A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks

Renaud Hartert *, Stefano Vissicchio *, Pierre Schaus *, Olivier Bonaventure *,
Clarence Filsfils †, Thomas Telkamp †, Pierre Francois ‡

* Université catholique de Louvain † Cisco Systems, Inc. ‡ IMDEA Networks Institute
* firstname.lastname@uclouvain.be † {cfilsfil,thtelkam}@cisco.com ‡ pierre.francois@imdea.org

ABSTRACT

SDN simplifies network management by relying on declarativity (high-level interface) and expressiveness (network flexibility). We propose a solution to support those features while preserving high robustness and scalability as needed in carrier-grade networks. Our solution is based on (i) a two-layer architecture separating connectivity and optimization tasks; and (ii) a centralized optimizer called DEFO, which translates high-level goals expressed almost in natural language into compliant network configurations. Our evaluation on real and synthetic topologies shows that DEFO improves the state of the art by (i) achieving better trade-offs for classic goals covered by previous works, (ii) supporting a larger set of goals (refined traffic engineering and service chaining), and (iii) optimizing large ISP networks in few seconds. We also quantify the gains of our implementation, running Segment Routing on top of IS-IS, over possible alternatives (RSVP-TE and OpenFlow).

CCS Concepts

•**Networks** → **Network architectures; Traffic engineering algorithms; Network management;** *Routing protocols;* •**Theory of computation** → *Constraint and logic programming;*

Keywords

SDN; traffic engineering; service chaining; segment routing (SR); MPLS; ISP; optimization

*R. Hartert is a research fellow of F.R.S.-FNRS, and S. Vissicchio is a postdoctoral researcher of F.R.S.-FNRS.

SIGCOMM '15, August 17 - 21, 2015, London, United Kingdom
 ISBN 978-1-4503-3542-3/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2785956.2787495

1. INTRODUCTION

By promising to overcome major problems of traditional per-device network management (e.g., see [1]), centralized architectures enabled by protocols like OpenFlow [2] and segment routing [3] are attracting huge interest from both researchers and operators. Two features are key to this success: declarativity and expressiveness. The former improves manageability, promoting abstractions and high-level interfaces to configuration. The latter enables flexibility of network behavior, e.g., in terms of packet forwarding and modification.

Unfortunately, prior works on Software Defined Networking (SDN) do not cover carrier-grade networks, i.e., geographically-distributed networks with hundreds of nodes like Internet Service Provider (ISP) ones. Those networks have special needs: Beyond manageability and flexibility, ISP operators also have to guarantee high scalability (e.g., to support all the Internet prefixes at tens of Points of Presence) and preserve network performance upon failures (e.g., to comply with Service Level Agreements). Moreover, the large scale and geographical distribution of those networks exacerbates SDN challenges, like controller reactivity, controller-to-switch communication and equipment upgrade. Consequently, SDN solutions targeting campuses [2], enterprises [4] and data-centers (DCs) [5], cannot be easily ported to carrier-grade networks. Even approaches designed for wide area and inter-DC networks [6, 7, 8] do not fit. Indeed, they assume that (i) the scale of the network (e.g., number of devices and geographical distances) is small, (ii) scalability and robustness play a more limited role (e.g., because of the small number of destinations [6]), and (iii) the SDN controller may apply some control over traffic sources (e.g., [7]).

Nevertheless, carrier-grade networks would also benefit from an SDN-like approach. Currently, network management (i) relies on protocols with practical limitations, either in terms of expressiveness (as for link-state IGPs, constrained by the adopted shortest-path routing model) or of scalability and overhead (like for MPLS RSVP-TE, based on per-path tunnel signaling); and

(ii) requires operators to manually fine-tune those protocols and handle complex interactions between them. This likely leads to erroneous and sub-optimal network configurations, as well as deployment obstacles for new functionalities like service chaining [9].

In this paper, we propose a declarative and expressive approach to program intra-domain forwarding in carrier-grade networks. We make two sets of contributions. First, we propose an effective, easily-deployable centralized architecture, based on a clear separation between connectivity and optimization tasks. In its realization, we rely on the robustness of a link-state Interior Gateway Protocol (IGP) for the former tasks and on the flexibility and scalability of segment routing for the latter ones. Second, we design and implement a network controller called DEFO (Declarative and Expressive Forwarding Optimizer). To support declarativity and expressiveness, DEFO automatically translates high-level goals of operators into network configurations. For efficiency and scalability, DEFO computes optimized paths by smartly combining (i.e., stitching together) paths used for connectivity. Moreover, it systematically takes advantage of opportunities provided by typical carrier-grade networks. Notably, it leverages their high physical redundancy [10, 11] by natively supporting even traffic splitting over multiple (ECMP) paths[1]. As a result, DEFO achieves huge optimizations by spreading the traffic of a small number of flows on multiple (ECMP) paths.

A more detailed list of contributions follows.

Network architecture. We follow the recent trend to separate traffic engineering and connectivity tasks [6, 8, 12]. However, in contrast with previous proposals, our architecture *explicitly* separates two layers. The underlying *connectivity layer* ensures network-wide reachability, robustness and fast failure recovery, by generating and updating *connectivity paths* for each pair of nodes in the network. The overlying *optimization layer* defines optimized paths for specific flows. When defined, optimized paths overwrite connectivity ones. DEFO determines optimized paths, while the connectivity layer is configured by operators. (§2)

High-level controller interface. To support declarativity, DEFO exposes a high-level interface for network configuration, implementing a small Domain Specific Language (DSL). This interface is based on the abstractions of *forwarding functions* and *network goals*, enabling the definition of a wide range of objectives for forwarding paths. Beyond classic traffic engineering, the DEFO API supports the declaration of refined goals, including delay-respectful, multi-objective and service chaining ones. (§3)

Expressive routing model. To overcome limitations of existing models based on shortest-path routing or end-to-end tunneling, we rely on a new model, called Middlepoint Routing (MR). MR abstracts forwarding paths as concatenations of *middlepoints*, i.e., nodes between which paths are pre-defined. By natively encompassing multi-path routing, MR provides a compact representation of sets of paths. Moreover, it supports the definition of optimized paths as combinations of connectivity sub-paths. (§4)

Algorithms for efficient path computation. Computing paths that realize DEFO goals in a carrier-grade network is far from being easy. As an example, a classic traffic engineering goal consists in minimizing the maximally loaded link [13]. Even admitting arbitrary flow splitting on each node (practically impossible to achieve), computing an optimal solution for a 100-node ISP network with the linear program for the corresponding multi-commodity flow problem (MCF) took us more than one night on a powerful server. To achieve scalability and flexibility, DEFO relies on Constraint Programming (CP) [14]. Within CP, we propose efficient data structures and new heuristics that (i) implement MR and leverage it to limit the number of decision variables; (ii) provide building blocks that can be re-used for multiple goals; and (iii) compute, in few seconds, excellent solutions to problems harder than MCF. (§4)

Robust and scalable realization. Our proposed architecture realization leverages existing technologies and can be implemented today. It relies on a link-state IGP like IS-IS at the connectivity layer, and Segment Routing (SR) [3] at the optimization one. SR enables our approach to be expressive and scalable: It indeed enables routers to enrich packets with instructions on intermediate destinations to be traversed, hence providing a perfect match to our MR model. Further, we describe our implementation of DEFO. (§5)

Evaluation on real ISP topologies and traffic matrices. By computing excellent solutions to disparate goals while limiting optimization overhead, DEFO outperforms state-of-the-art traffic engineering techniques like [13, 15]. Intuitively, DEFO's optimization power comes from the adopted routing model (MR), which is less constrained than shortest-path routing and natively supports multi-path routing (contrary to end-to-end tunneling). Given its very limited execution time, DEFO can readily be used both offline (for offline traffic engineering or what-if analyses) and online (for path recomputation upon traffic changes or failures). On the implementation side, our results quantify scalability gains provided by SR (e.g., thanks to its native support for multi-path routing) with respect to alternatives like MPLS and OpenFlow. This also provides the first-of-its-kind assessment of advantages provided by SR in real networks. (§6-8)

We finally discuss possible extensions of our proposal, compare it with related works, and conclude (§9-11).

[1]we restrict to even traffic splitting as it is currently supported by all IGP-speaking routers, contrary to uneven splits.

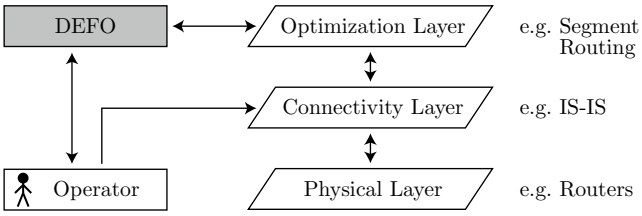

Figure 1: Proposed architecture.

2. ARCHITECTURE

In this section, we detail the role of every component of our architecture, which is illustrated in Fig. 1.

The right part of the figure highlights that two logical layers are installed on top of the physical network topology (routers and links). The underlying connectivity layer is responsible for the default forwarding behavior and for network-wide connectivity. It defines *connectivity paths* between all pairs of routers, hence for any traffic flow possibly traversing the network. In contrast, the optimization layer defines exceptions to this default routing behavior. It implements *optimized paths* used for a subset of the traversing flows (e.g., a subset of router pairs). Optimized paths overwrite connectivity ones: Whenever an optimized path is defined, the corresponding flows always use it. While connectivity and optimization layers represent a logic separation, they can also be implemented by different protocols running on the routers. This provides a cleaner design and comes with a set of advantages. For example, it enables changes in the optimization layer (expected to be frequent) without any impact on the connectivity one. It also simplifies the assessment of the impact of changes in the connectivity layer (e.g., consequently to less frequent events like failures). Finally, it ensures that controller failures do not disrupt forwarding paths.

The connectivity layer is configured (possibly once in the network lifetime) by operators. This choice is motivated by several reasons. First, it maximizes the utility of the operators' domain knowledge (expected traffic matrix, capacity of links, etc.) to provide a good basis for network optimization. Second, it leaves operators with a mean to take back the control of the network, e.g., in the case of major problems on the controller or for maintenance operations that cannot be supported by DEFO (e.g., router physical replacement). Third, it facilitates progressive deployment of our proposal.

DEFO controls the optimization layer to fine-tune forwarding. It receives high-level goals from operators, and automatically translates them into optimized paths, e.g., configurations of the optimization-layer protocol. Since optimized paths are preferred over connectivity ones, DEFO decides the actual forwarding paths used for any traffic flow crossing the network. DEFO can perform this goal-to-path translation for any change of connectivity paths, traffic flows or physical topology, e.g., for online traffic engineering.

function	DSL syntax	semantics
max load	d.`load`	maximum load of any link in $F(d)$
max delay	d.`delay`	maximum delay of source-destination paths in $F(d)$
deviations	d.`deviations`	number of deviations from connectivity paths in $F(d)$
traversal	d `passThrough` S	true if $F(d)$ crosses any node in S, false otherwise
sequencing	d `passThrough` S_1 `then` S_2 ... `then` S_k	true if $F(d)$ sequentially crosses nodes in $S_1 \dots S_k$
avoid	d `avoid` S	true if no node in S is also in $F(d)$, false otherwise

d, $F(d)$ and $S, S_1, \dots, S_k$ respectively represent any demand, the forwarding paths for d, and sets of network nodes.

Figure 2: Constructs of the current DEFO DSL.

3. EXPRESSING NETWORK GOALS

DEFO exposes a high-level interface, based on a small Scala DSL [16]. It enables operators to intuitively declare desired characteristics of forwarding paths.

The central abstractions used in the interface are the concepts of demand, forwarding function and goal. A *demand* is an aggregate of flows. In the following, we focus on demands at the granularity of source-destination pairs, for simplicity. A *forwarding function* maps a set of links to the value of a parameter (e.g., maximum load) associated to them. Our DSL includes constructs for pre-defined forwarding functions. It permits forwarding functions to be applied to sets of links, to paths or to demands. In the latter case, a forwarding function maps a demand to the value of a parameter associated to the forwarding paths for that demand. Fig. 2 details the forwarding functions currently supported by DEFO (when applied to demands). A *goal* is defined by forwarding functions applied to demands. Those functions can be composed as (i) constraints that restrict the set of forwarding paths for given demands; and (ii) objective functions specifying parameters to be optimized.

In the following, we show examples of DEFO goal definitions. They illustrate both usage of forwarding functions and practically-meaningful compositions of them. The examples include standard Scala keywords like `val` for value declaration and `<-` for variable assignment in a loop. Variables storing parameters not determined by DEFO, like the network topology (`topology` variable) or sets of traffic demands (`Demands`, `LowDelayDemands` and `SecDemands` variables), are dynamically initialized at runtime, e.g., reading from preconfigured input files provided by operators or routing daemons.

Classic Traffic Engineering Goals. The use of resources often needs to be optimized in carrier-grade networks. For example, the classic *MinMaxLoad* goal consists in minimizing the load of the maximally loaded link, e.g., to avoid performance bottlenecks. It can be declared in DEFO in two lines.

```
var MaxLoad = max(for(l<-topology.links){yield l.load})
val goal = new Goal(topology){ minimize(MaxLoad) }
```

Tactical Traffic Engineering Goals. To limit traffic disruptions, operators are often reluctant to change many forwarding paths, and can prefer sub-optimal configurations to an unbounded number of path changes. DEFO forwarding functions can be combined to express those tactical goals. As an example, the following code instructs DEFO to find the best solution bringing all link utilization under a certain threshold with at most 2 deviations from connectivity paths.

```
val goal = new Goal(topology){
 for(d<-Demands) add(d.deviations <= 2)
 for(l<-topology.links) add(l.load <= 0.9 l.capacity)
 minimize(MaxLoad)}
```

Refined Goals. Operators often have to accommodate specific (e.g., per-customer) needs. In DEFO, goals including arbitrary combinations of the forwarding functions in Fig. 2 are supported by adding constraints and changing the objective function. The following example shows how to define a *delay-respectful goal*, that is, a variant of MinMaxLoad with constraints on the maximum delay experienced by specific demands. In the example, the latency of every network path is assumed to be known by DEFO, e.g., as a result of few active measurements. Our DSL also permits to define other delay-respectful goals, e.g., with delay constraints expressed as a percentage of their current values or delays for given demands in the objective function.

```
val goal = new Goal(topology){
  for(d <- LowDelayDemands) add(d.latency <= 10.ms)
  minimize(MaxLoad)}
```

DEFO also supports *multi-objective goals* [17]. The next snippet shows a goal in which both the maximum link load and the average latency have to be optimized.

```
val goal = new Goal(topology){
  minimize(MaxLoad, AvgLatency)}
```

Support for New Applications. Many operators have lately shown interest for *service chaining*, i.e., the ability to steer packets through a sequence of services. In the example below, each demand in `SecDemands` is forced to pass first through one firewall in `FirewallSet` (whose position is assumed to be known by the operator) and through either `IPS1` or `IPS2` after.

```
val goal = new Goal(topology){
  for(d <- SecDemands){
    add(d passThrough FirewallSet then ('IPS1, 'IPS2))}
  minimize(MaxLoad)}
```

The `avoid` and `passThrough` constructs can also be used to specify *anycast* goals, in which a set of routers must be avoided or forcedly used. This meets the needs of operators that have to comply with political or business rules, like preventing or ensuring that given traffic flows cross a specific country.

Finally, operators can run DEFO to achieve the declared goal, possibly specifying an upper bound for its computation time.

```
DEFOptimizer(goal).solve(30.sec)
```

Note that infeasible goals can be defined in our DSL, e.g., trying to force a demand through an isolated node or requiring a link utilization that cannot be achieved on the given topology and input demands. In our implementation, DEFO discards unsatisfiable constraints, computes a solution satisfying the other ones, and returns a warning to the operator. Similarly, if DEFO does not terminate in the specified time, it returns the best solution found during that time.

4. OPTIMIZED PATH COMPUTATION

In this section, we describe how DEFO computes forwarding paths. Namely, we detail how it tackles the challenges posed by the need for (i) supporting a wide variety of possible input goals (defined by arbitrary combinations of forwarding functions in Fig. 2), and (ii) fast computation of solutions to computationally-hard (translation) problems. First, we present the Middlepoint Routing model, internally used by DEFO for a compact representation of paths (§4.1). Then, we overview DEFO formal representation of input goals in terms of Middlepoint Routing instances (§4.2). Finally, we describe the optimization algorithm that it runs to compute compliant optimized paths (§4.3).

4.1 The Middlepoint Routing Model

DEFO relies on the Middlepoint Routing (MR) model. MR is similar to the pathlet routing model [18, 19] in that it represents paths as concatenations of sub-paths. In contrast to pathlet routing, however, MR is based on forwarding graphs rather than single paths. This generalization enables to incorporate equal-cost multipath in the model, and to provide a compact representation of several distinct paths between any source-destination pair. Moreover, we use MR to represent forwarding paths rather than for routing protocol design.

An MR instance represents acyclic paths from any source s to any destination d as sequences of acyclic graphs $\mathcal{S} = G_1, \ldots, G_n$, with $n \geq 1$. In $\mathcal{S}$, any pair of consecutive graphs G_i and G_{i+1} (with $i = 1, \ldots, n-1$) share a single common node m_i, which is the sink of G_i and the root of G_{i+1}. We refer to any graph in $\mathcal{S}$ as *partial forwarding graph (PFG)*, and to any shared node between two consecutive PFGs as *middlepoint*. Of course, a PFG can represent a single path. In an MR instance, a source-destination pair is associated to one or more sequences $\mathcal{S}_1, \ldots, \mathcal{S}_N$ of PFGs. As an illustration, consider Fig. 3. Circles and arrows respectively represent routers and corresponding forwarding next-hops, while dashed squares identify PFGs. The MR representation of the paths from s to t is then given by sequences $\mathcal{S}_1 = [G_{(s,m)}, G_{(m,t)}]$ and $\mathcal{S}_2 = [G_{(s,t)}]$.

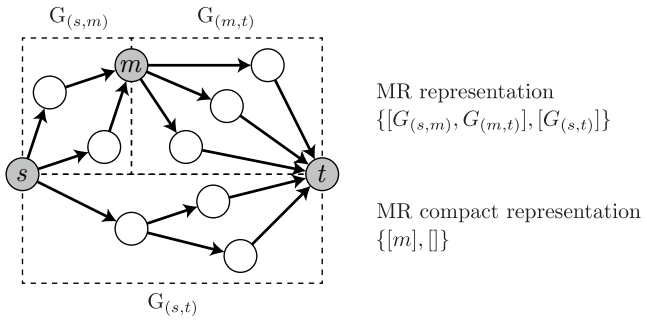

Figure 3: MR representations of $s-t$ paths.

In the specific case of our two-layer architecture, each PFG represents connectivity paths. Since PFGs are defined by the connectivity layer, we can simplify the representation of optimized paths as a set of middlepoint sequences. The right part of Fig. 3 also reports the simplified MR representation of $s-t$ paths, which is $\{[m], []\}$, under the assumption $G_{(s,t)}$ represents the connectivity paths from s to t.

Intuitively, DEFO uses the compact representation of paths provided by MR to limit the number of decision variables in the forwarding optimization. Indeed, it associates decision variables to the sequences of middlepoints to be used for a given traffic flow. In Fig. 3, for instance, DEFO uses 2 variables (for the two middlepoint sequences from s to t) instead of 8 (as needed if variables were associated to single paths) or 23 (as needed to represent traversed links and paths, like in classic linear programming formulations). In the following, we explain how DEFO computes the value for middlepoint sequences.

4.2 Network Goal Formalization

DEFO formalizes an input goal as an (initially empty) MR instance with associated constraints and optimization functions. This formalization is based on the *Constraint Programming (CP)* optimization framework [14]. A CP problem is defined by a set of variables, each having its own finite domain of possible values, and a set of constraints that apply to them. Contrary to linear programming, constraints are implemented by algorithms that preserve consistency between variable domains. For this reason, CP supports high-level, independent and easily composable constraints. An objective function can also be added to a CP problem.

We now describe *novel variables, data structures and constraints* used for goal formalization in DEFO. Consistently with §3, we assume that topology, demands and parameters (like path delays) independent from DEFO computations are provided as an input. More details on our CP formalization are reported in [20].

Middlepoint variables model the MR representation of per-demand forwarding paths. Every input demand is mapped to a middlepoint variable, representing a set of middlepoint sequences. We say that a middlepoint variable is *assigned* to a value when all the represented sequences end with the destination of the corresponding demand. If this condition does not hold, we say that the middlepoint variable is *unassigned*. For example, the forwarding paths for the demand from s to t in Fig. 3 are represented by an assigned middlepoint variable $\{[t], [m, t]\}$. Middlepoint variables represent the *decision variables* of the optimizations performed by DEFO. When forwarding paths need to be optimized, the middlepoint variables are re-initialized to an empty value. Then, they are progressively assigned to values according to the algorithm described in §4.3.

Forwarding function algorithms guarantee the satisfaction of constraints defined on forwarding functions. Every supported forwarding function implemented by a specific algorithm, specialized for that function. For example, the `load` and `delay` forwarding functions are supported by different algorithms. Forwarding function algorithms (i) extract the value of the associated forwarding function (for example, the load of the maximally loaded link for the `load` forwarding function algorithm) from a set of links or from forwarding paths corresponding to the value of a middlepoint variable; (ii) compare the extracted value with a configured threshold, to assess if the represented constraint is satisfied; and (iii) reduce the domain of middlepoint variables by excluding values that violate a constraint on the associated forwarding function. The thresholds checked by those algorithms are initialized by the constraints of the input goal. For example, a constraint `l.link < l.capacity` defined on a link `l` initializes the value to be checked on `l` by the `load` forwarding function algorithm to the link capacity provided by the input topology.

4.3 Middlepoint Selection

We propose an *efficient algorithm* to solve CP problems corresponding to DEFO input goals. By assigning values to middlepoint variables, our algorithms compute which middlepoints to use in the optimized paths of which traffic flow. The same algorithm can also be used to compute backup paths (e.g., to be pre-installed in routers as in well-known fast reroute techniques [21]).

Unfortunately, selecting middlepoints is a hard problem. Indeed, we proved [20] that middlepoint selection problems are NP-hard, even if only link capacity constraints have to be respected. Additional constraint or specific objective function can make the corresponding selection problem even harder. Despite this, our algorithms are required to be efficient and scalable, for the controller to quickly react to events (failures, demand and goal changes, etc.) in large-scale networks.

To quickly compute good solutions, we then propose a *heuristic approach*, mixing a pure CP solution with a local search technique called Large Neighborhood Search (LNS). Intuitively, we use CP to compute the best solution in a given portion of the search space, and we rely

While all the demands have not been optimized:

1. Select a non-optimized demand D which corresponds to the worst value of the objective function (e.g., the most bandwidth-consuming one). Let m_D be the middlepoint variable for D;
2. If m_D cannot be expanded without violating some constraint, backtrack by reconsidering the current value of m_D;
3. Otherwise, expand m_D as follows:
   (a) If the destination of D is a valid expansion candidate, expand m_D to the destination and store the demand as optimized;
   (b) Otherwise, expand m_D to a middlepoint n that locally optimizes the value of the objective function.
4. Update the domain of other variables.

Figure 4: The first-close heuristic that drives the construction step in DEFO.

on LNS to heuristically decide the sequence of subspaces to explore. Starting from a state in which all middlepoint variables are unassigned, DEFO computations keep alternating (i) a *construction step with inference*, used to identify a current best solution; and (ii) a *partial reset step*, which resets a random part of the current best solution. The partial reset step reworks a significant part of the current best solution, hence increasing the likelihood to escape local optima and quickly find solutions close to the optimum. The price for such efficiency consists in losing the proven optimality of a pure CP approach: Indeed, the random-based reset step does not guarantee that the entire search space is explored.

Forwarding Optimization Algorithms. While combining LNS with CP is not completely new [22, 23], we devised *original algorithms* to implement both the construction and the partial reset steps in DEFO.

Construction Step. Starting from an empty or partial solution (with some unassigned variables), this step *expands* all middlepoint variables, assigning values to them. Its output is a complete variable assignment compliant with all input constraints.

In DEFO, the construction step is implemented by a depth-first search combined with branch and bound [24]. This search is driven by the *first-close heuristic* summarized in Fig. 4. Our heuristic iterates over the non-optimized demands, i.e., those with unassigned middlepoint variables. At every iteration, it identifies the demand with the biggest negative contribution to the objective function. Then, it first tries to complete the assignment of the corresponding middlepoint variable by appending the destination to all its internal middlepoint sequences. This attempt is intended to reduce the number of middlepoints in optimized paths, hence the deviations from connectivity paths and the amount of information to be injected in the optimization layer. If

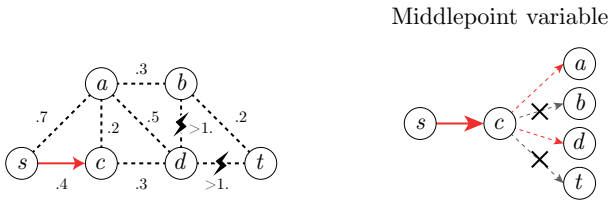

Figure 5: Illustration of a construction-step iteration (with corresponding domain reduction). Numbers aside links represent their utilization.

such an assignment is not feasible, the heuristic expands the middlepoint variable by greedily selecting the middlepoint that locally optimizes the objective function without violating any constraint: This middlepoint is either appended to all internal sequences of the considered middlepoint variable (if possible) or used in a new sequence. The final value of the middlepoint variable is then incrementally built repeating those operations. If no expansion is possible at a given iteration, the algorithm backtracks and replaces the last visited middlepoint with an alternative one.

All those operations are implemented with an inference approach, based on a continuous update of the variable domains. To this end, we run forwarding function algorithms to exclude possible expansions which are not compliant with the input constraints and adjust the domain of every unassigned variable accordingly.

Fig. 5 illustrates an iteration of the construction step algorithm for the middlepoint variable m associated to the demand from s to t. The input goal constrains every link to carry less load than its capacity and aim at minimizing the maximum link load. In the given network, links (d, b) and (d, t) are over-utilized. Assume that connectivity paths are computed as shortest paths and all links have unitary weight. Since connectivity paths from s to t cross (d, t), it is not possible to complete the assignment of m by directly appending the destination t. Hence, our heuristic updates the value of m to $\{[c]\}$, locally minimizing the objective function. Moreover, it runs forwarding function algorithms to update unassigned variable domains. To avoid links (d, b) and (d, t), the `load` forwarding function algorithm excludes b and t from the domain of m, leaving only a and d in it. The first-close heuristic is then re-run until t is reached. Note that values discarded in one iteration may be used in successive iterations. For example, while b is not in the m's domain in the depicted iteration, it re-enters the domain whenever m's value is updated to $\{[c, a]\}$.

Partial Reset Step. The first-close heuristic locally optimizes the expansion of middlepoint variables but does not provide any guarantee on the global optimality of the computed variable assignment. We rely on the partial reset step to statistically ensure the good quality of the returned solution. In this second step, DEFO resets a (large) subset of middlepoint variables of the

1. Let $\mathcal{D}$ be the set of demands with the worst value of the objective function;
2. While fewer than k demands and not all the demands in $\mathcal{D}$ have been selected
   (a) Sort non-extracted demands in $\mathcal{D}$ according to their objective function value (from the worst to the best). Let S be the resulting sorted array;
   (b) Generate a random number $r \in [0, 1[$;
   (c) Extract the $\lfloor r^{\alpha}|S|\rfloor$-th demand from S.
3. Re-initialize the value of all middlepoint variables corresponding to extracted demands.

Figure 6: Partial reset algorithm.

current best solution. This operation identifies a new starting state from which the construction step is rerun, to explore another portion of the search space.

Our proposed partial reset algorithm is summarized in Fig. 6. It randomly selects the middlepoint variables to reset among those associated to demands with the worst value of the objective function, e.g., routed through the most congested link in the MinMaxLoad goal. Let $\mathcal{D}$ be the set of those demands. To allow fine-tuning of the optimization process, the algorithm is parametric: (i) $k \in [1, 15]$ controls the number of variables to reset; and (ii) $\alpha \in [1, 6]$ influences the random choice of which variables are selected. In particular, if $\alpha = 1$, all middlepoint variables corresponding to the demands in $\mathcal{D}$ have the same chance to be selected. In contrast, if $\alpha = \infty$, the k demands in $\mathcal{D}$ with the worst value of the objective function (e.g., highest bandwidth) are deterministically selected. We change the value of k and α during the computation. Namely, depending on whether the previous construction step improved the current best solution or not, we respectively decrease or increase k by 1, hence causing the reset of more or less variables. This is meant to dynamically control the size of the explored parts of the search space during the optimization in order to quickly converge to good solutions and escape local optima. Moreover, to progressively allow more diversification, we decrease α (initially set to 6) with the number of iterations. Values and bounds of the parameters are determined empirically.

5. REALIZATION

In this section, we describe our proposed realization of the architecture depicted in Fig. 1.

A link-state IGP at the connectivity layer. Link-state (LS) IGPs are the routing protocols currently used inside carrier-grade networks. They are well-understood; their implementation is robust to software bugs and optimized for fast convergence [25]. Built-in features also ensure fast reroute in many failure cases [10]. Moreover, LS IGPs natively support (ECMP) multi-path routing through even splitting over multiple shortest paths, a crucial feature to exploit the physical redundancy of carrier-grade networks. Finally, their information flooding mechanism allows DEFO to easily communicate with devices and monitor topological changes. Being based on shortest-path routing, the main limitation of LS IGPs is their relative inflexibility. However, flexibility is not critical for the connectivity layer.

Segment Routing at the optimization layer. Contrary to the connectivity layer, flexibility in the definition of forwarding paths and scalability in the introduced overhead are the main requirements for the optimization layer. We propose to rely on segment routing (SR), in order to avoid both limitations of shortest-path based protocols as well as scalability issues of end-to-end tunneling (e.g., RSVP-TE) and hop-by-hop (OpenFlow) ones. SR is a recently-standardized protocol [3], supported by major router vendors and attracting growing interest from operators. In SR, routers can enrich (MPLS [26] or IPv6 [27]) packet headers with a sequence of *segments*, i.e., instructions forcing packets through specific intermediate nodes or links. The IGP ECMP shortest paths are used to reach those intermediate points. SR provides a perfect match with the MR model used in DEFO. Indeed, any middlepoint used in an MR instance can be mapped to an SR segment, in constant time. Thus, paths computed by DEFO for a given demand can be efficiently translated into sequences of SR segments to be applied by the ingress router for that demand. This has the additional side effect of limiting the optimization overhead to sequences of SR segments (instead of full paths) configured only at ingress routers (with no additional state on internal ones). We evaluate scalability gains of using SR in §8.

Scala implementation of DEFO. We implemented DEFO in about 9,000 lines of Scala code, on top of the OscaR open-source CP solver [28]. Our implementation is provided at [20]. It supports all the constructs defined in Fig. 2, hence all the use cases presented in §3. To gather information on connectivity paths and network topology, DEFO can participate in the link-state IGP by maintaining a single adjacency with any IGP router, as in [29, 30]. To configure the optimization layer, it has to push specific SR configurations to routers. The control channel between DEFO and network routers can be implemented in several ways, including OpenFlow [31] and slight modifications of configuration protocols, like Netconf [32] or Flowspec [33]. For compatibility with current routers, we prefer to rely on the PCEP extension for SR [34].

6. EVALUATION SETUP

We evaluated our realization on multiple topologies and demand matrices. We used real and realistic topologies and matrices to assess the results of DEFO in practical cases, and we relied on synthetic data to explore its behavior in corner cases. For example, we exper-

Type	ID	# nodes	# links	# demands
Real ISP	R1	600	2150	11,500
Real ISP	R2	400	1200	116,000
Real ISP	R3	200	750	10,000
Real ISP	R4	150	700	24,000
Inferred	I5-9	79-317	296-1,946	6,000-96,000
Synthetic	S10-11	50-100	280-570	2,500-10,000

Table 1: Evaluation dataset summary

imented with very connected topologies (i.e., with an average node degree higher than 5) and uniform demand matrices to perform stress tests on DEFO efficiency.

Our dataset is summarized in Table 1. The table provides more detailed information (approximated to preserve anonymity) on real-world ISP topologies. The other topologies in our dataset include inferred ones in the Rocketfuel project [35] and synthetic ones generated with the Delaunay triangulation algorithm in IGen [36]. The table reports aggregated information about those topologies, since they are publicly available at [20].

To run experiments on delay-constrained goals, we enriched each topology with path delays. We relied on direct input from operators for real topologies. In the other cases, we assigned path delays proportionally to the physical distance between the geographical positions of path endpoints.

Our dataset also contains demand matrices (one snapshot per topology). For all real topologies, we used demands provided by operators. In the other cases, we relied on synthetic demand matrices computed with the approach described in [37]. This approach is based on a gravity model fed with independently and identically distributed exponential random variables. It produces highly-skewed demand matrices which are quite realistic, as shown in [37] and as we confirmed by a comparison with real matrices in our dataset. Finally, to consider extreme use cases for worst-cases analyses, we proportionally inflated all the demands until a maximum link usage of at least 120% was reached.

The experiments for our evaluation were run on MacBook Pro laptops, with 2.6 GHz Intel CPU and 16 GB of memory. Unless differently specified in the text, DEFO has been configured with a timeout of 3 minutes for each experiment. Moreover, we set the maximum number of middlepoints per optimized path to 2. Indeed, we experimentally noticed that DEFO solutions with this setting are significantly better than those with at most 1 middlepoint per optimized path, while further increasing the number of middlepoints per path tends to lead to less significant improvements.

7. OPTIMIZED PATH EVALUATION

We now present an experimental evaluation of DEFO. We first compare the forwarding optimization achieved by DEFO on classic goals with solutions provided by Cisco MATE [38], a commercial traffic engineering tool

Topo ID	Initial Load	Optimum	DEFO ($\leq$ 3mins)	cRSVP-TE	IGP-WO
R1	121%	81%	90%	81%	96%
R2	120%	89%	94%	90%	108%
R3	120%	*NA*	94%	80%	107%
R4	121%	86%	89%	86%	86%
I5	130%	86%	86%	86%	95%
I6	124%	*NA*	76%	77%	77%
I7	142%	72%	82%	76%	80%
I8	120%	66%	72%	69%	78%
I9	195%	*NA*	91%	70%	99%
S10	103%	69%	71%	73%	91%
S11	250%	56%	75%	69%	120%

Table 2: Comparison on maximum link load minimization.

(§7.1-7.2). Namely, we used MATE to simulate IGP weight optimizations (IGP-WO) and RSVP-TE tunnel optimization. The former influences forwarding path computation by tweaking link weights on the topology shared by IGP routers. The latter optimizes paths through explicit RSVP-TE tunnels. We evaluated centralized and distributed RSVP-TE optimizations, which we refer to as cRSVP-TE and dRSVP-TE respectively. In cRSVP-TE a central controller setups the needed tunnels, while a full-mesh of tunnels is individually optimized by single routers (tunnel endpoints) in dRSVP-TE. In the second part of the section, we show results on DEFO support for custom goals (§7.3) and new applications (§7.4), which are largely unsupported by available tools. Those experiments show the effectiveness and time efficiency of DEFO, and the possibility to use it both offline (offline traffic engineering, planning, what-if analyses, etc.) and online (online traffic engineering, reaction to failures, etc.).

7.1 DEFO Quickly Computes Excellent Solutions for Classic Goals

We start by evaluating DEFO on classic traffic engineering (TE) goals. We take as an example the MinMaxLoad goal that aims at maximizing the spare capacity of the maximally-loaded link (see §3).

Table 2 compares the quality of the solutions (in terms of maximal link load) achieved by the considered optimization techniques with the theoretical optimum. Optimum results are computed by running the standard Linear Program (LP) formalizing the multi-commodity flow problem [39]. This LP assumes the possibility for any network node to fractionally split flows with arbitrary proportions. Such splitting ability is not realistically feasible in practice even with uneven load balancing, because it would require to know in advance the size of all flows and continuously change split proportions according to flow sizes. We provided this LP as input to the commercial Gurobi solver. The computation always ran out of memory on the laptop used in our evaluation, except for 3 topologies. We then re-ran the LP solver on a much more powerful server (32 CPU

cores and 96 GB of RAM), and reported the results in Table 2. We stress that the LP could not be solved in some cases (*NA* values in the table), even on that server.

DEFO computes close-to-optimum solutions. Despite restricting to even load-balancing over IGP paths, DEFO computes forwarding paths that lower the maximum load to a value close to the theoretical optimum. In some cases, this value is even equal to the optimum value, as for $I5$. In all the other cases except one, it is less than 10% worse than the optimum. Moreover, DEFO terminates in 3 minutes (by setting, see §6) on a commodity laptop, while running the LP took us up to many hours (including an entire night for the experiment on $R2$) and sometimes ran out of memory on a much more powerful server.

DEFO optimizes more than IGP-WO. IGP weight tweaking techniques do not have fine-granularity. In particular, they do not allow to modify forwarding paths on a per-demand basis. As a result, contrary to DEFO, IGP-WO could not avoid congestion in some of our experiments. Indeed, the utilization of the maximally-loaded link is above 100% for $R2$, $R3$, $S11$ (i.e., 3 out of our 11 topologies). More generally, IGP-WO results are much less close to the optimum than DEFO ones.

DEFO optimization power is comparable with RSVP-TE. RSVP-TE tunnels allow to enforce arbitrary paths for each demands. Despite this allows fine-grain optimization, results achieved by DEFO tend to be close and sometimes even better than the ones obtained with cRSVP-TE (as for $I6$ and $S10$). Also, DEFO comes with the additional advantage of much higher time efficiency. It indeed always terminated in 3 minutes (by setting, see §6), while cRSVP-TE optimization took hours on our largest topologies.

7.2 DEFO Can Ease Network Operation

Operators often try to limit the number of applied changes. This eases the deployment of the optimized configuration, without the risk of triggering huge traffic shifts and transient disruptions. To evaluate support for those kind of tactical use cases, we experimented with a variant of MinMaxLoad which minimizes the number of optimized paths while ensuring that all links are utilized for at most 90% of their respective capacity.

DEFO is much more effective than incremental IGP-WO. MATE supports an incremental IGP-WO procedure that tries to lower the utilization of the maximally-loaded link with a limited number of IGP weight changes. In our experiments, it did not succeed in lowering the maximally loaded-link under 100% for 3 topologies out of 11. In contrast, DEFO finds a congestion-free solution in all our experiments. Also, observe that the IGP weight changes are more intrusive than those applied in our architecture. For example, IGP weight changes can create forwarding loops [40] and traffic shifts [41] for Internet destinations. In contrast, SR prevents such problems by clearly separating forwarding optimization from IGP-based connectivity.

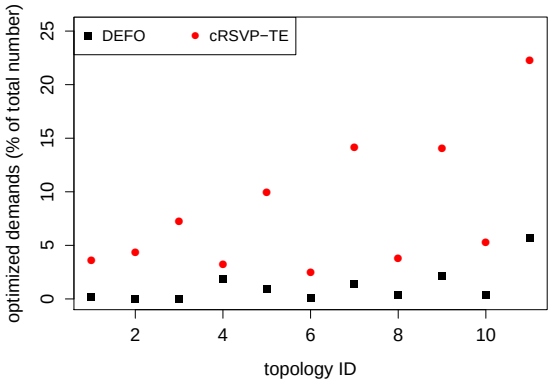

Figure 7: Optimization overhead to bring utilization of the maximally loaded link under 90%.

Load	Min	1st Qu.	Median	3rd Qu.	Max
<98%	0.17s	0.54s	1.38s	3.16s	12.83s
<95%	0.17s	0.59s	1.55s	3.56s	19.78s
<90%	0.22s	0.71s	2.05s	4.33s	46.8s

Table 3: DEFO computation time to lower the maximum link load under a given threshold.

DEFO reduces the overhead of RSVP-TE optimizations. We also let MATE optimize RSVP-TE tunnels according to the considered MinMaxLoad variant. As in Table 2, the resulting maximal load is comparable with DEFO output paths. However, Fig. 7 shows that many more demands have to be optimized with cRSVP-TE. Indeed, cRSVP-TE optimization often affects one order of magnitude more demands than DEFO, that is, typically thousands of demands instead of hundreds. Even worse, dRSVP-TE (omitted in Fig. 7 for readability) requires to optimize 2 to 3 orders of magnitude more demands than DEFO. Hence, both cRSVP-TE and dRSVP-TE optimizations imply a significant increase in (i) routers to be re-configured; (ii) traffic to be shifted from one path to another; and (iii) control-plane overhead. Those three factors contribute to a number of management issues including delay in the application of TE actions and reduced ability for operators to debug post-optimization configurations.

Recent versions of MATE and the commercial WAE controller [42] support a heuristic to compute SR configurations for tactical TE [43]. It achieves optimization overhead reductions very similar to DEFO: Those results are not reported in Fig. 7 for readability.

Another fundamental factor from an operational viewpoint is the computation time. We considered variants of MinMaxLoad with the maximum load constrained to be at most 90%, 95%, or 98%. Consistently with Table 2, DEFO always found a compliant solution except when run on $I9$ with the 90% constraint. In the latter case, it reported the best solution that it found

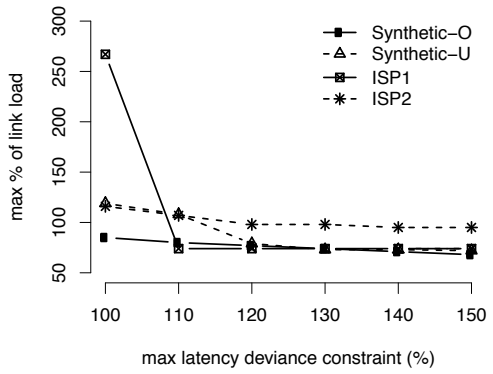

Figure 8: Results of DEFO for the delay-constrained MinMaxLoad goal.

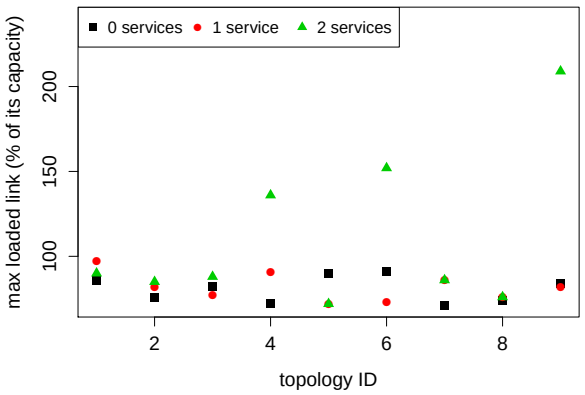

Figure 9: DEFO optimization in different service chaining scenarios.

(with 91% maximum link utilization) after the configured 3-minute timeout. Table 3 reports the time taken by DEFO in all the other experiments. The table shows that DEFO always completed its computation in less than one minute. In the vast majority of the cases, it actually took at most 10 seconds; not rarely, it terminated in a fraction of a second. Those results show that DEFO can readily be used for periodic forwarding optimization, e.g., for online traffic engineering.

7.3 DEFO Supports Refined Goals

We now evaluate DEFO on refined goals, not supported by current IGP-WO and RSVP-TE techniques. We implemented support for some of those goals in a recent SR controller [42].

Delay-respectful TE. We ran experiments on a MinMaxLoad variant with additional constraints forcing the maximum delay of every forwarding path to be lower than a pre-defined threshold. Precisely, for each source-destination pair, we constrained the post-optimization path delay to be lower than a given percentage variation of the initial one. We ran different sets of experiments. In each experiment set, we bound the delay increase to respectively be at most 0% (i.e., no increase admitted), 10%, 20%, 30%, 40% and 50% of the initial delay.

The results obtained on two real topologies ($R1$ and $R2$) and a synthetic one ($S11$) are plotted in Fig. 8. We used two configurations for the synthetic topology, one with uniform link weights (Synthetic-U) and the other with optimized weights (Synthetic-O). The figure correlates the maximum link utilization (y-axis) with increasingly loose constraints on the permitted delay increase (x-axis). Results match the intuition that if a larger delay increase is tolerated, DEFO has more flexibility to distribute traffic on different paths, and can lower the load per link. Interestingly, our experiments also suggest that tolerating 10% or 20% delay increase tends to reduce the maximum link load under 100%, with a maximal link utilization comparable to this obtained with much looser delay constraints (40% or 50%).

Multi-Objective Goals. DEFO accepts input goals with multiple objectives, in which a solution that optimizes all the objectives may not exist. In those cases, *DEFO can build a Pareto front*, i.e., a set of Pareto optimal solutions that cannot be improved in any of the objectives without degrading at least another one. Then, DEFO can either return the Pareto front to the operator or enforce any Pareto optimal solution.

DEFO can also plot the evolution of objective functions over time. This allows operators to use DEFO as a decision support system, e.g., to pick the right tradeoff between optimization time and objectives.

7.4 DEFO Supports Service Chaining

We now evaluate DEFO when used for service chaining applications. We provided DEFO with the goal of minimizing the maximally-loaded link, under the constraint that some demands need to pass through any middlepoint in specified sets. This matches the scenario in which middlepoints in the same set implement the same network service or virtualized function.

In our experience, operators do not deploy service chaining, and are still reluctant to disclose realistic use cases. However, most of them are interested in it. We used DEFO as a what-if analysis tool, to evaluate the degradation of link load optimization in the realistic case in which services are deployed in central network nodes. To provide a benchmark to future approaches, we performed those experiments on inferred and synthetic topologies (adding two new ones to our dataset), and we publicly released them [20].

In each of those experiments, we selected the 5% nodes traversed by the highest number of IGP paths as those deploying services. We then randomly extracted 5% of the demands to pass through those nodes. We performed two groups of experiments. In the first group, the selected demands were forced to pass through any service-enabled node. In the second group, the service-enabled nodes were further split in two service sets, and every selected demand was forced to sequentially pass through any router in the first set and then through any one in the second set.

Services	Min	1st Qu.	Median	3rd Qu.	Max
0	0.24s	9.37s	41.89s	61.06s	140s
1	0.25s	17.21s	41.9s	59.15s	210s
2	0.30s	6.86s	27.09s	59.74s	71.78s

Table 4: DEFO computation time for service chaining goals.

DEFO computed optimized paths always compliant with the input constraints. Unsurprisingly, however, service chaining constraints may prevent DEFO to compute a congestion-free solution. The results on the maximum load after running DEFO are displayed in Fig. 9. In 3 cases (out of 9), no solution can be computed by DEFO when demands needed to pass a sequence of two services. This can be due to both the (i) limited running time, and (ii) the fact that DEFO currently configures acyclic paths. In all the other experiments with one and two services, however, those two factors did not prevent DEFO from optimizing forwarding paths from keeping all link utilization below 100%. In some case, DEFO reduces link loads more in the presence of services than in the absence of them. This is due to the fact that service chaining constraints generally restrict the possible paths to be considered during the optimization; hence, they can increase the likelihood for our optimization heuristics to quickly explore a larger portion of the search space and find a better solution.

Finally, Table 4 shows that service chaining constraints have no statistical effect on DEFO time efficiency.

8. PATH REALIZATION EVALUATION

In this section, we evaluate the effectiveness of our two-layer architecture implementation, both from a scalability (§8.1) and a reactivity (§8.2) point of view.

8.1 Realization Scalability

We first compare different implementations (SR, tunnelling and hop-by-hop ones) of the optimized paths computed by DEFO. For simplicity, we take as reference the MinMaxLoad goal. The entries to be installed on routers for each of the compared implementations are plotted in Fig. 10. For each topology (on the x-axis), the figure reports on y-axis (in logarithmic scale) the ratio between the number of entries needed by a given alternative and those employed by our SR realization.

1.5-10x gain vs end-to-end tunnelling. Provided that a tunnelling technology is supported by all routers, a straightforward implementation of the optimized paths computed by DEFO consists in deploying one (E2E) tunnel per optimized path. Fig. 10 shows that this solution comes with significantly more overhead than SR. This is because end-to-end tunnels scale with the number of paths between each source-destination pair, while SR scales with the number of middlepoints, i.e., factoring all the paths between each middlepoint. Practically, the number of additional entries is between 1.5 to 10

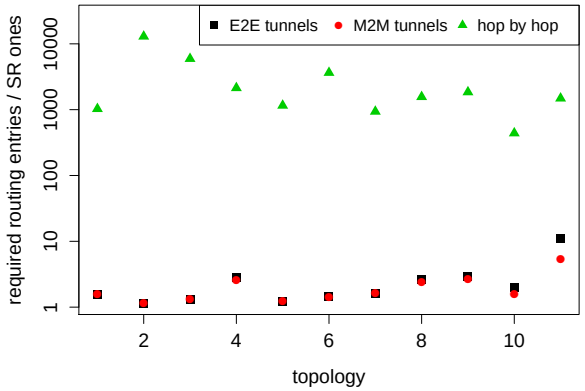

Figure 10: Comparison between different implementations of DEFO optimized paths.

times bigger than those needed by SR. Two topologies ($R2$ and $I5$) make exception, needing about 20% more entries for the E2E tunnels.

1.5-5x gain vs middlepoint to middlepoint tunnelling. A more scalable realization of DEFO paths that can be achieved with tunnelling protocols consists in configuring tunnels between each pair of middlepoints. This way, the total number of tunnels is equal to the sum of the paths between middlepoints in the paths (instead of their combinations). Fig. 10 shows that middlepoint to middlepoint (M2M) tunnels enable to reduce the number of tunnels up to a factor of 2. This, however, means that the entries needed by SR are still between one half and one fifth of M2M tunnel ones.

1,000-10,000x gain vs hop by hop implementation. Finally, optimized path can be implemented with hop-by-hop technologies (e.g., OpenFlow and static routing). In this case, every router must be configured with a forwarding entry for every optimized path crossing it. This leads to an explosion of the number of entries needed to implement DEFO solutions. While this number can be optimized (e.g., by tagging packets), Fig. 10 shows that the gap to fill with respect to the SR (and even with tunnelling solutions) is huge, i.e., among 3 or 4 orders of magnitude in our experiments.

8.2 Reactivity to Network Dynamics

Connectivity needs to be guaranteed and forwarding has to be re-optimized upon a number of events, ranging from network failures (e.g, link failures) to changing demands (e.g., traffic surges). We now describe how components of our architecture react to those changes.

The link-state IGP used at the connectivity layer efficiently re-establishes disrupted paths for all physically-connected node pairs. Indeed, it (i) supports fast local re-routing by relying on per-router decisions [10], and (ii) can recompute new connectivity paths in less than one second even in large networks [25]. Consequently, relying on an IGP mitigates the need for fast reaction to several events. For example, whenever an optimized

path includes a set of middlepoints which are still reachable after a failure, traffic can still flow over that path.

Still, forwarding may need to be quickly re-optimized in some cases. Those cases include failures that disrupt the connectivity between middlepoints in some optimized paths, and congestion triggered by new connectivity paths. Fig. 7 and Table 3 show that DEFO can remove forwarding disruptions by very quickly re-optimizing few demands. This enables to use DEFO for (i) online traffic engineering applications in which forwarding is re-optimized periodically, with a period of few minutes [6, 8]; (ii) pre-compute optimized paths for most likely failures; and (iii) react to failures almost in real-time, in many cases. About failure reaction, we also publicly released a simulation (see [20]) of forwarding optimization with DEFO upon single-link failures on our biggest synthetic topology. This reaction took from few milliseconds to a couple of seconds for the vast majority of the cases. As soon as the new optimized paths are computed, they can be directly installed on routers. Since we rely on SR, *only the configuration of the ingress point of the re-optimized demands needs to changed*, which provides additional architectural support for fast forwarding re-optimization.

9. DISCUSSION

We now discuss limitations of our proposal, and how they can be tackled by slightly extending it.

Per-flow forwarding control. For simplicity, we described our approach assuming source-destination granularity. Our evaluation shows that this level of granularity effectively supports practical goals, especially traffic engineering ones. Nevertheless, our proposal can also support finer-grained control, by leveraging capabilities commonly supported by commercial routers. For example, policy-based routing can be configured on most routers, enabling the specification of their routing behavior in terms of rules on additional packet-header fields than only the destination IP. DEFO can also be easily adapted for finer-grained forwarding optimization, since it models traffic with the generic concept of demands (i.e., flow aggregates).

Additional network goals. DEFO currently supports a large but limited set of network goals. However, it can be extended to deal with additional goals. Consider, for example, the case in which some middleboxes add or drop packets, hence modifying en route the size of demands. Within DEFO, we can adjust our algorithms to split any demand D passing through one of those middleboxes m in two demands D_1 and D_2, such that (i) D_1 goes from the source of D to m and carries the same traffic as D, and (ii) D_2 carries the modified amount of traffic from m to the destination of D. More in general, both the concept of demand and the network parameters considered during the optimization can be generalized. To this end, new constructs may be needed in DEFO DSL. Moreover, new forwarding function algorithms may have to be implemented (e.g., to run Step 4 in Fig. 4). Note that those additions would preserve the support for all forwarding functions in Fig. 2, thanks to their implementation as independent CP constraints.

10. RELATED WORK

To the best of our knowledge, DEFO's ability to solve classic, refined and service chaining goals under a common framework, at scale and efficiently, is unmatched by previous research approaches. Moreover, prior techniques typically require major changes in problem formulation and internal algorithms even only to support an additional constraint type (e.g., node avoidance).

Most previous works focused on classic traffic engineering problems in ISP networks [44, 45]. Two main approaches have emerged over the years. The first one consists in tuning IGP weights [13, 46] according to traffic demands. Network operators use commercial tools for basic IGP weight tuning [38, 47]. We compared DEFO against one of these tools [38] in §7. More sophisticated weight optimizations, based on additional routing entries [48] or uneven traffic splitting [49, 50], have also been explored in the literature. However, adding routing entries affects scalability, and arbitrary traffic splitting is hard to enforce in practice. Our experimental results (§7) show that DEFO can achieve close-to-optimal optimizations with few additional entries and even load splitting. The second approach is to rely on RSVP-TE [51]. It allows to create end-to-end tunnels to forward flows over any network path. RSVP-TE tunnels can be configured in a distributed or centralized mode. In the distributed mode, tunnels are dynamically recomputed by ingress routers to match traffic patterns or topological changes [52]. This mode is prone to operational issues [53] such as routing instability and latency inflation [54]. In the centralized mode, a central component (PCE or the network operator) explicitly specifies the RSVP-TE tunnels to be installed. Algorithms have been proposed to optimize tunnel positioning [15, 55], and some have been implemented in commercial tools [38, 47]. We compared DEFO against distributed and centralized RSVP-TE in §7.

A few contributions looked at refined traffic engineering goals. Notably, FUBAR [56] tackles the problem of maximizing utility of flow aggregates, defined as a specific composition of load and delay metrics. DEFO can solve this problem (see §3) and many more variants of it with arbitrary compositions of delay, load and service chaining constraints and objectives. As partial outcome of this work, we also contributed to support some refined goals in the WAE controller [42] (see §7).

Online traffic engineering has been the target of recent research. Several contributions [6, 8, 57, 58] proposed different techniques to re-optimize forwarding periodically, i.e., every 5-10 minutes. Our results (§7-8) show that DEFO (which computes paths in seconds)

and our network realization (quick to converge on a new routing state) can be used for online traffic engineering.

Finally, some SDN works [1, 45] target individual problems that can be solved by DEFO, including service chaining. For example, [59, 60] provide architectures and optimization frameworks to forward traffic through middleboxes. However, previous SDN proposals do not consider stringent robustness and high scalability requirements of carrier-grade networks. Our evaluation shows that both path computation algorithms (e.g., LP- or MILP-based) and network protocols (OpenFlow) used in previous works may not easily match those requirements. In comparison, hybrid SDN architectures, running both IGP and an SDN protocol, have better chances to fit carrier-grade networks. Our proposal can be seen as an integrated hybrid SDN architecture according to the classification in [61]. As such, it enriches previous efforts in the area (e.g., [6, 12, 30, 62]) with an original architecture based on segment routing and a multi-purpose declarative controller. Note that DEFO can be used as a translator from abstract goals to forwarding paths in protocols different from SR (see, e.g., §8.1). We plan to compare our SR-based implementation with recent proposals for implementing forwarding paths (like Fibbing [30]) in future work.

11. CONCLUSIONS

This paper presents a proposal to realize appealing promises of SDN, like declarative and expressive management, in carrier-grade networks. To accommodate specific needs of those networks, e.g., robustness and scalability, our proposal is based on a two-layer architecture, separating connectivity and optimization tasks. Our centralized forwarding optimizer, DEFO, exposes a declarative interface enabling operators to naturally define a variety of high-level goals. Given a goal, DEFO automatically computes optimized paths that overwrite the connectivity ones used by default. We proposed and implemented a heuristic which makes this computation effective and efficient (from few hundreds of milliseconds to less than one minute on real ISP networks). Our evaluation also shows that the overhead of DEFO optimizations is very limited (few optimized paths, each implemented with at most two node segments).

We finally compared different realizations of our architecture, providing a quantitative analysis of the trade-offs achieved by each of them. Among the considered realizations (including MPLS and OpenFlow), the combination of IGP and segment routing currently provides the best results in terms of robustness and scalability.

We contributed to implement most practical DEFO's capabilities in a commercial controller [42]. In future work, we plan to fully integrate DEFO flexibility in it.

Acknowledgements

We are grateful to SIGCOMM anonymous reviewers and our shepherd, Ming Zhang, for insightful comments. This work has been partially supported by ARC grant 13/18-054 from Communauté française de Belgique. Pierre Francois is fully funded by Cisco Systems.

12. REFERENCES

[1] D. Kreutz et al., "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," *ACM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[3] C. Filsfils *et al.*, "Segment Routing Architecture," Internet draft, 2014.

[4] M. Casado et al., "Rethinking enterprise network control," *Trans. Netw.*, vol. 17, no. 4, pp. 1270–1283, 2009.

[5] M. Al-Fares *et al.*, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *NSDI*, 2010.

[6] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *SIGCOMM*, 2013.

[7] S. Kandula et al., "Calendaring for Wide Area Networks," in *SIGCOMM*, 2014.

[8] C-Y Hong et al., "Achieving High Utilization with Software-driven WAN," in *SIGCOMM*, 2013.

[9] P. Quinn and T. Nadeau, "Service function chaining problem statement," 2014.

[10] C. Filsfils et al., "Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks," RFC 6571, 2012.

[11] R. Krishnan et al., "Mechanisms for Optimizing LAG/ECMP Component Link Utilization in Networks," Internet Draft, 2014.

[12] O. Tilmans and S. Vissicchio, "IGP-as-a-Backup for Robust SDN Networks," in *CNSM*, 2014.

[13] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *INFOCOM*, 2000.

[14] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[15] A. Elwalid *et al.*, "Mate: Mpls adaptive traffic engineering," in *INFOCOM*, 2001.

[16] D. Ghosh, *DSLs in action*. Manning Publications, 2010.

[17] P. Schaus and R. Hartert, "Multi-objective large neighborhood search," in *CP*, 2013.

[18] P. B. Godfrey, S. Shenker, and I. Stoica, "Pathlet routing," in *HotNets*, 2008.

[19] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *SIGCOMM*, 2009.

[20] R. Hartert, "DEFO Web Site," sites.uclouvain.be/defo.

[21] A. Raj and O. C. Ibe, "A survey of IP and multiprotocol label switching fast reroute schemes," *Computer Networks*, vol. 51, no. 8, pp.

1882–1907, 2007.
[22] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *CP*, 1998.
[23] P. Laborie and D. Godard, "Self-adapting large neighborhood search: Application to single-mode scheduling problems," in *MISTA*, 2007.
[24] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.
[25] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving Sub-second IGP Convergence in Large IP Networks," *ACM CCR*, vol. 35, no. 3, 2005.
[26] C. Filsfils *et al.*, "Segment Routing with MPLS data plane," Internet draft, 2014.
[27] S. Previdi *et al.*, "IPv6 Segment Routing Header (SRH)," Internet draft, 2014.
[28] OscaR Team, "OscaR: Scala in OR," 2012, `https://bitbucket.org/oscarlib/oscar`.
[29] A. Shaikh and A. Greenberg, "OSPF Monitoring: Architecture, Design and Deployment Experience," in *NSDI*, 2004.
[30] S. Vissicchio *et al.*, "Central control over distributed routing," in *SIGCOMM*, 2015.
[31] A. Sharafat *et al.*, "MPLS-TE and MPLS VPNS with Openflow," in *SIGCOMM*, 2011.
[32] R. Enns *et al.*, "Network Configuration Protocol (NETCONF)," RFC 6241, 2011.
[33] P. Marques *et al.*, "Dissemination of Flow Specification Rules," RFC 5575, 2009.
[34] S. Sivabalan *et al.*, "PCEP Extensions for Segment Routing," Internet Draft, 2014.
[35] N. Spring *et al.*, "Measuring isp topologies with rocketfuel," *Trans. Netw.*, vol. 12, no. 1, 2004.
[36] B. Quoitin *et al.*, "IGen: Generation of router-level Internet topologies through network design heuristics," in *ITC*, 2009.
[37] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *ACM CCR*, vol. 35, no. 5, pp. 93–96, 2005.
[38] Cisco, "Planning and Designing Networks with the Cisco MATE Portfolio," white paper, 2013.
[39] T. Cormen *et al.*, *Introduction to Algorithms*. McGraw-Hill, 2001.
[40] L. Vanbever *et al.*, "When the cure is worse than the disease: The impact of graceful IGP operations on BGP," in *INFOCOM*, 2013.
[41] R. Teixeira *et al.*, "Network sensitivity to hot-potato disruptions," in *SIGCOMM*, 2004.
[42] Cisco, "Cisco WAN Automation Engine: Greater Traffic and Bandwidth Awareness for Easier Programmability," white paper, 2015.
[43] C. Filsfils, "Segment routing: update and future evolution," MPLS/SDN world congress, 2014.
[44] N. Wang *et al.*, "An overview of routing optimization for internet traffic engineering," *Commun. Surveys Tuts.*, vol. 10, pp. 36–56, 2008.
[45] I. Akyildiz *et al.*, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.
[46] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, 2002.
[47] Juniper, "WANDL IP/MPLSView," http://www.juniper.net/assets/us/en/local/pdf/datasheets/1000500-en.pdf.
[48] A. Sridharan *et al.*, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *Trans. Netw.*, vol. 13, no. 2, 2005.
[49] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *INFOCOM*, 2001.
[50] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *Trans. Netw.*, vol. 19, no. 6, pp. 1717–1730, 2011.
[51] I. Minei and J. Luceck, *MPLS-Enabled Applications*. Wiley, 2005.
[52] S. Dasgupta *et al.*, "Dynamic traffic engineering for mixed traffic on international networks," *Computer Networks*, vol. 52, no. 11, Aug. 2008.
[53] R. Steenbergen, "Mpls autobandwidth," RIPE 64.
[54] A. Pathak *et al.*, "Latency inflation with MPLS-based traffic engineering," in *IMC*, 2011.
[55] M. Kodialam *et al.*, "Oblivious routing of highly variable traffic in service overlays and IP backbones," *Trans. Netw.*, vol. 17, pp. 459–472, 2009.
[56] N. Gvozdiev, B. Karp, and M. Handley, "FUBAR: Flow Utility Based Routing," in *HotNets*, 2014.
[57] H. Wang *et al.*, "COPE: traffic engineering in dynamic networks," in *SIGCOMM*, 2006.
[58] S. Kandula *et al.*, "Walking the tightrope: responsive yet stable traffic engineering," in *SIGCOMM*, 2005.
[59] D. Joseph *et al.*, "A Policy-aware Switching Layer for Data Centers," in *SIGCOMM*, 2008.
[60] Z. Qazi *et al.*, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *SIGCOMM*, 2013.
[61] S. Vissicchio *et al.*, "Opportunities and Research Challenges of Hybrid Software Defined Networks," *ACM CCR*, vol. 44, no. 2, 2014.
[62] S. Agarwal *et al.*, "Traffic Engineering in Software Defined Networks," in *INFOCOM*, 2013.