max} - c_G((x,y))}{4c_{max}} + 1\right)$ into D. For each edge $e' \in H$ that has no flow assigned, add flows $((x, y), \frac{1}{2})$ and $((y, x), \frac{1}{2})$ into D. Finally, add a flow $((\phi(s), \phi(t)), \frac{1}{5c_{max}})$.

Removing chords from G'. Consider the subgraph of H induced by vertices of G', where $V(G') = \{v \in V(H') | \exists e \in E(G)$ such that $\phi(e)$ passes through $v\}$

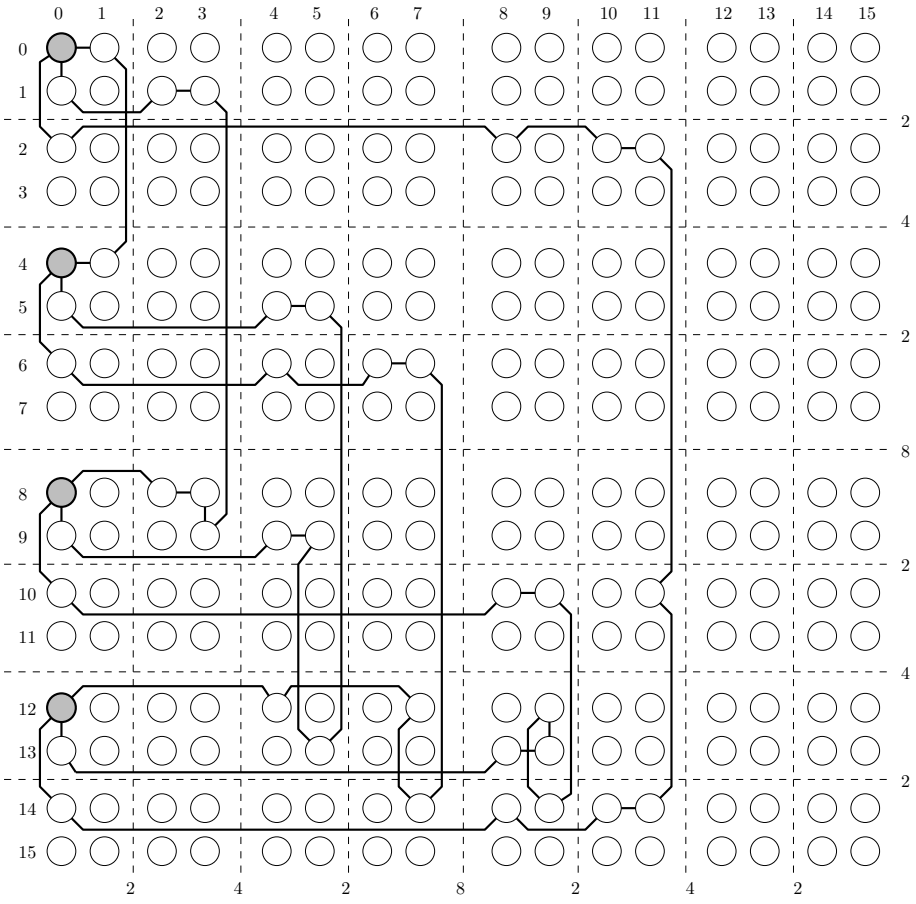

Figure 5.6: A clique with 4 vertices embedded in a 8-hypercube. Vertices of the clique are mapped to vertices $v_{0,0}, v_{0,4}, v_{0,8}$ and $v_{0,12}$, depicted as gray vertices. Dashed lines are depicted in order to ease the readability of this figure. Let l be the highest number of a dashed line intersected by an edge $(v_{x,y}, x_{x,z})$ $((v_{x,y}, x_{w,y}))$. Then, it must hold that $|y - z| = l$ $(|x - w| = l)$.

.

and $E(G') = \{e' \in E(H') | \exists e \in E(G)$ such that $\phi(e)$ traverses $e\}\}$. By the above construction, G' may contain edges that are not in $E(G')$. We call these edge "chords". We now show a procedure that, given a mapping ϕ of a graph G into a k-hypercube H, produces a new mapping ϕ' from G' to a $2k$-hypercube H' such that the subgraph of H' induced by vertices in $\{v \in V(H') | \exists e \in E(G')$ such that $\phi'(e)$ passes through $v\}$ is chordless. The construction works as follow. Map each vertex $x = (x_0, \ldots, x_k)$ of H, where $x \in V(G')$, to vertex $\phi'(x) = (q_x, q_x) = (x_0, \ldots, x_k, x_0, \ldots, x_k)$ of H. Consider each edge $e = (x, y) = ((x_0, \ldots, x_k), (y_0, \ldots, y_k))$ of H, where $(x, y) \in E(G')$. Since (x, y) is an edge of an hypercube, x and y differ in exactly one coordinate i. Map (x, y) to a path $\phi'(e) = (\phi'(x), \bar{x}^i, \phi'(y))$, where $\bar{x} = (q_x^i, q_x) = (x_0, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_k, x_0, \ldots, x_k)$, where q_x^i is obtained by flipping the i-th coordinate of q_x. Let G'' be a subgraph of H', such that $V(G'') = \{v \in V(H') | \exists u \in V(G')$ s.t. $\phi'(u) = v\}$ and $E(G') = \{e \in E(H') | \exists e' \in E(G')$ s.t. $\phi'(e')$ traverses $e\}$.

Lemma 5.7 *G'' is chordless.*

Proof: Suppose, by contradiction, that G'' is not chordless. Then, there exists at least a pair of vertices x and y of G'' that are not adjacent in G'' and are adjacent in H'. Let i be the coordinate in which x and y differs by one element. We have two cases. If there exist two vertices a and b of G' such that $\phi'(a) = x$ and $\phi'(b)$, then, by construction of G'', since a and b differs in one coordinate, we have that x and y must differ in two coordinates. This is a contradiction since (x, y) cannot be a chord. Otherwise, if such two vertices does not exists, w.l.o.g., let x be a vertex such that there does not exist a vertex a of G' with $\phi'(a) = x$. Observe that, by construction of ϕ', each vertex $z = (q_z, q_z)$ of $\phi'(G)$ has coordinate either (q_z, q_z) or (q_z^j, q_z), with $j = 1, \ldots, k$. Hence, x has coordinates (q_x^j, q_x), with $j = 1, \ldots, k$. x and, since y is a neighbor of x and it is a vertex of G'', it must have coordinate either (q_x, q_x) or (q_x^i, q_x^i). This leads to a contradiction since, by construction of G'', both $((q_x, q_x), (q_x^i, q_x))$ and $((q_x^i, q_x), (q_x^i, q_x^i))$ are edges of G''.

□

Assigning flow demands in H'. We construct set D' from (H, D) and ϕ' as follows. For each flow demand D_{xy} in D, with $x \neq s$ and $y \neq t$, for each edge $e' = (x', y')$ traversed by $\phi'((x, y))$, add a flow demand $D_{x'y'} = D_{xy}$ into D'. For each edge $e' = (x', y') \in E(H')$ such that there does not exists an edge e of G' such that $\phi'(e)$ traverses e', add flows $((x', y'), \frac{1}{2})$ and $((y', x'), \frac{1}{2})$ into D. Finally, add a flow demand $D'\phi'(s), \phi'(t) = D_{st}$ into D'.

Proving optimal solution for (H', D').

Lemma 5.8 *If $OPT_{MC}(I) = 1$, then $OPT_{MC}(H', D') = 1$.*

Proof: We first compute a weight assignment of (H', D') from a weight assignment of (I, f) that has congestion at most 1. We first map each flow (x, y) in D' to an (x, y)-DAG $\sigma((x, y))$. Then, we compute weight links from this set of DAG. Suppose $OPT_{MC}(I) = 1$. By construction of D', each flow $((x, y), 1) \in D'$, with $x \neq \phi'(\phi(s))$ and $y \neq \phi'(\phi(t))$, is such that $(x, y) \in E(H')$. Let $\sigma((x, y)) = A_{xy}$, where A_{xy} is a (x, y)-DAG that consists of a single edge (x, y). Consider an optimal (s, t)-DAG A_I of (I, f). We construct an $(\phi'(\phi(s)), \phi'(\phi(t)))$-DAG A_{st} from A_I. For each edge $e \in E(A_I)$, add directed path $\phi'(\phi(e))$ into $E(A_{st})$. Finally, let $\sigma((\phi'(\phi(s)), \phi'(\phi(t)))) = A_{st}$. By construction of (H', D'), it is trivial to see that this solution has congestion equal to 1. Consider an edge $(x, y) \notin A_{st}$. By construction of (H', D') and σ, only two saturating flows $((x, y), \frac{1}{2})$ and $((y, x), \frac{1}{2})$ are routed through (x, y), which implies that (x, y) has congestion equal to 1. Consider now an edge $(x, y) \in E(A_{st})$. By construction of (H', D') and σ, we have that (x, y) is traversed by a fraction of the flow from $\phi'(\phi(s))$ to $\phi'(\phi(t))$ and, a saturating flow $\left((y, x), \frac{2}{5}\left(\frac{c_{max} - c_G((u,v))}{4c_{max}} + 1\right)\right)$, where (u, v) is the edge of A_I and edge e' of $\phi((u, v))$ is such that (x, y) is an edge of $\phi'(e')$, and a saturating flow $\left((y, x), \frac{2}{5}\left(\frac{c_{max} - c_G((u,v))}{4c_{max}} + 1\right)\right)$. Observe that, if a fraction q of the unit flow from s to t is routed through (u, v) in G, then, by construction of A_{st}, also a fraction q of the flow from $\phi'(\phi(s))$ to $\phi'(\phi(t))$ is routed through (x, y). Hence, we have that the amount of flow routed through (x, y) is

$$2 \cdot \frac{2}{5}\left(\frac{c_{max} - c_G((u,v))}{4c_{max}} + 1\right) + q\frac{1}{5c_{max}} =$$

$$= \frac{4}{5}\left(\frac{c_{max} - c_G((u,v))}{4c_{max}} + 1\right) + q\frac{1}{5c_{max}} =$$

$$= \frac{1}{5}\left(\frac{c_{max} - c_G((u,v)) + 4c_{max} + q}{c_{max}}\right) =$$

$$= \frac{1}{5}\left(\frac{5c_{max} - c_G((u,v)) + q}{c_{max}}\right) =$$

$$= 1 - \frac{c_G((u,v)) - q}{c_{max}} \leq 1, \text{ since } q \leq c_G((u,v)).$$

We now show how to set link weights in H' in order to obtain a flow assignment where flow is routed according to σ. For each saturating flow (x, y) of H' such that $(x, y) \notin E(G'')$, we set a very large weight $W >> 1$ to edge (x, y). Since G'' is chordless, each of these flow is routed through (x, y). Let $\bar{\sigma} = \sigma(\phi'(\phi(s)), \phi'(\phi(t)))$. We set link weights in $\bar{\sigma}$ using Lemma 5.7. Hence, flow demand $(\phi'(\phi(s)), \phi'(\phi(t)))$ is routed through $\bar{\sigma}$ and each saturating flow (x, y) of H' such that $(x, y) \in E(G'')$ is routed exactly through (x, y). This concludes the proof of the lemma.

□

Lemma 5.9 *If $OPT_{MC}(I) > 1$ and G has maximum degree 3, then $OPT_{MC}(H', D') > 1$.*

Proof: Suppose, by contradiction, that there exists an assignment of the link weights such that $mc^*(H', D') \leq 1$. Among these optimal assignments, consider the one that has the higher number of saturating flows routed through the edge that interconnects their source and target vertices. For each flow $((x, y), \cdot) \in D'$, let A_{xy} be the (x, y)-DAG where the flow is routed through. We show that for each saturating flow $((x, y), \frac{1}{2})$, where $(x, y) \in E(H')$, we have that A_{xy} consists of a single edge (x, y). Suppose, by contradiction, that it is not true. Observe that, if A_{xy} contains an edge (x', y'), then $A_{x'y'} \subset A_{xy}$. It implies that there must exists a saturating flow $f^* = ((x, y), \cdot)$ such that A_{xy} consists of at least a directed path p different from (x, y). Observe that A_{yx} contains an edge (u, v) iff A_{xy} contains an edge (v, u). Let $A*$ be such (x, y)-DAG of a saturating flow $((x, y), \cdot)$ such that it does not consist of a single edge (x, y). Consider all the n_x edges adjacent to x. Since G has maximum degree 3, we have that at least $n_x - 4$ (one edge connects x to y) of these edges incident to x are edges whose capacity is saturated by saturating flows. The remaining edges different from (x, y) (at most 3) have congestion at least $\frac{4}{5}$. If x splits $((x, y), \cdot)$ among 5 of its neighbors, then it is sending a flow with size greater than 0 through an edge that already has congestion 1. This is a contradiction, since we assumed that $mc^*((H', D')) \leq 1$. Otherwise, x sends at least a fraction $\frac{1}{4}$ of flow $((x, y), \frac{1}{2})$ through at least an edge that already has congestion at least $\frac{4}{5}$, which is a contradiction since we assumed that $mc^*((H', D')) \leq 1$.

Hence, for each saturating flow $((x, y), \frac{1}{2})$, where $(x, y) \in E(H')$, we have that A_{xy} consists of a single edge (x, y). Consider now $A = A_{\phi'(\phi(s))\phi'(\phi(t))}$. Observe that A contains an edge (x, y) only if there exists an edge $e \in E(G)$ such that there exists an edge e' of H traversed by $\phi((x, y))$ and (x, y) is traversed by $\phi'(e')$. We now compute an (s, t)-DAG A^* of (I, f) such that

$OPT_{MC}(I) \leq 1$, which is a contradiction. For each edge $e \in E(G)$ such that for every $e' \in E(G')$ traversed by path $\phi(e)$, $\phi'(e')$ is contained in A, add e into $E(A^*)$. Observe that, since ϕ and ϕ' defines a subdivision of G, we have that if a fraction q of the flow from $\phi'(\phi(s))$ to $\phi'(\phi(t))$ is routed through $\phi'(e')$, then the same fraction q of the flow from s to t is routed through e, where $\phi(e)$ traverses e'. Hence, since congestion of $\phi'(e')$ is at most 1, it implies that

$$2 \cdot \frac{2}{5} \left(\frac{c_{max} - c_G((u,v))}{4c_{max}} + 1 \right) + q \frac{1}{5c_{max}} \leq 1$$

$$1 - \frac{c_G((u,v)) - q}{c_{max}} \leq 1$$

$$c_G((u,v)) \geq q,$$

which means that each edge has congestion less than 1, i.e., $OPT_{MC}(I) \leq 1$, a contradiction. Hence, the statement of the theorem holds.

□

The theorem easily follows by Lemma 5.8 and Lemma 5.9. Observe also that since the embedded instance I has degree at most 3, hypercube H' has dimension polynomial w.r.t. the size of I.

□

5.7 Routing Elephants in Datacenter Networks

A key shortcoming of ECMP is that large, long-lived ("elephant") flows traversing a router can be mapped to the same output port. Such "collisions" can cause load imbalances across multiple paths and network bottlenecks, resulting in substantial bandwidth losses. To remedy this situation, recent studies, e.g., Hedera [AFRR+10] and DevoFlow [CMT+11], call for dynamically scheduling elephant flows in folded Clos datacenter networks so as to minimize traffic imbalances (while still routing small, "mice" flows via link-state routing and ECMP). We therefore next focus on the so called "unsplittable-flow model".

MIN-CONGESTION-UNSPLITTABLE-FLOW (MCUF). We study the MIN-CONGESTION-UNSPLITTABLE-FLOW (MCUF) objective: The input is a capacitated graph $G = (V, E, c)$ and a set $\bar{D}$ of "flow demands" of the form (s, t, γ) for $s, t \in V$ and $\gamma > 0$, where a single source-target pair (s, t) can appear in more than one flow demand. The goal is to select, for every flow demand (s, t, γ), a *s*ingle

shortest path from s to t, such that the maximum load, i.e., $\frac{f_e}{c_e}$, is minimized (as in MIN-ECMP-CONGESTION, see Section 5.2 for formal definitions of flow assignments and load). We aim to understand how well unsplittable flows can be routed in datacenter network topologies and, specifically, in FCNs.

MCUF **cannot be approximated within a factor better than 2 even in 2-FCNs.** We show in the following theorem that approximating MCUF within a factor better than 2 is NP-hard even in a 2-FCN.

Theorem 5.1 *Approximating* MCUF *within a factor of* $2 - \epsilon$ *is NP-hard for 2-FCNs for any constant* $\epsilon > 0$.

Proof: We prove it by a polynomial time reduction from the 3-EDGE-COLORING problem. The input of 3-EDGE-COLORING consists of an unoriented graph G and a set of 3 colors $\{c_1, c_2, c_3\}$. Each edge can be colored with any of these colors. An *edge-coloring* of G assigns a color to each edge of G. An edge-coloring is *valid* if, for each vertex $v \in V(G)$, no pair of edges incident to v have the same color. If there exists a valid edge-coloring of G, G is 3-*colorable*. In 3-EDGE-COLORING, it is asked to determine whether G is 3-colorable. The following lemma is a well-known result about edge-coloring problems.

Lemma 5.2 *[Hol81] It is NP-hard to determine whether a graph* G *with maximum degree* 3 *is* 3*-colorable.*

We now use this result to prove the following theorem.

In this proof, each flow demand that we will have to route has size 1. Therefore, we avoid to specify the size of each flow demand and, consequently, the set of flow demands D is modeled simply as a set of pairs of vertices. Let G be an input graph of 3-EDGE-COLORING. We construct an instance $I = (F, D)$ of MCUF, where F is a 2-FCN and D is a set of flow demands constructed as follows. Add three vertices y_1, y_2, and y_3 in the last stage of F. For each vertex $v \in V(G)$, add a vertex u_v in the first stage of F. Connect each vertex in the first stage to each vertex in the last stage, i.e., F is a complete bipartite graph. Each edge of F has capacity 1. We now map each edge of G to a flow demand in D. For each edge $(x, y) \in E(G)$, add a flow demand (u_x, u_y) into D. It is easy to verify that this construction reduction can be done in polynomial time w.r.t. the size of G. Observe that each flow demand must traverse exactly one vertex in the last stage of F in order to be routed between its source and target vertices.

We now prove that G is 3-colorable iff there exists a routing R of flow demands in D such that $mc(I, R) = 1$. Suppose that G is 3-colorable and let γ be a valid edge-coloring of G that assigns a color $\gamma(e)$ to each edge of G. We construct a routing solution R as follows. We first map colors c_1, c_2, and c_3 to vertices y_1, y_2, and y_3, respectively. Then, for each flow demand (u_x, u_y), if $\gamma((x, y)) = c_i$, with $i = 1, 2, 3$, we route (u_x, u_y) through y_i. Suppose, by contradiction, that there exists an edge of F that has congestion 2. This implies that at least two flow demands d_1 and d_2 that share at least one endpoint v, are routed through the same vertex y_i, with $i = 1, 2, 3$. By construction of R, this implies that d_1 and d_2 are both incident to v in G and γ colors both d_1 and d_2 with the same color, which is a contradiction since γ is a valid edge-coloring. Suppose now that all flow demands in D can be routed with congestion at most 1. We construct a edge-coloring γ of G as follows. For each flow demand (u_x, u_y), let y_i be the vertex in the last stage of F, with $i = 1, 2, 3$, that is traversed by (u_x, u_y). Let $\gamma((x, y)) = c_i$. Suppose, by contradiction, that γ is not a valid edge-coloring of G. Let (x, y) and (x, z) be two edges of G such that $\gamma((x, y)) = \gamma((x, z))$. By construction of γ, flow demands (u_x, u_y) and (u_x, u_z) are both routed to the same vertex y_i, with $i = 1, 2, 3$. This implies that there are two flows routed through edge (u_x, y_i), which is a contradiction.

The above reduction shows that, if G is 3-colorable, F has congestion at most 1, otherwise, if G is not 3-colorable, F has congestion at least 2 since at least two flows are routed through the same edge. Hence, by Lemma 5.2, the hardness of the problem is proved.

□

A 5-approximation algorithm for 3-FCNs. We now consider 3-FCNs, which are of much interest in the datacenters context. [AFLV08] and VL2 [GHJ+11] advocate 3-FCNs as a datacenter topology, and Hedera [AFRR+10] and DevoFlow [CMT+11] study the routing of elephant flows in such networks. We present a natural, greedy algorithm for MCUF, called EQUILIBRIUM-ALGO:

- Start with an arbitrary assignment of a single shortest path for every source-target pair (s, t).
- While there exists a source-destination pair (s, t) such that rerouting the flow from s to t to a different path can either (1) result in a lower maximum load or (2) lower the number of links in the network with the highest load, reroute the flow from s to t accordingly. We call this a "reroute operation".

We show that EQUILIBRIUM-ALGO has provable guarantees. Recall that D is a set of flow demands.

Theorem 5.3 *After $|\bar{D}|$ reroute operations,* EQUILIBRIUM-ALGO *approximates* MCUF *in 3-FCNs within a factor of* 5.

Theorem 5.1 establishes that even in 2-FCNs (and hence also in 3-FCNs) no approximation ratio better than 2 is achievable. We leave open the question of closing the gap between the lower bound of 2 and upper bound of 5 (see Section 5.9). We do show that the analysis of EQUILIBRIUM-ALGO is tight for equal-size flows. We point out that the key idea behind EQUILIBRIUM-ALGO (rerouting flows to least loaded paths until reaching an equilibrium) resembles the simulated annealing procedure in Hedera [AFRR$^+$10] and can be regarded as a first step towards analyzing the provable guarantees of this family of heuristics.

Proof for 5-approximation. We introduce the following notation. Consider a 3-FCN F that contains k_r 2-FCN, each with k_b vertices in its first stage and k_m vertices in its last stage. Every i'th vertex in the last stage of a 2-FCN is connected to the same k_t vertices in the last stage of F. Hence, there are $k_t k_m$ vertices in the last stage of F. We denote by b_i^j (m_i^j) the i'th vertex in the first (second) stage of the j'th FCN. Each vertex m_i^j is connected to vertices $t_1^j, \ldots, t_{k_t}^j$ in the last stage of F. See Figure 5.7. Consider a flow assignment computed by EQUILIBRIUM-ALGO. A flow demand $d \in \bar{D}$ from vertex s to vertex t of size γ_d is denoted by $((x, y), \gamma_d)$. For each demand $d \in \bar{D}$, let p_d be the simple path along which d is routed and $c(p_d)$ be the value of the most congested link of p_d.

Lemma 5.4 *Let $d \in \bar{D}$ be a flow demand such that $c(p_d) \geq 5 \cdot OPT$. There exists a path p' between s and t such that $c(p') \leq 5 \cdot OPT - \gamma_d$.*

Proof: Suppose, by contradiction, that such a path p' does not exist. Let $s = b_i^j$ and $t = b_g^l$, with $i, g \in [k_b]$ and $j, l \in [k_r]$, where $[n] = 1, \ldots, n$. Observe that f_d is a lower bound for the optimal solution, i.e. $OPT \geq f_d$. It implies that $c(p_d) \geq 5 \cdot OPT \geq 5 f_d$. Let n_b be the number of edges incident to b_i^j plus the number of edges incident to b_g^l that have congestion at least $5 \cdot OPT$. Let n'_b be the number of edges incident to b_i^j plus the number of edges incident to b_g^l that have congestion at least $5 \cdot OPT - f_d$ and at most $5 \cdot OPT$. We denote by $\mathcal{F}_v$ the amount of flow demands that have v as a source or target vertex, i.e. $\mathcal{F}_v = \sum_{d'=((v,\cdot),\cdot) \in D} f_{d'} + \sum_{d'=((\cdot,v),\cdot) \in D} f_{d'}$. Hence we have that,

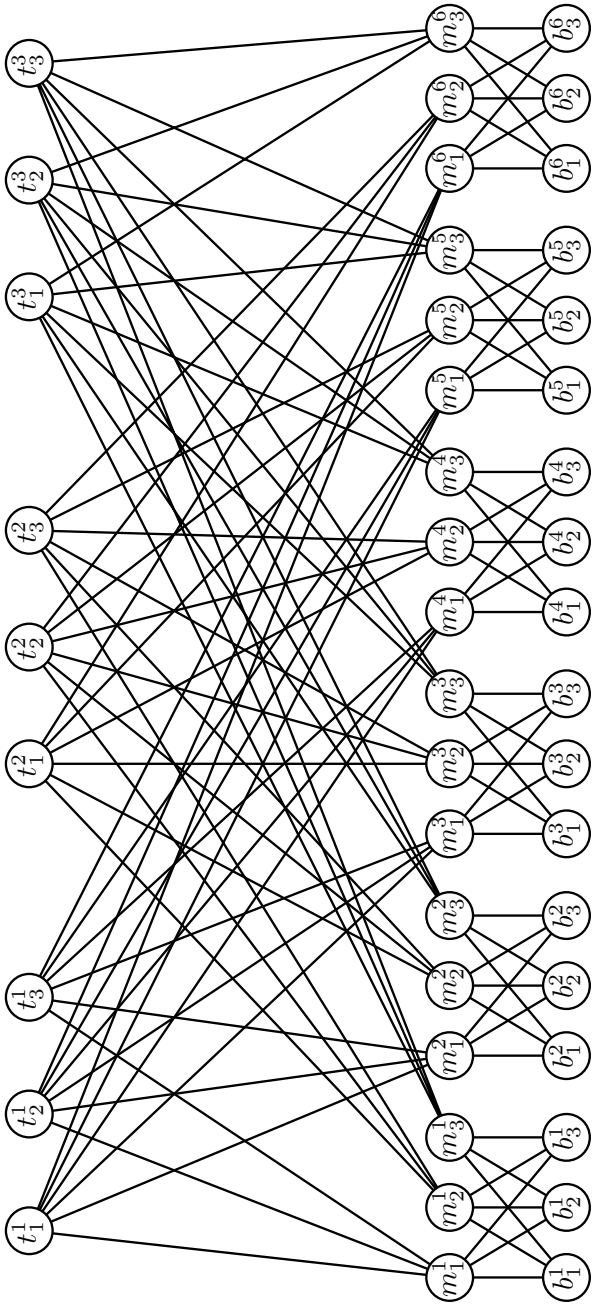

Figure 5.7: A 3-FCN with $k_r = 6$, $k_b = 3$, $k_m = 3$, and $k_t = 3$.

$$\mathcal{F}_{b_i^j} + \mathcal{F}_{b_g^l} \geq n_b(5 \cdot OPT) + n'_b(5 \cdot OPT - f_d) \geq$$

$$\geq 5n_bOPT + n'_b(5 \cdot OPT - OPT) = 5n_bOPT + 4n'_bOPT$$

Let $\mathcal{F}_* = \max\{f^{b_i^j}, f^{b_g^l}\}$. We have that

$$2\mathcal{F}_* \geq 5n_bOPT + 4n'_bOPT \tag{5.2}$$

Consider now the following obvious lower bound for OPT

$$OPT \geq \frac{\mathcal{F}_{b_i^j}}{k_m}, OPT \geq \frac{\mathcal{F}_{b_g^l}}{k_m} \Rightarrow OPT \geq \frac{\mathcal{F}_*}{k_m} \tag{5.3}$$

In the first lower bound, we say that the total amount of flow originated from or directed to b_i^j must necessarily be split among its k_m edges that connect it to the vertices in the second stage. The same bound holds for b_g^l.

Combining (5.2) and (5.3), we obtain

$$k_mOPT \geq \frac{5n_bOPT + 4n'_bOPT}{2}$$

$$k_m \geq \frac{5n_b + 4n'_b}{2}$$

$$\frac{k_m}{2} \geq \frac{5}{4}n_b + n'_b \tag{5.4}$$

Let H be the set of indices h such that both (b_i^j, m_h^j) and (b_g^l, m_h^l) have congestion lower than or equal to $5 \cdot OPT - f_d$. By Equation (5.4), we have that $|H| \geq k_m - n_b - n'_b \geq k_m - (\frac{5}{4}n_b + n'_b) \geq \frac{k_m}{2}$. Observe that, if $j = l$, i.e., the source and target vertex are both in the j-th 2-FCN, hence d can be routed through any path (b_i^j, m_h^j, b_g^j), with $h \in H$, that has congestion less than $5 \cdot OPT - f_d$. This is a contradiction, since we assumed that such path does not exists. Hence, $j \neq l$. In this case, let n_t be the number of edges incident to any vertex t_x^h, with $h \in H$ and $1 \leq x \leq k_t$ and congestion at least $5 \cdot OPT$, and n'_t be the number of edges incident to any vertex t_x^h, with $h \in H$ and $1 \leq x \leq k_t$ and congestion between $5 \cdot OPT - f_d$ and $5 \cdot OPT$. Observe that each path (m_h^j, t_x^h, m_h^l) must have congestion at least $5 \cdot OPT - f_d$, otherwise d can be routed through $(b_i^j, m_h^j, t_x^h, m_h^l, b_g^l)$, which is a contradiction since we assumed that such path does not exists. Hence,

$$n_t + n'_t \geq |H| k_t \geq (k_m - n_b - n'_b) k_t \tag{5.5}$$

and we have that

$$\sum_{i \in [k_b]} \mathcal{F}_{b_i^j} + \sum_{i \in [k_b]} \mathcal{F}_{b_i^l} \geq n_t (5 \cdot OPT) + n'_t (5 \cdot OPT - f_d) \geq$$

$$\geq 5 n_t OPT + 4 n'_t OPT = OPT(5 n_t + 4 n'_t)$$

where $\sum_{i \in [k_b]} \mathcal{F}_{b_i^j}$ ($\sum_{i \in [k_b]} \mathcal{F}_{b_i^l}$) is the sum of the flows originated from or directed to a vertex in the j-th (l-th) FCN. Let $\mathcal{F}_H = max\{\sum_{i \in [k_b]} \mathcal{F}_{b_i^j}, \sum_{i \in [k_b]} \mathcal{F}_{b_i^l}\}$. We have that

$$2\mathcal{F}_H \geq OPT(5 n_t + 4 n'_t) \tag{5.6}$$

Consider now the following obvious lower bounds for OPT

$$OPT \geq \frac{\sum_{i \in [k_b]} \mathcal{F}_{b_i^j}}{k_t k_m}, OPT \geq \frac{\sum_{i \in [k_b]} \mathcal{F}_{b_i^l}}{k_t k_m} \Rightarrow$$

$$\Rightarrow OPT \geq \frac{\mathcal{F}_H}{k_t k_m} \tag{5.7}$$

After $|\bar{D}|$ reroute operations, EQUILIBRIUM-ALGO approximates MCUF in 3-FCNs within a factor of 5.

In the first lower bound, we say that the total amount of flow originated from or directed to vertices in the j-th FCN must necessarily be split among its $k_m k_t$ edges that connect it to the last stage vertices. The same bound holds for the l-th FCN. Combining (5.6), and (5.7), we obtain

$$k_t k_m OPT \geq \frac{OPT(5 n_t + 4 n'_t)}{2}$$

Using (5.5) and (5.4), we have that

$$2 k_t k_m \geq 5 n_t + 4 n'_t = 4(n_t + n'_t) + n_t \geq 4 k_t (k_m - n_b - n'_b) + n_t =$$

$$= 4 k_t \left(k_m - \frac{5}{4} n_b - n'_b + \frac{1}{4} n_b \right) + n_t \geq 4 k_t \left(\frac{k_m}{2} + \frac{n_b}{4} \right) + n_t$$

We have that

$$2k_m \geq 2\left(k_m + \frac{n_b}{4}\right) + \frac{n_t}{k_t}$$

$$0 \geq \frac{n_b}{2} + \frac{n_t}{k_t}$$

which is a contradiction since at least n_b or n_t is bigger than 0. In fact, at least one edge have congestion at least $5 \cdot OPT$. This concludes the proof of the lemma.

□

Theorem 5.3. *After $|\bar{D}|$ reroute operations,* EQUILIBRIUM-ALGO *approximates* MCUF *in 3-FCNs within a factor of* 5.

Proof: Let $\bar{D}' = \{d \in \bar{D} | c(p_d) \geq 5 \cdot OPT\}$. By Lemma 5.4, each flow $d \in \bar{D}'$ can be routed through a path p' such that $c(p') \leq 5 \cdot OPT - \gamma_d$ by a single rerouting operation. Once a flow is rerouted, it does no longer belong to $\bar{D}'$. Hence, since $|\bar{D}'| \leq |\bar{D}|$, after at most $\bar{D}$ rerouting operations, each flow $d \in \bar{D}$ is such that $c(p_d) < 5 \cdot OPT$.

□

Corollary 5.5 *After $|\bar{D}|$ reroute operations,* EQUILIBRIUM-ALGO *approximates* MCUF *in 3-FCNs within a factor of* 4*, if all flows have equal size.*

We now show that the analysis is tight for equal-size flows.

Tightness analysis of Equilibrium-Algo

Equilibrium-Algo analysis is tight in the case of equal size flows. We construct an instance $I = (F, D)$ of MCUF and a routing solution R of I such that there exists a routing solution R such that $mc(I, R) = 1$ and there exists a routing solution R' such that R' is an equilibrium and $mc(I, R') = 4$. This proves that the result in Corollary 5.5 is tight.

We construct $I = (F, D)$ based on the following recursive construction. Let $k = 4n$, for a certain $n > 0$. $I_0 = (F_0, \varnothing)$ consists of a $(k, k, k, 1)$-FCN and an empty set of flow demands. $I_n = (F_n, D_n)$ is a $(k, k, k, 1 + k^2 k'_r)$-FCN, where k'_r is the number of $(2, k, k)$-FCN of I_{n-1}, constructed as follow. We denote the first $(2, k, k)$-FCN of F_n by F^1 and the following $k^2 k'_r$ $(2, k, k)$-FCN by $F_{i,j}$, with $i = 1, \ldots, k$ and $j = 1, \ldots, k$. The main idea is to map flow

demands of several I^-_{n-1} instances to distinct set of k'_r consecutive $(2,k,k)$-FCN of F_n. Given a graph G and a set of vertices $V' \subseteq V(G)$, a subgraph G' of G *induced* by V' is defined as $V(G') = V'$ and $E(G') = \{(x,y) \in E(G) | x \in V' \wedge y \in V'\}$. Let $\phi(x,y) = ((x-1)k + y - 1)k'_r + 1$. For each $i = 1,\ldots,k$ and $j = 1,\ldots,k$, denote by $N_{i,j}$ the subgraph of F_n induced by vertices in $V(F_{\phi(i,j)}) \cup \cdots \cup V(F_{\phi(i,j)+k'_r-1})$ and vertices in the last stage of F_n. Intuitively, $\phi(x,y)$ returns the index of the first $(2,k,k)$-FCN of $N_{x,y}$. Let D_{n-1} be the set of flow demand of an I_{n-1} instance. We now map flows in D_{n-1} to sets of k'_r consecutive $(2,k,k)$-FCN of F_n. For each $i = 1,\ldots,k$ and $j = 1,\ldots,k$, for each flow demand $((b^u_h, b^y_g), f) \in D_{n-1}$, add $((b_h^{u+\phi(i,j)}, b_g^{y+\phi(i,j)}), f)$ into D_n. Observe that, by construction of D_n, there does not exist a flow demand in D_n that has b^1_i as a source vertex, for any $i = 1,\ldots,k$. We create flow demands from these vertices as follows. For each $1 \leq i \leq k$, for each $j = 1,\ldots,k$, if $i \neq 1 \wedge j \neq k$, add a flow demand d from vertex b^i_1 of F_n to vertex $b_1^{\phi(i,j)}$ into D_n. Hence, $\phi(i,j)$ returns the index of the first $(2,k,k)$-FCN of F_n that contains the target vertex of the j-th flow demand that has b^1_i as a source vertex. We denote by $\rho(d)$ the subgraph $N_{i,j}$. Let $\bar{D}_n \subseteq D_n$ be the set of flow demands that have b^1_i as a source vertex, with $i = 1,\ldots,k$. Observe that, for each $i = 1,\ldots,k$ and $j = 1,\ldots,k$, there exists at most one flow demand in $\bar{D}_n$ that has a target vertex in $N_{i,j}$. This concludes the definition of I_n.

We now prove some properties of I_n, with $n = 1,2,3,4$. We first introduce some terminology. Given a routing solution R of an instance (F,D), we say that a flow demand $d = ((s,t), f_d) \in D$ is *non-reroutable* if EQUILIBRIUM-ALGO cannot reroute it to a less loaded path, i.e., there does not exist a path p between s and t such that $c(p_d) > c(p) + f_d$. A routing solution is in *equilibrium* if every flow in D is non-reroutable. For any $d \in D_n$, we denote by $D(\rho(d))$ the set of flow demands in D_n that have both their source and target vertices in $\rho(d)$ and by $\bar{D}(\rho(d))$ the set of flow demands that have their source vertex in the first $(2,k,k)$-FCN of $\rho(d)$. A *top-down* path of F is a path between a vertex in the last stage of F and a vertex in the first stage of F.

We first prove that, for any $n \geq 0$, I_n admits a solution with congestion exactly 1.

Lemma 5.6 *Given an I_n instance and a top-down path p of I_n to b^1_1, there exists a routing solution of I_n with congestion 1 and $c(p) = 0$.*

Proof: We prove that, given an I_n instance, with $n \geq 0$, given a top-down path p of I_n, there exists a routing solution R of I_n such that $mc(I_n, R) = 1$ and $c(p) = 0$. We prove it by induction on n. If $n = 0$, the statement trivially holds

since there is no flow routed through I_0. If $n \geq 1$, by inductive hypothesis, we have that for each subgraph $N_{i,j} = (F_{i,j}, D_{i,j})$, with $i = 1, \ldots, k$ and $j = 1, \ldots, k$, contained in I_n, there exists a routing solution R_{n-1} of $N_{i,j}$ such that, given an arbitrary top-down path p' of flows in $D(N_{i,j})$, each edge of $N_{i,j}$ has congestion at most 1 and $c(p') = 0$. We use this inductive hypothesis in order to build a routing solution for I_n with congestion 1 and $c(p) = 0$. Hence, we now show how to route flows in $\bar{D}_n$ and then we use the inductive hypothesis for routing flows in $D(N_{i,j})$, with $i = 1, \ldots, k$ and $j = 1, \ldots, k$. Let $p = (t_l^g, m_g^1, b_1^1)$ be the given top-down path, with $g = 1, \ldots, k$ and $l = 1, \ldots, k$. For each $i = 1, \ldots, k$, consider flow demands $d_1^i, \ldots, d_l^i$ in $\bar{D}_n$, with $l \in \{k-1, k\}$. If $i = 1 \wedge g \neq k$, route d_g^1 through (b_1^1, m_k^1, t_1^k) and through an arbitrary top-down path $p_{1,g}$ from t_1^k to its destination $b_1^{\phi(1,g)}$. If $i > 1 \wedge$, for each $j = 1, \ldots, k$, if $i = l$ route d_g^l through (b_i^1, m_g^1, t_1^g) and through an arbitrary top-down path $p_{l,g} \neq p$ from t_1^g to its destination $b_1^{\phi(l,g)}$, otherwise, if $i \neq l$, route d_j^i through (b_i^1, m_j^1, t_i^j) and through an arbitrary top-down path $p_{i,j} \neq p$ from t_i^j to its destination $b_1^{\phi(i,j)}$ Observe that, by inductive hypothesis, there exists a routing solution of flow demands in $D(N_{i,j})$ such that it has congestion at most 1 and $c(p_{i,j}) = 0$, with $i = 1, \ldots, k$ and $j = 1, \ldots, k$. Hence, since each flow in $\bar{D}_n$ is routed through a distinct path, we have that the congestion of I_n is at most 1 and $c(p) = 0$.

□

We now prove that there exists a routing solution R of I_4 such that $mc(I_4, R) = 4$ and R is an equilibrium. We first introduce the following three lemmas and then we use them to prove construct such instance I_4.

Lemma 5.7 *Given a instance $I_n = (F_n, D, n)$, consider a routing R' where only flow demands in $\bar{D}_n$ are routed. For any $d = (s, t) \in \bar{D}_n$ such that $c(p_d) \leq n$ and $c(p_d) \leq 2$, there exists a routing solution R of I_n such that d and every flow demand in $D(\rho(d))$ is non-reroutable and each flow in D_n is routed according to R'.*

Proof: We prove it by induction on n. In the base case $n = 0$, it trivially holds since D_0 of I_0 is the empty set. In the inductive step $n > 0$, consider an arbitrary routing where only flows in $\bar{D}_n$ are routed through F_n. Consider a flow demand $d \in \bar{D}_n$. If $c(p_d) = 1$, then it is easy to see that it is not possible to reroute d in order to obtain a better routing solution. In fact, for every $d \in \bar{D}_n$, we always have that $c(p_d) \geq 1$. Otherwise, if $c(p_d) = 2$, suppose that d is routed through a path (b_1^g, m_l^g, t_h^l), with $1 \leq g \leq k$, $1 \leq l \leq k$, and

$1 \leq h \leq k$. Let b_1^i be a vertex of the first $(2,k,k)$-FCN of $\rho(d)$. Observe that d is routed through $p' = (t_h^l, m_l^g, b_1^g)$ in $\rho(d)$. We consider an arbitrary routing R' of flow demands in $\bar{D}(\rho(d))$ such that (i) only d is routed through p', (ii) b_1^i routes its $k-1$ flow demands $d_1, \ldots, d_{k-1}$ through its $k-1$ neighbors $m_1^1, \ldots, m_{g-1}^1, m_{g+1}^1, \ldots, m_k^1$, and (iii) the value of the most congested edge in $\rho(d)$ is less than 2. This implies that, by construction of R, d is non-reroutable and, by inductive hypothesis, for every $d' \in \bar{D}(\rho(d))$, d' and every flow demand in $D(\rho(d'))$ is non-reroutable. Hence, every flow demand in $D(\rho(d))$ is non-reroutable, which proves the statement of the lemma also in this case.

□

Lemma 5.8 *Given an I_n instance, with $n \geq 3$, consider a routing R' where only flow demands in $\bar{D}_n$ are routed such that: (i) for each $i = \frac{k}{2}+1, \ldots, k$, edges $(m_i^1, t_1^i), \ldots, (m_i^1, t_{\frac{k}{2}}^i)$ have congestion 3, and (ii) edge $(m_{\frac{k}{2}}^1, t_1^{\frac{k}{2}})$ has congestion 2. Consider a flow demand $d = (s,t) \in \bar{D}_n$, with $c(p_d) = 3$, such that d is routed neither through $m_{\frac{k}{2}}^1$ nor through t_y^x, for any $x = \frac{k}{2}+1, \ldots, k$ and $y = \frac{k}{2}+1, \ldots, k$. There exists a routing solution R of I_n such that d and every flow demand in $D(\rho(d))$ are non-reroutable and each flow in D_n is routed according to R'.*

Proof: We compute such routing solution R as follows. Assume that d is routed through a path (b_1^g, m_l^g, t_h^l), where $m_l^g \neq m_{\frac{k}{2}}^1$ and $t_h^l \neq t_y^x$, and this path has congestion less than 4. Let b_1^i be a vertex of the first $(2,k,k)$-FCN of $\rho(d)$. Observe that d is routed through $p' = (t_h^l, m_l^g, b_1^g)$ in $\rho(d)$. For each $j = \frac{k}{2}+1, \ldots, k$, let $d_1^j, \ldots, d_k^j$ be flow demands in $\bar{D}(\rho(d))$ that have b_j^i as a source vertex. For each $l = 1, \ldots, k$, if $\lceil \frac{l}{2} \rceil + \frac{k}{2} \neq j$, let $\bar{l} = \lceil \frac{l}{2} \rceil + \frac{k}{2}$ and route d_l^j through $(m_{\bar{l}}^j, t_j^{\bar{l}})$. Otherwise, if $\lceil \frac{l}{2} \rceil + \frac{k}{2} = j$, route d_l^j through $(m_{\frac{k}{2}}^j, t_j^{\frac{k}{2}})$. For each $j = 2, 3$, let $d_1^j, \ldots, d_k^j$ be flow demands in $\bar{D}(\rho(d))$ that have b_j^i as a source vertex. For each $l = \frac{k}{2}+1, \ldots, k$, route d_l^j through (m_l^j, t_l^l). For each $j = 2, \ldots, \frac{k}{2}$, let $d_1^j, \ldots, d_k^j$ be flow demands in $\bar{D}(\rho(d))$ that have b_j^i as a source vertex. Route d_1^j and d_2^j through $(m_{\frac{k}{2}}^j, t_j^{\frac{k}{2}})$. Let $d_1^1, \ldots, d_k^1$ be flow demands in $\bar{D}(\rho(d))$ that have b_1^i as a source vertex. For each $l = 1, \ldots, k-1$, let $\bar{l} = \lceil l\frac{4}{k} \rceil$ and route d_l^1 through $(m_{\bar{l}}^1, t_1^{\bar{l}})$. Observe that $c(p') = 0$. Now, route all the remaining flows in $\bar{D}(\rho(d))$ that were not routed so far in such a way that the congestion of edges in $\rho(d)$ is at most 2 and $c(p') = 0$. This concludes the

definition of R. By construction of R, d is non-reroutable and, by Lemma 5.7, for every $d' \in \bar{D}(\rho(d))$ which has $c(p') \leq 2$, there exists a routing of flows in $D(\rho(d'))$ such that d' and every flow demand in $D(\rho(d'))$ is non-reroutable and every flow is routed according to R. Hence, every flow demand in $D(\rho(d))$ is non-reroutable, which proves the statement of the lemma also in this case.

□

It is easy to see that, by a symmetry argument, Lemma 5.8 can be used to prove the following lemma.

Lemma 5.9 *Given an I_n instance, with $n \geq 3$, consider a routing R' where only flow demands in $\bar{D}_n$ are routed such that: (i) for each $i = \frac{k}{2}+1, \ldots, k$, either edges $(m_i^1, t^i_{\frac{k}{2}+1}), \ldots, (m_i^1, t_k^i)$ have congestion 3, and (ii) edge $(m^1_{\frac{k}{2}}, t_1^{\frac{k}{2}})$ has congestion 2. Consider a flow demand $d = (s,t) \in \bar{D}_n$, with $c(p_d) = 3$, such that d is routed neither through $m^1_{\frac{k}{2}}$ nor through t^x_y, for any $x = \frac{k}{2}+1, \ldots, k$ and $y = 1, \ldots, \frac{k}{2}$. There exists a routing solution R of I_n such that d and every flow demand in $D(\rho(d))$ are non-reroutable and each flow in D_n is routed according to R'.*

We can now exploit Lemma 5.7, Lemma 5.8, and Lemma 5.9 in order to build a routing solution R of I_4 such that at least an edge has congestion 4 and R is in an equilibrium.

Lemma 5.10 *There exists a routing solution of I_4 that has congestion 4.*

Proof: We compute such routing solution R as follows. Let $d_1^1, \ldots, d_k^1$ be flow demands in $\bar{D}_4$ that have b_1^1 as a source vertex. For each $j = 1, \ldots, 3\frac{k}{4}$, let $\bar{j} = \lceil j\frac{3}{k} \rceil$ and route d_j^1 through $(m_{\bar{j}}^1, t_1^{\bar{j}})$. We have that edges $(b_1^1, m^1_{\frac{k}{4}+1}), \ldots, (b_1^1, m^1_{\frac{k}{2}})$ have congestion 3. Route $d^1_{3\frac{k}{4}+1}$ through (m_1^1, t_1^1). Both edges (b_1^1, m_1^1) and (m_1^1, t_1^1) have congestion 4. For each $i = \frac{k}{4}+1, \ldots, k$, let $d_1^i, \ldots, d_k^i$ be flow demands in $\bar{D}_4$ that have b_i^1 as a source vertex. For each $j = 1, \ldots, k$, let $\bar{j} = \big((j-1) mod\ \frac{k}{2}\big) + \frac{k}{2} + 1$, $\bar{i} = 2\big(i\ mod\ \frac{k}{4}\big) + \lceil j\frac{2}{k} \rceil$ and route d_j^i through $(b_i^1, m_{\bar{j}}^1, t_{\bar{i}}^{\bar{j}})$. We have that, for each $j = \frac{k}{2}+1, \ldots, k$, edges $(m_j^1, t_1^j), \ldots, (m_j^1, t^j_{\frac{k}{2}})$ have congestion 3. Let d_1^2 and d_2^2 be two flow demands in $\bar{D}_4$ that have b_2^1 as a source vertex. Route d_1^2 and d_2^2 through $(m^1_{\frac{k}{2}}, t_1^{\frac{k}{2}})$. Route all the remaining flows in $\bar{D}_4$ through any path that does not traverse any vertex m_j^1, with $j \geq \frac{k}{2}$, in such a way that the congestion created by these flow demands is less than

4. Observe that, edges $(b_1^1, m_1^1), \ldots, (b_1^1, m_{\frac{k}{4}}^1)$ have have congestion 4. Consider any flow demand d in $\bar{D}_4$ that is routed through any of these edges. Observe that d is routed in $\rho(d)$, through $(t_1^1, m_1^{i^*}, b_1^{i^*})$, where i^* is the index of a vertex $b_1^{i^*}$ of the first $(2,k,k)$-FCN of $\rho(d)$ in I_4. We construct a routing solution R' of flows in $D(\rho(d))$ that is similar to R. Let $d_1^{i^*}, \ldots, d_k^*$ be flow demands in $\bar{D}(\rho(d))$ that have $b_1^{i^*}$ as a source vertex. For each $j = 1, \ldots, 3\frac{k}{4}$, let $\bar{j} = \lceil j\frac{3}{k} \rceil + \frac{k}{4}$ and route $d_j^{i^*}$ through $(m_{\bar{j}}^{i^*}, t_1^{\bar{j}})$. We have that edges $(b_1^{i^*}, m_{\frac{k}{4}+1}^{i^*}), \ldots, (b_1^{i^*}, m_{\frac{k}{2}}^{i^*})$ have congestion 3. For each $i = \frac{k}{4} + 1, \ldots, k$, let $d_1^i, \ldots, d_k^i$ be the flow demands in $\bar{D}_4$ that have b_i^1 as a source vertex. For each d_j^i, with $j = 1, \ldots, k$, and t_y^x be a vertex traversed by $p_{d_j^i}$. Let $d_1^{i^*+i}, \ldots, d_k^{i^*+i}$ be the flow demands in $\bar{D}(\rho(d))$ that have $b_i^{i^*}$ as a source vertex. Route $d_l^{i^*}$, with $l = 1, \ldots, k$, through t_{k-y+1}^x. We have that, for each $j = \frac{k}{2} + 1, \ldots, k$, edges $(m_j^{i^*}, t_{\frac{k}{2}+1}^{i^*}), \ldots, (m_j^{i^*}, t_k^j)$ have congestion 3. Let $d_1^{i^*}$ and $d_2^{i^*}$ be two flow demands in $\bar{D}(\rho(d))$ that have $b_2^{i^*}$ as a source vertex. Route $d_1^{i^*}$ and $d_2^{i^*}$ through $(b_2^{i^*}, m_{\frac{k}{2}}^{i^*}, t_1^{\frac{k}{2}})$, exactly as we have done in $\bar{D}_4$. We have that, edges $(m_{\frac{k}{2}}^{i^*}, t_2^{\frac{k}{2}}), \ldots, (m_{\frac{k}{2}}^{i^*}, t_k^{\frac{k}{2}})$ have congestion 2. Route all other flows in $\bar{D}(\rho(d))$ through any path that does not traverse any vertex $m_j^{i^*}$, with $j \geq \frac{k}{2}$ in such a way that the congestion created by these flow demands is less than 4. Observe that, by construction of R and R', d is non-reroutable and, by Lemma 5.9, there exists a routing solution of flows in $D(\rho(d))$ such that every flow demand in $D(\rho(d))$ is non-reroutable and every flow demand in $D(\rho(d))$ is routed according to R'. Moreover, for every flow demand $d'\bar{D}_4$ that is routed through a path with congestion 3, by Lemma 5.8, there exists a routing R'' of flows in $D(\rho(d'))$ such that d' and every flow demand in $D(\rho(d''))$ are non-reroutable and every flow demand in $D(\rho(d))$ is routed according to R''. For all the remaining flow demands $d'' \in \bar{D}_4$ that are routed through a path that has congestion 2 or 1, by Lemma 5.7, there exists a routing solution R''' of flows in $D(\rho(d''))$ such that d'' and every flow demand in $D(\rho(d''))$ are non-reroutable and every flow demand in $D(\rho(d''))$ is routed according to R'''. Hence, R is in an equilibrium and the lemma is proved.

□

The following theorem is a direct consequence of Lemma 5.6 and Lemma 5.10.

Theorem 5.11 *There exists an instance I of* MCUF *such that $mc^*(I) = 1$ and* EQUILIBRIUM-ALGO *returns a routing solution R such that $mc(I, R) = 4$.*

5.8 Related Work

Configuring OSPF link weights and ECMP routing have been the subject of extensive research in the past two decades (in a broad variety of contexts: ISP networks, datacenters, and more). Generally speaking, research along these lines has thus far primarily focused on experimental and empirical analyses. We now discuss relevant past studies and their connections to our work. We refer the reader to [AFU12], [Rex06] and [SBT03] for more complete surveys.

TE with EMCP. We study TE with ECMP routing within the ("splittable flow") model of Fortz and Thorup [FT04]. Past work on optimizing ECMP routing mostly examined heuristic approaches (e.g., local search [FT04], branch-and-cut for mixed-integer linear programming [PAS06], memetic [BRRT02] and genetic [ERP02] algorithms) with no provable performance guarantees. [FT04] proves that MIN-ECMP-CONGESTION is NP-hard and cannot be approximated within a factor of $\frac{3}{2}$. These results leave hope that an (efficient) algorithm for configuring link weights with good (provable) guarantees is possible. Our inapproximability results for MIN-ECMP-CONGESTION, MIN-SUM-COST, and MAX-ECMP-FLOW, shatter this hope (and, in a sense, establish the necessity of heuristics).

TE with ECMP in datacenters. The emergence of datacenter networks spurred a renewed interest in interconnection networks [DT03]. Topologies such as Clos networks [AFLV08] and generalized hypercubes [ABD$^+$09, GLL$^+$09, WLL$^+$09] have been proposed as datacenter topologies. We compare Clos and hypercube networks from an ECMP routing perspective. Our analysis of Clos networks (Theorem 5.1) supports and explains (i) the experimental results in [DPHK13] regarding packet-level traffic splitting in Clos networks, and also (ii) the experimental results in [AFRR$^+$10] regarding the routing of small (mice) flows via ECMP in Clos networks. Our optimality result for Clos networks shows that the optimal link weight configurations with respect to MIN-ECMP-CONGESTION, MIN-SUM-COST, and MAX-ECMP-FLOW, can be computed independently of the actual demand matrix and can therefore be regarded as "oblivious routing". [YNDM07] presents results for oblivious routing in fat tree topologies. Our optimality result for Clos networks can be regarded as a generalization of the result in [YNDM07] for oblivious multipath routing in fat trees to more general (Clos) networks and edge capacities, and to other performance metrics (namely, MIN-SUM-COST and MAX-ECMP-FLOW).

Routing elephant flows in datacenters. Under ECMP routing, all packets

belonging to the same IP flow are routed along the same path. Consequently, a router might map large (elephant) flows to the same outgoing port, possibly leading to load imbalances and throughput losses. Optimizing routes for "unsplittable flows" is shown to be $O(\log n)$-approximable in [CCGK02] for general networks. Recent work studies the routing of unsplittable flows in Clos datacenter networks [AFRR$^+$10, CMT$^+$11, GHJ$^+$11] and experimentally analyzes greedy and other heuristic approaches, e.g., simulated annealing. We initiate the formal analysis of the routing of unsplittable flows in datacenter networks and present upper and lower bounds on the approximability of this task in Clos networks. We present, among other results, a simple, greedy 5-approximation algorithm. We point out that the key idea behind our algorithm (rerouting flows to least loaded paths until reaching an equilibrium) resembles the simulated annealing procedure in Hedera [AFRR$^+$10] and can be regarded as a first step towards analyzing the provable guarantees of this natural heuristic.

5.9 Conclusion and Future Research

We studied TE with ECMP from an algorithmic perspective. We proved that, in general, not only is optimizing link-weight configuration for ECMP an intractable task, but even achieving a good approximation to the optimum is infeasible. We showed, in contrast, that in some environments ECMP(-like) routing performs remarkably well (e.g., Random Packet Spraying in multi-rooted trees [DPHK13], specific traffic patterns). We then turned our attention to the question of optimizing the routing of elephant flows and proved upper and lower bounds on the the approximability of this task. Our results motivate further research along the following lines:

- **ECMP in datacenters.** We showed that TE with ECMP is NP-hard for hypercubes. What about approximating the optimum? Can a good approximation be computed in a computationally-efficient manner? Another interesting question is adapting this result to show similar hardness results for specific hypercube-inspired topologies (e.g., Bcube [GLL$^+$09], and MDCube [WLL$^+$09]). What about other proposed datacenter topologies, e.g., Jellyfish-like random graphs [SHPG12]?

- **Routing elephants.** We presented positive and negative approximability results for routing elephants in folded Clos networks. What is the best achievable approximation-ratio? What are the provable guarantees

of simulated annealing (see Hedera [AFRR$^+$10]) in this context? We believe that research along these lines can provide useful insights into the design of elephant-routing mechanisms.

- **ECMP with bounded splitting.** Consider a model of TE with ECMP in which, to reflect the limitations of today's routers' static hash functions used for ECMP, a router can only split traffic to a destination between a bouded number of links. What can be said about the provable guarantees of TE with ECMP in this model?

Conclusions

Understanding routing protocols dynamics is a fundamental requirement for network management that troubled the network community for decades. Controlling routing allows network operators to carefully select the most suitable paths that fully exploit their network infrastructure. On the other hand, the inability to predict routing dynamics makes it difficult to perform network debugging, routing configuration migrations, and traffic-engineering.

Our contributions focused on routing stability, the impact of local routing changes on global routing, and the problem of achieving optimal network utilization using equal-split load balancers.

Stability. We studied the problem of checking if a network is guaranteed to converge to a stable routing. Unfortunately, we found that this (and many related problems) are algorithmically hard. We investigated this problem and showed our most valuable intuition: a fascinating mapping between BGP configurations and logic circuits that puts existing results in a new perspective. In terms of computational resources, the existence of this mapping between logic circuits and BGP configurations implies that problems regarding BGP dynamics are very likely to be intractable. Overall, we believe that this mapping serves as a guide to protocol designers as it applies to a broad range of routing protocols that includes BGP as a specific case. Also, it has a great potential to foster new research on the design of routing protocols and their analysis.

We investigated whether the inherent complexity of route propagation could be mitigated. We asked whether limiting policy expressiveness could improve the protocol predictability (e.g., deterministic route propagation). Unfortunately, we observed that all the simple and realistic policy restrictions considered in research, standard and best practices do not alleviate the problem. On the positive side, we showed that computational tractability of BGP stability can be achieved when filters expressivity is limited to very simple coarse-grained

policies. While such results are primarily related to BGP, they can be generalized to any policy-based path vector routing protocol.

Impact of local routing changes on global routing. We gave several contributions to this topic by acting in behalf of a malicious AS that wants to modify Internet routing by locally forging bogus announcements. We considered three goals: creating routing oscillations, hijacking a specific flow of traffic, intercepting traffic without creating a black-hole. All these problems are clearly intractable when expressiveness of routing policies is not restricted. Hence, we assumed standard Gao-Rexford conditions, which models simple economic relationships between ASes. We showed that routing oscillations attacks are not possible when routes are steadily announced. Moreover, our main striking result states that finding an hijacking strategy in BGP is easier than in S-BGP. We stress the fact that this result is not an obvious consequence of the fact that S-BGP uses cryptography. In particular, we designed a polynomial time algorithm to device whether an attacker can hijack a specific flow of traffic. Finally, we showed how to guarantee that a successful attack does not create a black-hole, i.e., an interception attack.

There are many open problems. First, when an attacker hijacks a flow of traffic, the real owner of a prefix may counterattack and regain control of its traffic. Both the attacker and the real owner may either indefinitely continue to "battle" for this traffic or one of them will eventually prevail over its opponent. Predicting the outcome of such complex interactions is both intriguing and challenging. Moreover, counterattacking may have some undesirable side-effects, e.g., it may expose other traffic flows to the risk of being hijacked. Last, our main assumption throughout this thesis is that both topology and routing policies are known. It is interesting to study what can be derived from a partial view of the topology and routing policies.

Load-balancing. We solved a long-standing open problem in intradomain traffic-engineering where traffic can be equally split among different paths. We proved that, in general, not only computing these paths is an unfeasible task, but even achieving a good approximation to the optimum is intractable. This holds even for shortest-path routing policies. In contrast, on specific network topologies we can achieve optimal load-balancing using shortest-path routing paths. A simple lesson can be derived from these two results. The former one alerts us to the risk of poor network performances when a network is built and configured in an arbitrary way. The latter shows that carefully designing a topology is a suitable way to achieve high network utilization.

We believe that there are many attractive open problems that can still stimulate interesting research. For instance, even if we showed that the problem is NP-hard for hypercube topologies, this does not mean that a good approximation cannot be computed efficiently. Also, the application of random graphs theory to the design of network topologies is attracting the attention of the datacenter networking community [SHPG12][SGK14], especially because of the good expansion properties that there topologies expose (i.e., high throughput). However, naive approaches for setting link weights do not perform well in random graphs [SHPG12]. It is unclear whether this is a limitation of shortest-paths routing policies or an intrinsic difficulty that arises in random graphs and we believe that a study of the problem is necessary.

List of Publications

Conference publications:

1. M. Chiesa, L. Cittadini, G. Di Battista, S. Vissicchio. Local Transit Policies and the Complexity of BGP Stability Testing. In *Proc. INFOCOM*, IEEE, 2011.

2. P. Angelini, T. Bruckdorfer, M. Chiesa, F. Frati, M. Kaufmann, C. Squarcella. On the Area Requirements of Euclidean Minimum Spanning Trees. In, *Proc. WADS*, 2011.

3. A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, A.Pescape'. Analysis of Country-wide Internet Outages Caused by Censorship. In *Proc. IMC*, ACM, 2011.

4. M. Chiesa, G. Di Battista, T. Erlebach, M. Patrignani. Computational Complexity of Traffic Hijacking under BGP and S-BGP. In *Proc. ICALP*, 2012.

5. M. Chiesa, G. Lospoto, M. Rimondini, G. Di Battista. Intra-Domain Pathlet Routing. In *Proc. ICCCN*, IEEE, 2013.

6. M. Chiesa, L. Cittadini, Laurent Vanbever, S. Vissicchio, G. Di Battista. Using Routers to Build Logic Circuits: How Powerful is BGP?. In *Proc. ICNP*, IEEE, 2013. **Best Paper Award**

7. M. Chiesa, G. Kindler, M. Schapira. Traffic Engineering with ECMP: an Algorithmic Perspective. In *Proc. INFOCOM*, IEEE, 2014.

Bibliography

[ABD+09] Jung Ho Ahn, Nathan Binkert, Al Davis, Moray McLaren, and Robert S. Schreiber. HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks. In *Proc. SC*, 2009.

[AFLV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.

[AFRR+10] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. NSDI*, 2010.

[AFU12] Aysegul Altin, B. Fortz, and Hakan Umit. Oblivious OSPF Routing with Weight Optimization under Polyhedral Demand Uncertainty. *Netw.*, 60(2):132–139, September 2012.

[BAAZ11] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: fine grained traffic engineering for data centers. In *CoNEXT*, page 8, 2011.

[BCC06] T. Bates, E. Chen, and R. Chandra. BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP). RFC 4456, 2006.

[BFZ07] Hitesh Ballani, Paul Francis, and Xinyang Zhang. A study of prefix hijacking and interception in the internet. In *Proc. SIGCOMM*, 2007.

[BG11] Pradeep Bangera and Sergey Gorinsky. Impact of prefix hijacking on payments of providers. In *COMSNETS*, 2011.

[BG14] Pradeep Bangera and Sergey Gorinsky. Economics of traffic attraction by transit providers. In *IFIP Networking*, 2014.

[BGS96] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps and non-approximability – towards tight results, 1996.

[BRRT02] Luciana S. Buriol, Mauricio G. C. Resende, Celso C. Ribeiro, and Mikkel Thorup. A Memetic Algorithm for OSPF Routing, 2002.

[BT10] John S. Baras and George Theodorakopoulos. *Path Problems in Networks*. Morgan & Claypool Publishers, 2010.

[CBRV09] Luca Cittadini, Giuseppe Di Battista, Massimo Rimondini, and Stefano Vissicchio. Wheel + ring = reel: the impact of route filtering on the stability of policy routing. In *Proc. ICNP*, 2009.

[CCD+12] M. Chiesa, L. Cittadini, G. Di Battista, L. Vanbever, and S. Vissicchio. Computing with BGP: from Routing Configurations to Turing Machines. Technical report, Université Catholique de Louvain, 2012.

[CCDV11] Marco Chiesa, Luca Cittadini, Giuseppe Di Battista, and Stefano Vissicchio. Local transit policies and the complexity of bgp stability testing. In *Proc. INFOCOM*, 2011. (to appear).

[CCGK02] Amit Chakrabarti, Chandra Chekuri, Anuptam Gupta, and Amit Kumar. Approximation Algorithms for the Unsplittable Flow Problem. In *Proc. APPROX*, 2002.

[Cis11] Cisco. OSPF Design Guide, 2011. http://www.cisco.com/image/gif/paws/7039/1.pdf.

[Cit10] Luca Cittadini. *Understanding and Detecting BGP Instabilities*. PhD thesis, Università degli Studi "Roma Tre", 2010.

[CMT+11] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling Flow Management for High-performance Networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011.

[CRCD09] Luca Cittadini, Massimo Rimondini, Matteo Corea, and Giuseppe Di Battista. On the feasibility of static analysis for bgp convergence. In *Proc. International Symposium on Integrated Network Management (IM 2009)*, pages 521–528, 2009.

[CRV+11] Luca Cittadini, Massimo Rimondini, Stefano Vissicchio, Matteo Corea, and Giuseppe Di Battista. From theory to practice: Efficiently checking bgp configurations for guaranteed convergence. *IEEE Trans. Netw. and Serv. Man.*, 2011.

[CWZ00] Zhiruo Cao, Zheng Wang, and Ellen W. Zegura. Performance of Hashing-Based Schemes for Internet Load Balancing. In *Proc. INFOCOM*, 2000.

[DPHK13] Advait Abhay Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in data center networks. In *Proc. INFOCOM*, 2013.

[DT03] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[EFSW13] Roee Engelberg, Alex Fabrikant, Michael Schapira, and David Wajc. Best-response dynamics out of sync: complexity and characterization. In *EC*, 2013.

[ERC+07] Cheng Tien Ee, Vijay Ramachandran, Byung-Gon Chun, Kaushik Lakshminarayanan, and Scott Shenker. Resolving inter-domain policy disputes. In *Proc. SIGCOMM*, 2007.

[ERP02] M. Ericsson, M. G. C. Resende, and P. M. Pardalos. A Genetic Algorithm for the Weight Setting Problem in OSPF Routing. *Journal of Combinatorial Optimization*, 6:299–333, 2002.

[ES11] Roee Engelberg and Michael Schapira. Weakly-acyclic (internet) routing games. In *Proceedings of the 4th international conference on Algorithmic game theory*, SAGT'11, pages 290–301, Berlin, Heidelberg, 2011. Springer-Verlag.

[FJB07] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of autonomy for the expressiveness of policy routing. *IEEE/ACM Trans. on Networking*, 15(6), 2007.

[FMS+10] A. Flavel, J. McMahon, A. Shaikh, M. Roughan, and N. Bean. BGP Route Prediction within ISPs. *Comput. Commun.*, 33, 2010.

[FP08] Alex Fabrikant and Christos Papadimitriou. The complexity of game dynamics: Bgp oscillations, sink equilibria, and beyond. In *Proc. SODA*, pages 844–853, 2008.

[FR09] A. Flavel and M. Roughan. Stable and Flexible iBGP. In *Proc. SIGCOMM*, 2009.

[Fri01] Eby G. Friedman. Clock distribution networks in synchronous digital integrated circuits. In *Proc. IEEE*, pages 665–692, 2001.

[FRT02] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40:118–124, 2002.

[FSS06] Joan Feigenbaum, Rahul Sami, and Scott Shenker. Mechanism design for policy routing. *Distributed Computing*, 18(4):293–305, 2006.

[FT00] Bernard Fortz and Mikkel Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *Proc. INFOCOM*, 2000.

[FT04] Bernard Fortz and Mikkel Thorup. Increasing Internet Capacity Using Local Search. *Comp. Opt. and Appl.*, 29(1):13–48, 2004.

[FT06] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS Weights in a Changing World. *IEEE J.Sel. A. Commun.*, 20(4):756–767, September 2006.

[GGSS09] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In *Proc. SIGCOMM*, 2009.

[GHJ+08] Sharon Goldberg, Shai Halevi, Aaron D. Jaggard, Vijay Ramachandran, and Rebecca N. Wright. Rationality and traffic attraction: incentives for honest path announcements in bgp. *SIGCOMM Comput. Commun. Rev.*, 2008.

[GHJ+11] Albert G. Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. *Commun. ACM*, 54(3):95–104, 2011.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[GLL+09] 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. SIGCOMM*, 2009.

[GR00] Lixin Gao and Jennifer Rexford. Stable Internet Routing without Global Coordination. In *Proc. SIGMETRICS*, 2000.

[GS05] Timothy Griffin and Joao Luis Sobrinho. Metarouting. In *Proc. SIGCOMM*, pages 1–12, 2005.

[GSHR10] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. How secure are secure interdomain routing protocols? In *Proc. SIGCOMM 2010*, 2010.

[GSW99] Timothy Griffin, F. Bruce Shepherd, and Gordon Wilfong. Policy disputes in path-vector protocols. In *Proc. ICNP 1999*, 1999.

[GSW02] Timothy Griffin, F. Bruce Shepherd, and Gordon Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. on Networking*, 10(2):232–243, 2002.

[GW99] Timothy Griffin and Gordon Wilfong. An analysis of BGP convergence properties. In *Proc. SIGCOMM*, pages 277–288. ACM Press, 1999.

[GW00] Timothy Griffin and Gordon Wilfong. A safe path vector protocol. In *Proc. INFOCOM*, 2000.

[GW02] Timothy Griffin and Gordon T. Wilfong. On the correctness of ibgp configuration. In *Proc. SIGCOMM*, pages 17–29, 2002.

[Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, July 2001.

[Hol81] Ian Holyer. The NP-Completeness of Edge-Coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.

[Hop00] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, 2000. http://www.ietf.org/rfc/rfc2992.txt.

[HR06] S. Hares and A. Retana. BGP-4 Implementation Report. RFC 4276, 2006.

[Hus99] Geoff Huston. Interconnection, peering, and settlements. In *Proc. INET 1999*, 1999.

[Hus12] Geoff Huston. AS6447 BGP routing table analysis report, 2012. http://bgp.potaroo.net/as6447/.

[JKM+13] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: experience with a globally-deployed software defined wan. In *SIGCOMM*, pages 3–14, 2013.

[JR04] Aaron D. Jaggard and Vijay Ramachandran. Robustness of class-based path-vector systems. In *Proc. ICNP*, pages 84–93, 2004.

[JSW11] Aaron D. Jaggard, Michael Schapira, and Rebecca N. Wright. Distributed computing with adaptive heuristics. In *ICS*, pages 417–443, 2011.

[KKK07] Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now?! It must be BGP. In *Computer Communication Review*, 2007.

[KLS00] S. Kent, C. Lynn, , and K. Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, pages 582–592, 2000.

[LGS12] Robert Lychev, Sharon Goldberg, and Michael Schapira. Network-destabilizing attacks. *CoRR*, abs/1203.1681, 2012.

[Lin06] Peter Linz. *An Introduction to Formal Language and Automata*. Jones and Bartlett Publishers, Inc., USA, 2006.

[LN04] James R Lee and Assaf Naor. Embedding the Diamond Graph in LP and Dimension Reduction in L1. *Geometric & Functional Analysis*, 2004.

[LSZ08] Hagay Levin, Michael Schapira, and Aviv Zohar. Interdomain routing and games. In *ACM STOC*, pages 57–66, 2008.

[Mar90] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. AUSCRYPT, 1990.

[MGWR02] D. McPherson, V. Gill, D. Walton, and A. Retana. Border Gateway Protocol (BGP) Persistent Route Oscillation Condition, 2002. RFC 3345 (Implementational).

[Moy98] J. Moy. OSPF Version 2. RFC 2328, 1998. http://www.ietf.org/rfc/rfc2328.txt.

[MP06] D. McPherson and K. Patel. Experience with the BGP-4 Protocol. RFC 4277, 2006.

[MWA02] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *In Proc. ACM SIGCOMM*, pages 3–16, 2002.

[Pap94] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[PAS06] Amandeep Parmar, Shabbir Ahmedy, and Joel Sokol. An Integer Programming Approach to the OSPF Weight Setting Problem. Technical report, Georgia Institute of Technology, 2006.

[PBG05] Mukul R. Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in sat-based formal verification. *STTT*, 7(2):156–173, 2005.

[Rex06] Jennifer Rexford. Route optimization in IP networks. In *Handbook of Optimization in Telecommunications, Springer Science + Business*. Kluwer Academic Publishers, 2006.

[RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, 2006.

[SBT03] Peerapon Siripongwutikorn, Sujata Banerjee, and David Tipper. A Survey of Adaptive Bandwidth Control Algorithms. *IEEE Communications Surveys and Tutorials*, 5(1):14–26, 2003.

[SGD05] Ashwin Sridharan, Roch Guérin, and Christophe Diot. Achieving Near-optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks. *IEEE/ACM Trans. Netw.*, 13(2):234–247, April 2005.

[SGK14] Ankit Singla, Philip Brighten Godfrey, and Alexandra Kolla. High Throughput Data Center Topology Design. In *NSDI*, 2014.

[SHPG12] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Proc. NSDI*, 2012.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[SSN+13] D. Savage, D. Slice, J. Ng, S. Moore, and R.White. Enhanced Interior Gateway Routing Protocol, 2013.

[SSZ09] Rahul Sami, Michael Schapira, and Aviv Zohar. Searching for stability in interdomain routing. In *Proc. INFOCOM 2009*, pages 549–557, 2009.

[SZR10] Michael Schapira, Yaping Zhu, and Jennifer Rexford. Putting BGP on the right path: a case for next-hop routing. In *Proc. HotNets 2010*, 2010.

[TG05] T. Griffin and G. Huston. BGP Wedgies. RFC 4264, Nov 2005.

[TMS07] P. Traina, D. McPherson, and J. Scudderl. Autonomous System Confederations for BGP. RFC 5065, 2007.

[Und08] Todd Underwood. Pakistan hijacks YouTube, 2008. http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml.

[VCVB12] S. Vissicchio, L. Cittadini, L. Vanbever, and O. Bonaventure. iBGP Deceptions: More Sessions, Fewer Routes. In *Proc. INFOCOM*, 2012.

[VGE00] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 32(1), 2000.

[wik12] IP hijacking, 2012. http://en.wikipedia.org/wiki/ip_hijacking.

[WLL+09] Haitao Wu, Guohan Lu, Dan Li, Chuanxiong Guo, and Yongguang Zhang. MDCube: A High Performance Network Structure for Modular Data Center Interconnection. In *Proc. CoNEXT*, 2009.

[WMW+06] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end Internet performance. In *Proc. SIGCOMM*, 2006.

[Wol02] Stephen Wolfram. *A new kind of science*. Wolfram Media Inc., 2002.

[WRCS13] D. Walton, A. Retana, E. Chen, and J. Scudder. BGP Persistent Route Oscillation Solutions, 2013. draft-walton-bgp-route-oscillation-stop-08.

[YNDM07] Xin Yuan, Wickus Nienaber, Zhenhai Duan, and Rami Melhem. Oblivious Routing for Fat-tree Based System Area Networks with Uncertain Traffic Demands. In *Proc. SIGMETRICS*, 2007.