A Simulation-based Optimization Framework for Online Adaptation of Networks

Stefan Herrnleben[1], Johannes Grohmann[1], Piotr Rygielski[2], Veronika Lesch[1], Christian Krupitzer[1], and Samuel Kounev[1]

[1] University of Würzburg, Würzburg, Germany
{firstname.lastname}@uni-wuerzburg.de
[2] D4L data4life gGmbH, Potsdam, Potsdam, Germany
piotr.rygielski@data4life.care

Abstract. Today's data centers face continuous changes, including deployed services, growing complexity, and increasing performance requirements. Customers expect not only round-the-clock availability of the hosted services but also high responsiveness. Besides optimizing software architectures and deployments, networks have to be adapted to handle the changing and volatile demands. Approaches from self-adaptive systems can optimize data center networks to continuously meet Service Level Agreements (SLAs) between data center operators and customers. However, existing approaches focus only on specific objectives like topology design, power optimization, or traffic engineering.
In this paper, we present an extensible framework that analyzes networks using different types of simulation and adapts them subject to multiple objectives using various adaptation techniques. Analyzing each suggested adaptation ensures that the network continuously meets the performance requirements and SLAs. We evaluate our framework w.r.t. in finding Pareto-optimal solutions considering a multi-dimensional cost model, and scalability on a typical data center network. The evaluation shows that our approach detects the bottlenecks and the violated SLAs correctly, outputs valid and cost-optimal adaptations, and keeps the runtime for the adaptation process constant even with increasing network size and an increasing number of alternative configurations.

Keywords: network, modeling, simulation, self-adaptation, optimization

1 Introduction

Modern technologies like cloud computing enable dynamic and flexible allocation of computing and storage resources without requiring the cumbersome booking of dedicated resources in advance [35]. Such technologies enable the dynamic instantiation of new services and create new opportunities for novel applications. It also supports flexible on-demand scaling of services depending on the workload intensity.

In addition to computing and storage resources, networks also have to be adapted and scaled accordingly to handle the variation of traffic. The performance requirements of a network, like bandwidth or latency, are usually specified in Service Level Agreements (SLAs). Different approaches exist to manage resources and traffic flows in network infrastructures. Examples include Virtual Local Area Networks (VLANs) that support isolation and prioritization of traffic, dynamic routing protocols for determining shortest paths, and Software-defined Networking (SDN) [24] that enables fine granular control of data traffic on a per traffic flow basis. Besides these techniques, sophisticated Quality of Service (QoS) configuration parameters, license upgrades, or hardware changes are valid adaptation approaches for network infrastructures. Configuration of these comprehensive configuration options could be a challenging task with increasing network size and complexity.

Network designers and performance experts can make an educated guess for network optimizations to meet the QoS criteria, but this raises several issues. First, with increasing size and complexity of networks, it is challenging to suggest SLA compliant and cost-optimal solutions, even for experts, and especially to validate them before applying them to a real network. Second, educated guesses might also require hypothetical what-if analyses, which require dedicated measurements or monitoring data. Third, relying on human intervention is generally undesirable due to limitations on availability and time-to-result. Existing recommendation tools for network optimizations either only support a limited set of adaptation operations, cover only specific objectives, or do not verify the adaptations, resulting in a trial and error approach [14, 43, 3].

In this paper, we propose a framework for network optimization that integrates a network simulation for bottleneck and SLA violation detection and suggests verified network adaptations. A network model is analyzed under a specified workload to obtain information about the performance characteristics of links, interfaces, switches, and nodes. Based on the analysis results, our framework detects bottlenecks and the resulting SLA violations. An iterative MAPE-K adaptation control loop [2] resolves these bottlenecks using objective-oriented strategies and adaptation tactics that ensure that the network complies with current SLAs under given technical constraints. Multiple adaptation tactics are triggered in parallel by a Branch & Bound algorithm [30] to determine the diversity of possible solutions, leveraging different types of adaptations like configuration changes, hardware replacements, or rerouting of flows. Before applying a solution to a real-world network, a simulation verifies the suggested adaptations on the modeled system. The returned solutions all lie on the Pareto-front, i.e., they are all cost-optimal.

The proposed approach enables data center networks to autonomically react and adapt to environmental changes for compliance with a given set of predefined SLAs. The approach furthermore considers costs in multiple dimensions and autonomically selects the cost-optimal solutions. Software-based adaptations, e.g., configuration changes or traffic rerouting, can be applied automati-

cally, without the need for human intervention and before the actual SLA violation occurs.

The remainder of this work is structured as follows. Section 2 discusses related work. Section 3 presents an overview of our framework, the workflow of the used models. Section 4 describes our multi-objective MAPE-K-based adaptation process, which our adaptation framework uses. Section 5 presents and discusses the evaluation results, including two data center network scenarios for effectiveness and scalability investigation. Section 6 addresses aspects of future work. Lastly, Section 7 concludes this paper.

2 Related Work

For single-objective problems, a system can be optimized towards the single best solution, but for multi-objective problems (MOPs), the optimization has to deal with several potentially conflicting goals. Consequently, the optimization ends with no clear optimal solution, but the Pareto-front [7] represents a multitude of so-called Pareto-optimal solutions. To handle the optimization of MOPs, several generic approaches exist, such as evolutionary algorithms [13] (e.g., NSGA-II [16] or SPEA2 [26]), scatter search [33], particle swarm optimizers [39], or ant colony optimization [18]. Several frameworks provide generic applicable implementations of optimization techniques, e.g., jMetal [19], Opt4J [31], and ECJ [32] to mention only a few. A recent study on optimization in the field of self-adaptive systems [21] found that self-adaptive systems often integrate generic optimization techniques by customizing the representation of the required information for specific modeling of a specific application, cf. Rainbow [22], hence, reducing the applicability of such an approach. As MOP modeling is an integral part of the optimization, we also pursue domain-specific modeling.

Consequently, the remainder of this section focuses on related work in the two areas of (i) network performance modeling and simulation and (ii) network optimization.

2.1 Network Performance Modeling and Simulation

Existing approaches for network modeling and network performance simulation do not completely integrate both aspects of modeling and simulation. They can be categorized as (i) approaches that focus on simulating the network performance directly and (ii) approaches that generate a model of the network and apply simulation-based evaluation.

Tools from the first category, focusing on simulation, are amongst others, OPNET [12], OpenWNS [8], Georgia Tech Network Simulator (GTNetS) [40], and IKR SimLib [44]. The scope of these tools is simulating large-scale topologies and more complex systems. The widely used approaches OMNeT++ [47], ns-3 [11], CloudSim [10], and DNI [41] belong to the second category and support modeling and simulation.

We chose the Descartes Network Infrastructures modeling language (DNI) [41] as a basis for our work as it provides a fine-grained description of networks, including a detailed performance specification of network interfaces. The generic modeling approach allows the definition of custom protocol stacks and an autonomous traffic pattern extraction from real networks. The modular design of DNI supports multiple exchangeable simulation tools like OMNeT++ or SimQPN [28]. SimQPN is a discrete event simulator based on stochastic modeling and analysis of Queueing Petri Nets (QPNs) [4] and is already used in other modeling languages for self-adaptive software systems, for modeling of Java EE applications [27], and message-oriented event-driven systems [42]. We use and extend DNI to specify the network structure, its configuration, the traffic patterns, and the adaptation points.

2.2 Network Optimization

In the literature, several network optimization approaches focus on the design-time optimization of network topology [14, 37, 1], energy efficiency [43], or network virtualization [38]. However, as our approach deals with the optimization of networks at runtime, we focus on runtime approaches. The identified related approaches can be classified into three categories: (i) service placement, (ii) power optimization, and (iii) traffic optimization.

The service placement approach optimizes the number of deployed services, for example, by using linear programming [9]. Approaches optimizing power consumption, for example, by turning off as many unneeded network components as possible, use greedy bin packing algorithms or linear programming for rerouting and placement optimization [20, 49, 45]. The third category of traffic optimization enhances the bandwidth, traffic flow distribution, or link utilization by using Markov Chain approximations [25], bin packing heuristics [6], CPLEX [46], or linear programming [5].

Although many of the approaches validate their suggested changes, these validations are often based on simple analytical models that are only suitable for a small range of applications. Existing simulative approaches are often too complex, which might violate runtime constraints for complex scenarios. However, a validation of the adaptation on the model level is useful; otherwise, the changes would have to be applied to a real network to obtain further information about additional bottlenecks and to determine whether all objectives are fulfilled. Our approach uses a feedback loop combined with network simulation to evaluate and improve the optimization plan without the need to execute the changes on a running network. Furthermore, existing approaches focus on specific objectives while our approach provides multiple adaptation tactics to cover possible alternative approaches for ensuring the fulfillment of multiple objectives. These tactics, as well as further objectives, can be easily extended.

3 Approach

This section provides an overview of our network adaptation approach, including the input models, the applied concepts MAPE-K and Branch & Bound, the underlying adaptation techniques, and the output models. Our adaptation approach pursues the following objectives and quality criteria:

- Bottleneck detection: Bottlenecks are identified and localized through network analysis.
- Model-based: The adaptation process integrates a model of the network, which enables analysis without influencing the real network.
- Online network adaptation: A network and its current state can be monitored and adapted in an autonomic manner at runtime.
- Validation: The suggested solutions are validated based on an analysis for each adaptation.
- Multi-objective: The adaptation considers multiple objectives and executes different adaptation tactics with various solution approaches (such as rerouting, reconfiguration, or hardware change).
- Efficient: Bounding mechanisms filter non-optimal branches to reduce the number of analyses.
- Extendable: New adaptation tactics can be easily added.
- Multiple solutions: Pareto-optimal solutions taking into account the multi-dimensional costs model.

The network optimization requires a *network modeling language* to define the network structure, the network configuration, and the workload. To analyze the network, i.e., to determine the utilization and detect the bottlenecks, a *network analysis* is used, which has to support the chosen network modeling language. Replacing the network modeling language and the analysis is easy due to the design of our network optimization approach. Depending on the selected network modeling language, both simulative and analytical methods can be used.

3.1 Input Models

The adaptation process requires several input models, which are depicted on top of Figure 1 and are introduced in the following. We refer the interested reader to [23] for a detailed specification of the respective meta-models.

The *network model* describes the network structure, its configuration, and the current workload. The *structure* captures the topology of the network, which includes physical and virtual nodes, the connections between them, the network interfaces, and the performance descriptions of all entities. The *configuration* defines the initial routes for the traffic flows as well as the used protocols and protocol stacks. The *workload* describes all flows between the nodes and specifies the type of flow as well as their source, destination, size, and temporal behavior. For the network model, we chose the *Descartes Network Infrastructures Modeling Language* (DNI) [41]. Existing adapters allow simulating a DNI network

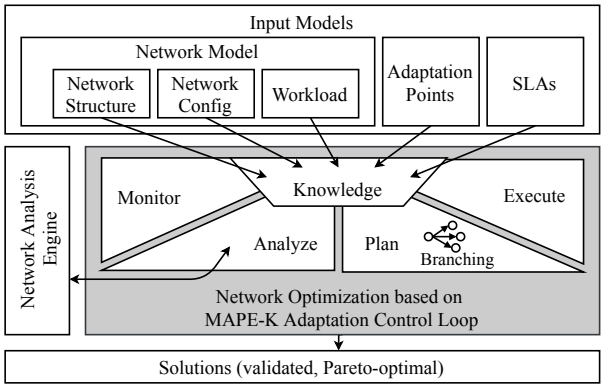

Fig. 1. Abstract illustration of network optimization algorithm.

model by OMNeT++ [48] or SimQPN [28]. Although our current implementation bases on DNI, the framework's modularly design allows integrating other network modeling languages. The interchangeability also applies to the analysis component, i.e., simulation or analytical method.

The *adaptation points* define the valid alternative components such as nodes, links, and components within the network model. This model also specifies the alternative configuration parameters, performance specifications and settings for the components. The adaptation points further describe the costs for component replacement or configuration changes. The multi-dimensional cost model can capture several cost aspects like downtime, handling time, or the total cost of ownership (TCO).

The *Service Level Agreements (SLAs)* contain the agreed performance characteristics (e.g., minimum bandwidth guarantees) between the customers and the network operator. The SLAs' objectives are bound to a specific link, switch, network interface, flow, or combination of them. The goal of network optimization is to fulfill all objectives.

3.2 Adaptation Process: MAPE-K

There are two challenges when eliminating a bottleneck in a network: On the one hand, the effect of a configuration change or a replacement cannot be fully predicted. On the other hand, eliminating a bottleneck can result in an additional bottleneck at another location, so-called bottleneck shifting. Through the integration of a *MAPE-K control loop*, the effects of changes can already be detected on the model, without being applied to the real network, and if necessary improved in further iterations. Therefore, the proposed control loop in Figure 1 solves the aforementioned challenges.

The four phases of the MAPE-K adaptation control loop **M**onitor, **A**nalyze, **P**lan, and **E**xecute as well as the **K**nowledge base are integrated by our approach as follows: The *monitoring phase* uses the network model as an input parameter

to capture the current network structure, configuration, and workload. These data are passed to the *analysis phase*, which triggers the network analysis, depicted on the left side of Figure 1. The analysis determines the used bandwidth, utilization, latency, and packet loss of nodes, including their components such as backplanes and interfaces, as well as links. These performance characteristics are used to localize the bottlenecks and to identify the violated objectives. The analysis result and the violated objectives are passed to the subsequent *planning phase* to determine operations that could eliminate the bottlenecks. The possible adaptation operations are derived from the domain knowledge and the provided adaptation points. If there are several, more performant options available for one parameter, e.g., alternative backplane speeds, the cost-optimal solution is chosen first. Branching supports the exploration of several solutions. The suggested adaptation operations are integrated into an adaptation plan, which is passed to the execution phase. The *execution phase* applies the scheduled adaptation operations to the network model so that the changed network can be analyzed again by the loop in the next iteration. If all bottlenecks have been eliminated by the scheduled operations, the proposed adapted network model is valid and is output. Otherwise, further adaptations will be applied in subsequent iterations. The *knowledge base* contains the defined SLAs and the adaptation points, as well as other user constraints and configuration parameters. As depicted in Figure 1, the knowledge base is accessible from all MAPE phases.

After one or several possible solutions are found, these proposed operations can be executed on a real-world network. If the adaptation operations can be executed in an automatic manner, such as changing configuration parameters or rerouting, they can be passed to a network management software API or, in case of *Software-defined Network (SDN)* to an SDN controller [24]. Manual changes can be forwarded to network operators.

The whole adaption process can be triggered continuously or event-based to react to changes in workloads, network structure, or configuration. Using forecasted workloads [50] allows a proactive adaptation or what-if analysis.

3.3 Adaptation Process: Branch & Bound

Sometimes it cannot be predicted if a particular operation will eliminate the bottleneck. Due to the multiple cost dimensions, there may be several cost-optimal solutions. To address this challenge, our adaptation framework employs a *Branch & Bound* algorithm to track different adaptation operations in parallel [36]. This algorithm explores different solution candidates if it is unclear which solution strategy will be most cost-optimal. For example, to increase the bandwidth, the algorithm will explore a branch of upgrading a network interface and explore another branch to replace the whole switch. Solutions that do not improve the system or violate constraints are bounded to limit the number of tracked branches.

3.4 Output Models

Adaptation plan models are used to describe the required adaptation on a network. Depending on the number of valid solutions, one or a set of adaptation plans are output. They represent the delta between the original network state and the desired one. An adaptation plan defines the replacement of entities like entire nodes or just individual components, the change of configuration parameters, or the modification of routes in a descriptive manner.

Depending on the validation status, different terminologies are used for the solutions identified in our network optimization algorithm:

- *Solution candidates* are network models that have not yet been validated w.r.t. SLA compliance. Further adaptations may be necessary on the model or it may be discarded later.
- *Solutions* are network models for resolving the SLA violations. The solutions are validated by analysis while at the same time ensuring that they do not violate filter criteria (bounding).
- *Cost optimal solutions* are solutions that are cost-optimal in terms of representing the Pareto-front.

The adaptation framework outputs a set of solutions with the following properties. First, the solution resolves all SLA violations, even if created by bottleneck shifting (verified by analysis). Second, they are cost-optimal and represent the Pareto-front concerning the multi-dimensional cost model.

4 Adaptation Process

This section describes the adaptation process of the multi-objective network optimization, based on the MAPE-K adaptation control loop with Branch & Bound (cf. Section 3.3). The process of the adaptation is separated into multiple modules (see Figure 2). This section describes each of these modules.

4.1 Analysis

The purpose of the analysis module is to analyze the network model, including the workload to determine, among other things, the utilization and throughput of components as well as packet loss and waiting times. To take the previously scheduled adaptations into account, the adaptation plan, which is empty in the first iteration, is applied to the network model. This adapted network model is analyzed through simulative or analytical methods, depending on the used network modeling language. The analysis is invoked as an external module, as depicted in Figure 2, which is not in the scope of our network optimization. Instead of the current workload, predicted workloads can be used for proactive adaptation of the network for adjusting to future demands. Customized network models for what-if analysis, e.g., changes in the network structure or additional customers, are also supported. The result of the analysis is passed to the underlying SLA violation detection.

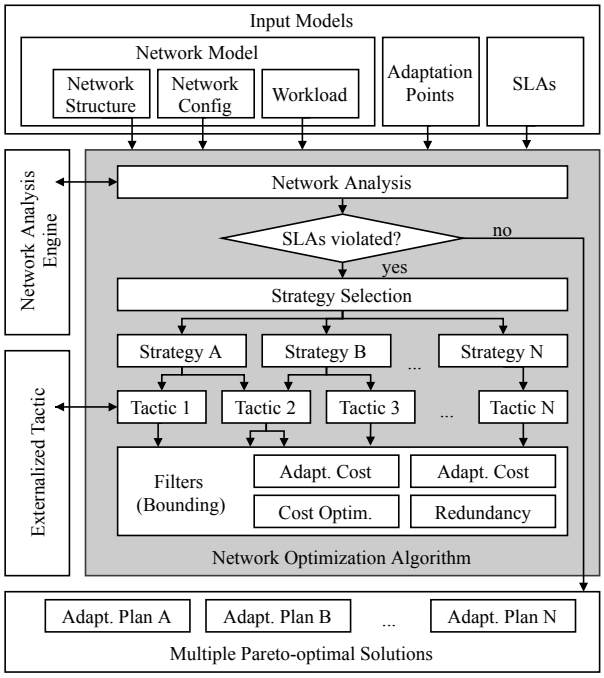

Fig. 2. Network optimization algorithm with the steps analysis, SLA violation detection, branching, and bounding.

4.2 SLA Violation Detection

The SLA violation detection module receives the results from the network analysis and uses the provided SLAs to determine if objectives are violated. If all objectives are fulfilled, the process outputs the current adaptation plan as no further adaptations are necessary. If all objectives are already fulfilled in the first iteration, this represents the trivial case in which the adaptation plan is empty, and the network complies with all SLAs without the need for adaptations. If the module detects an SLA violation, the objective, and the associated bottleneck are marked for adaptation and are passed to the next module, the strategy selection (cf. Figure 2).

4.3 Branching through Strategies and Tactics

The strategy selector chooses a high-level goal, i.e., an adaptation strategy, by evaluating the violated objectives identified in the SLA violation detection. If multiple strategies can resolve a bottleneck, all of them are started independently by branching, as shown in Figure 2. For each of these strategies, several adaptation tactics exist. Those are specific algorithms to reach the goal. For example, a bottleneck on an overloaded link can be handled either by rerouting

one of the respective flows or increasing the link capacity. Several tactics can be run in parallel and independent from each other using our branching approach.

4.4 Adaptation Tactics

An adaptation tactic is a stateless, black-box algorithm to discover suitable network adaptations. Hence, it is self-contained and does not require any historical knowledge about the system. It can be implemented as part of the adaptation framework or as an externalized module, as shown in Figure 2. Each tactic has its scope, such as optimizing interface performance, replacing links, or optimizing routes. Through the network-specific domain knowledge, only tactics that can remedy the bottlenecks are triggered. This knowledge characterizes our domain-specific concept, which is not possible in more generic approaches.

Our adaptation framework currently supports ten tactics focusing on performance optimization and rerouting. Developers can easily extend them by improving/adjusting the adaptation-related to an existing objective, or adding new adaptations like energy optimization or addressing security concerns. In combination with the iterative evaluation of the suggested adaptations, this helps to avoid stability issues that might arise through uncoordinated adaptations applied directly to the network, such as oscillations, overshooting, or damping of adaptation effects [29]. Due to space constraints, we only describe the tactics for alternative routes and interface upgrades in the following. Other tactics can be found in the implementation given in our Git repository[3].

Alternative Routes. Rerouting is particularly useful in data centers where alternative paths exist that are intended for load-balancing purposes. In contrast to existing flow optimization algorithms, our approach tries to minimize the number of changed routes. We use the Dijkstra algorithm to search for alternative routes [17]. The Dijkstra algorithm is a greedy algorithm and searches in its native version the shortest path between two nodes. Although a network is already a graph with nodes and edges, the Dijkstra algorithm cannot be applied directly on the network, as it does not consider the capacities of interfaces and nodes. To take this into account, each entity is transformed into one input and one output graph node in a preprocessing step. The algorithm of the rerouting tactic removes flows that should not be rerouted from the graph, and the capacities of the other links are reduced by the bandwidths of these flows to avoid a bottleneck shifting. The Dijkstra shortest-path algorithm is then executed for every flow on the overloaded link, and one or multiple candidate solutions are passed to the next module.

Interface Upgrade. This tactic addresses the replacement of network interfaces. The upgrade of an interface can either be hardware-based for physical nodes or software-based for virtual machines or SDN. Examples for physical replacements are *small form-factor pluggables (SFPs)*, which are frequently used in network devices to connect fiber optic cables. This tactic leverages the bottleneck determined by the SLA violation detection, to identify the interface, which

[3] https://gitlab2.informatik.uni-wuerzburg.de/descartes/dni-adaptation

should be upgraded. A separate adaptation process is initiated for each interface representing a bottleneck, identified by the SLA violation detection. The adaptation points model defines the valid alternative interfaces and configuration parameters that are compatible with the containing device. As a replacement, the most cost-effective, performance-optimizing interface is selected first. If the performance optimization is not sufficient yet, further optimization occurs in the next iteration of the MAPE-K adaptation control loop. Additionally, if there is no uniquely cost-optimal or performance-optimal solution, the tactic creates a branch for each possibility and returns all potential solutions.

4.5 Bounding through Filters

The filter module receives the adaptation plans from the tactics, which are all tracked in independent branches. Sometimes branches are generated which are redundant, exceed cost limitations, are not cost-optimal, or violate user constraints. To save computation time and increase the efficiency of the adaptation process, the filter module bounds (i.e., cuts) such branches, as intended by the Branch & Bound algorithm. Figure 2 shows the four stages of this module. In the following, this section briefly describes them.

Adaptation Count Bounding. To avoid a complete redesign of the network which usually increases downtime and risk of errors, this filter removes branches that exceed the specified limit of adaptation operations to ensure that network optimization is applied with manageable effort.

Cost Constraint Bounding. This filter removes branches that exceed a specified limit for at least one cost dimension. Since the tactics select the most cost-effective adaptations, a branch that exceeds the cost limit can be safely removed, as the costs of this branch can only increase in subsequent iterations.

Cost Optimization Bounding. If multiple branches are tracked in parallel, some branches can become irrelevant once cost-optimal solutions are found on other branches. This filter module bounds branches exceeding the costs in all dimensions compared to an already found solution.

Redundancy Eliminator. By branching several times in repeated iterations, different branches can result in the same adaptation operations. Since identical branches lead to the same results in further iterations, this filter discards branches with redundant adaptation plans.

Adaptation plans, which are not bounded in the filter module, are passed to the next iteration of the adaptation process, as shown in Figure 2. This is repeated until a valid solution is discovered, no further possible adaptations are found, or the solution is bounded later.

4.6 Solutions

The adaptation process discovers all cost-optimal (Pareto-optimal) solutions. To apply the computed adaptations, these solutions do not represent an adapted network model; instead, solutions simply consist of adaptation plans, i.e., a list

of adaptation operations to be performed on the network. This makes it easier to interpret and to apply in practice. By applying these adaptation operations on the network model, the resulting network model can be generated. The solutions have the following three characteristics: (i) Every solution is validated through analysis and fulfills the SLA. (ii) Every solution is cost-optimal, and the set of all solutions represents the Pareto-front. (iii) Every solution does not exceed the specified amount of adaptation operations and does not exceed the defined cost limits in any dimension.

5 Evaluation

We evaluated our framework w.r.t. different qualitative and quantitative aspects. Section 5.1 validates our network adaptation process that the found *Pareto-optimal* solutions resolve the bottlenecks, i.e., no further SLA violations exist, and are cost-optimal. Section 5.2 presents an analysis of the scalability of our approach within an example network model with cascaded adaptation operations. Section 5.3 summarizes the results of the evaluation and discusses their applicability.

All experiments are executed on a notebook with an Intel i7-7500U CPU with 2.7 GHz and 16 GB RAM. The used operating system is Windows 10 64-Bit, running an OpenJDK 11.0.2.

5.1 Pareto-optimality and Performance

The adaptation framework outputs cost-optimal solutions representing the Pareto-front [15, 19]. This is particularly important since the multi-dimensional cost model can provide several most favorable solutions for different cost dimensions. Depending on a weighting function, the most appropriate solution is selected for applying it to the respective real-world network. Our framework focuses only on cost-optimized solutions through our objective-oriented approach, the sophisticated bounding mechanism, and several filters. This is especially important since every further solution candidate is analyzed and leads to a longer duration for the overall process.

In this evaluation, we investigate the aspects of: (i) finding all Pareto-optimal solutions and (ii) the performance gain of our optimized algorithm compared to a brute-force approach. For this experiment, we use a small network consisting of 9 nodes, 8 links, and 20 alternative configurations. They possible adaptations are annotated by costs that are partly specified in opposite ways, which means that some are preferred for a low investment and others for short handling time. This leads to several best solutions, depending on the weights of the cost dimensions. We executed the experiment in two different setups. In the first setup, the network is adapted by our optimized approach with objective-oriented strategies, smart selection of alternative configurations, and bounding after introducing the artificial bottleneck. A second setup uses a brute-force approach to explore all possible adaptations and find all valid solutions as a baseline. The use of the

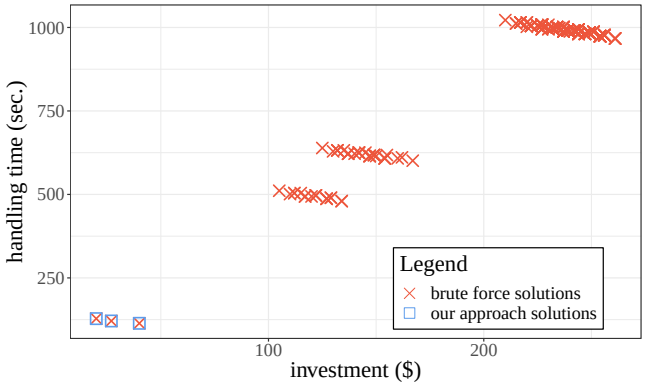

Fig. 3. Pareto front generated by the optimized adaptation process compared to solutions, generated by a brute-force approach. The results of the brute-force approach are depicted as red crosses and the results of our optimized adaptation are depicted as blue squares.

Table 1. Comparison between brute-force and our approach regarding solutions, solution candidates, and runtime.

Observed metric	Brute-force	Optimized approach
cost-optimal solutions	3	3
solutions	231	3
solution candidates	3190	10
overall duration (sec.)	6379	24

brute-force algorithm ensures that all solutions are found, therefore especially including all cost-optimal ones.

Figure 3 depicts all returned solutions from both setups. The x-axis presents the cost dimension of the investment, while the y-axis shows the cost dimension of handling time. As the brute-force algorithm outputs all found solutions – which is a significantly higher number than the number of solutions that our optimization approach outputs – it also contains all cost-optimal solutions. Our approach only returns three solutions. However, it can be concluded from Figure 3 that these are the three cost-optimal solutions.

Table 1 shows a comparison between the brute-force and our optimized approach. We validated in Figure 3 that both approaches return all three cost-optimal solutions. The brute-force approach identified 231 solutions. However, taking the focus on of our cost objective into account, 228 of them are non-optimal solutions, and hence, not useful in practice. As our optimization approach works in a goal-oriented manner and bounds useless branches, it creates only 10 solution candidate models, much less than the brute-force approach with 3190 solution candidates. The number of solution candidates especially affects the algorithm's run-time, since each candidate model has to be analyzed and a

small number of models leads to a shorter overall duration. For that reason, the brute-force approach takes 6379 seconds, and our optimization approach only takes 24 seconds, including the time necessary for the simulative analysis, for calculating all cost-optimal solutions.

5.2 Scalability

Data center networks typically consist of a vast amount of nodes and links. While Section 5.1 only considers a small network topology due to the execution of the time-intensive brute-force run, this section focuses on the scalability aspect of our adaptation algorithm. An addition to the previous section, the selected scenario requires multiple subsequent adaptations to remedy the violated SLAs.

The considered network, depicted in Figure 4, represents a fat-tree topology commonly used in data centers. It consists of two core switches on the top, access and edge switches on the subsequent layers, and at least some servers with VMs. We consider each pair of access and edge switches and two servers with VMs as one block. The core switches are independent of the number of blocks. The initial setup contains one block; more blocks will be added later during the evaluation. As each block consists of 6 nodes and the number of core switches is independent, the number of overall nodes can be derived from the number of blocks as follows:

$$|nodes| = 2 + 6 \cdot |blocks| \tag{1}$$

The notation $|x|$ is taken from graph theory and describes the number of elements x, e.g., $|nodes|$ means the amount of *nodes*. Each block adds 10 links to the network, 6 within the block, and 4 links to the core switches. Therefore the number of links can be derived from the number of blocks as follows:

$$|links| = 10 \cdot |blocks| \tag{2}$$

For each link, six alternative bandwidths are specified in the adaptation points by replacing the cable or reconfiguration. Furthermore, each switch can be replaced by three alternatives. As the number of nodes and links increases with the network size, the number of adaptation points also increases, which can be determined by the following formula:

$$|adaptationpoints| = 6 \cdot |links| + 4 \cdot (|nodes| - 2 \cdot |blocks|) \tag{3}$$

The relationship between the number of blocks, nodes, links, and adaptation points is depicted in the first four columns of Table 2.

Figure 4 illustrates the scenario for the scalability evaluation: `VM1` initiates a transmission of a 20 GB file to `VM2`. Initially the flow is routed over the red path, from `VM1` over `E1-A1-E2` to `VM2`. In the network, the links between `VM1` and `E1` as well as between `E1` and `A1` represent bottlenecks, denoted by the lightning symbol in Figure 4. These bottlenecks may be caused either by links with insufficient bandwidth or through the impacts of other flows. To remedy the bottleneck between `VM1` and `E1`, a link upgrade is the only feasible adaptation

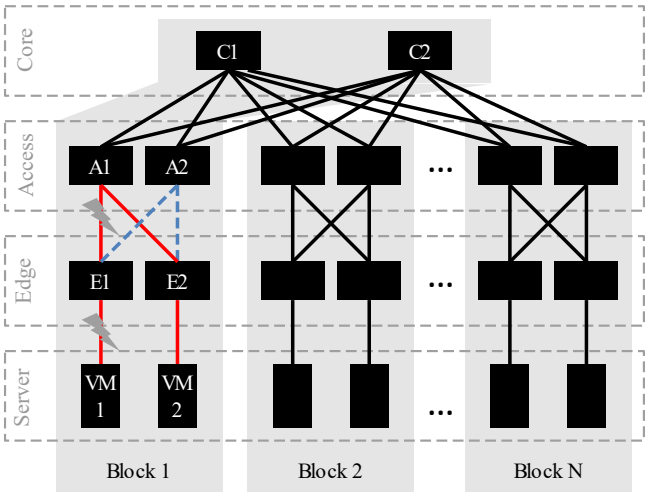

Fig. 4. Network used for the scalability evaluation.

Table 2. Runtime of our adaptation process considering increasing network sizes.

Blocks	Nodes	Links	Adap. points	Bran.	Anal.	Bound. bran.	Sol.	Adaptation process (mean)	(sd)
1	8	10	84	5	4	2	2	0.693 s	0.108 s
5	32	50	388	5	4	2	2	0.695 s	0.120 s
10	62	100	768	5	4	2	2	0.715 s	0.130 s
20	122	200	1528	5	4	2	2	0.547 s	0.145 s
30	182	300	2288	5	4	2	2	0.690 s	0.105 s
40	242	400	3048	5	4	2	2	0.547 s	0.256 s
50	302	500	3808	5	4	2	2	0.792 s	0.446 s

Adap. points = number of adaptation points; Bran. = number of generated branches; Anal. = number of executed analysis, i.e., simulations; Sol. = number of returned solutions

operation defined in the adaptation points. The second bottleneck between `E1` and `A1` can either be resolved by another link upgrade or by bypassing the insufficient link using the blue dashed path from `E1` over `A2` to `E2`. The resulting two possible solutions are: (i) link upgrades from `VM1` to `E1` and from `E1` to `A1`, or (ii) one link upgrade from `VM1` to `E1` and rerouting the flow over the blue dashed path (`E1-A2-E2`). The adaptation framework correctly determines the two solutions. However, finding the solutions is not discussed here, as we focus in this section on the scalability of our approach.

To investigate our approach's scalability, the presented setup is adjusted by a variable number of blocks, each consisting of two access switches, two edge switches, and two servers (cf. Figure 4). The initial network contains one block, and more blocks are added so that the network sizes of 1, 5, 10, 20, 30, 40, and 50 blocks can be examined, as depicted in Table 2. The adaptation of each

setup with a fixed number of blocks was repeated 10 times. The source code of our adaptation framework, the input models, a script for running the scalability experiments, a description of how to run the tests, and the output measurement results can be found on our Git repository [4].

Table 2 shows that independent from the number of blocks, five branches have been generated during each adaptation runs, representing the link upgrade between `VM1` and `E1` (two; one for each cost dimension), the subsequent link upgrade between `E1` and `A1` (two; one per cost dimension), and the parallel rerouting. The constant number of four network analyses results from the initial analysis, the analysis after the first link upgrade between `VM1` and `E1`, and the final analysis of the two solutions. In each of the runs, two branches are bounded by the redundancy eliminator. This number results from the parallel tracked cost-efficient solutions, which finished in identical adaptation plans and therefore, could be eliminated. Table 2 shows that the required time for the adaptation process is constant, independent from the network size and the number of adaptation points, and varies between 0.547 and 0.792 seconds.

5.3 Discussion

In Section 5.1, our adaptation process was compared to an approach determining all solutions via brute-force. We purposefully chose a small example network for this experiment, as the time-consuming brute force approach made the evaluation of bigger networks infeasible. Even with this network size, it becomes clear how important the reduction to the relevant solutions is. Although the chosen brute force solution represents a lower baseline, it shows clearly the applicability of our adaptation approach as it finds all cost-optimal solutions only within a few iterations by objective orientation and domain knowledge.

The evaluation of scalability in Section 5.2 shows that our approach performs well, even on increasing network sizes and an increasing number of adaptation points. Due to the target-oriented approach, the size of the network beyond the bottleneck does not matter, so that the number of branches, analyses, and truncated branches remains constant. This also applies to the adaptation time, which remains constant on a wide-scaled network. The required time for the adaptation process is below 0.8 seconds for all investigated network sizes. In cases of physical changes, which require human hands-on, this time is negligible. Given a self-adaptive autonomic environment, the time for the adaptation is acceptable compared to, e.g., flow rule installation time in SDN switches [34].

The evaluation does not consider the time required to analyze the network. This depends on the used method (simulative or analytical) and the chosen network modeling language. Our ongoing research includes comparing the experiments presented in this paper with results using different network modeling approaches and a variety of different network analysis approaches. These results can be utilized to analyze different modeling approaches and/or to compare the performance of different analysis methods.

[4] https://gitlab2.informatik.uni-wuerzburg.de/descartes/dni-adaptation

6 Future Work

As future work, we aim to extend our framework with tactics concerning the placement and movement of virtual network functions (VNFs) and virtual machines (VMs) of data centers. This enables our framework to actively organize the flow of network traffic in order to optimize network performance. A further direction could be to develop an adapter to connect our optimization framework to a real data center network. This enables the fully autonomous management of the network concerning software changes (e.g., SDN flow rerouting, reconfigurations) using our approach. Our multi-dimensional cost-model will become useful as we can associate different changes with different types of costs (e.g., investment cost, required working hours, expected downtime, network stability). In combination with other advanced monitoring and forecasting techniques, our approach will also be able to warn and possibly even proactively react to performance issues before they appear in the network.

7 Conclusion

To continuously meet the requirements in changing environments with frequently changing demands, networks have to be adapted at runtime. Several approaches exist for the adaptation and optimization of networks, however, these either focus on design-time or are limited to a single objective. In this paper, we present a multi-objective adaptation framework for the online adaptation of networks through different adaptation techniques. A MAPE-K adaptation control loop enables iterative adaptations until all SLAs are met. Branch & Bound tracks different solutions and filters useless solution candidates at an early stage.

The strategies and tactics consider the objectives to only trigger meaningful adaptation operations w.r.t. to the violated SLA. Adaptation tactics use network-specific domain knowledge for choosing alternative configurations. Additional adaptation tactics can be easily added. Each solution candidate is analyzed to provide only SLA compliant solutions. The resulting solutions are all cost-optimal and represent the Pareto-front.

A comparison to a brute force approach shows that all cost-optimal solutions are found, where the simulation ensures the correctness. An additional scenario demonstrates the subsequent execution of adaptation operations and identifies a constant adaptation time even on large networks and an increasing number of adaptation points.

Acknowledgements

This work was funded by the German Research Foundation (DFG) under grant No. (KO 3445/18-1). Special thanks to our student Pascal Fries, who assisted us with the implementation and evaluation of the alternative route adaptation tactic.

References

1. Ahn, J.H., Binkert, N., Davis, A., McLaren, M., Schreiber, R.S.: HyperX: Topology, Routing, and Packaging of Efficient Large-Scale Networks. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 1–11 (Nov 2009)
2. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing MAPE-K feedback loops for self-adaptation. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 13–23. IEEE (2015)
3. Bari, M.F., Boutaba, R., Esteves, R., Granville, L.Z., Podlesny, M., Rabbani, M.G., Zhang, Q., Zhani, M.F.: Data Center Network Virtualization: A Survey. IEEE Communications Surveys Tutorials **15**(2), 909–928 (Second 2013)
4. Bause, F.: Queueing Petri Nets-A formalism for the combined qualitative and quantitative analysis of systems. In: Proceedings of 5th international workshop on Petri nets and performance models. pp. 14–23. IEEE (1993)
5. Benson, T., Anand, A., Akella, A., Zhang, M.: The Case for Fine-Grained Traffic Engineering in Data Centers. INM/WREN (2010)
6. Benson, T., Anand, A., Akella, A., Zhang, M.: MicroTE: Fine Grained Traffic Engineering for Data Centers. In: 7th CoNEXT. ACM (2011)
7. Buchanan, J.M.: The relevance of Pareto optimality. Journal of conflict resolution **6**(4) (1962)
8. Bültmann, D., Mühleisen, M., Klagges, K., Schinnenburg, M.: openWNS-open Wireless Network Simulator. In: European Wireless Conference, EW. IEEE (2009)
9. Butt, N.F., Chowdhury, M., Boutaba, R.: Topology-Awareness and Reoptimization Mechanism for Virtual Network Embedding. In: International Conference on Research in Networking. Springer (2010)
10. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1) (2011)
11. Campanile, L., Gribaudo, M., Iacono, M., Marulli, F., Mastroianni, M.: Computer network simulation with ns-3: A systematic literature review. Electronics **9**(2), 272 (2020)
12. Chen, M., Miao, Y., Humar, I.: OPNET IoT Simulation. Springer Nature (2019)
13. Coello, C.A.C., Lamont, G.B., Van Veldhuizen, D.A., et al.: Evolutionary algorithms for solving multi-objective problems, vol. 5. Springer (2007)
14. Curtis, A.R., Carpenter, T., Elsheikh, M., López-Ortiz, A., Keshav, S.: Rewire: An Optimization-based Framework for Unstructured Data Center Network Design. In: INFOCOM. IEEE (2012)
15. Datta, S., Das, S.: Multiobjective support vector machines: handling class imbalance with pareto optimality. IEEE transactions on neural networks and learning systems **30**(5), 1602–1608 (2018)
16. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
17. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik **1**(1) (1959)
18. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406). vol. 2, pp. 1470–1477. IEEE (1999)

19. Durillo, J.J., Nebro, A.J., Alba, E.: The jMetal framework for multi-objective optimization: Design and architecture. In: IEEE congress on evolutionary computation. pp. 1–8. IEEE (2010)
20. Fang, W., Liang, X., Li, S., Chiaraviglio, L., Xiong, N.: VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. Computer Networks (2013)
21. Fredericks, E.M., Gerostathopoulos, I., Krupitzer, C., Vogel, T.: Planning as optimization: Dynamically discovering optimal configurations for runtime situations. In: Proceedings of the 13th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2019. IEEE (June 2019)
22. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer **37**(10), 46–54 (2004)
23. Herrnleben, S.: Model-Based Network Analysis and Optimization. Master Thesis, University of Wuerzburg (2017)
24. Jarschel, M., Zinner, T., Hoßfeld, T., Tran-Gia, P., Kellerer, W.: Interfaces, attributes, and use cases: A compass for SDN. IEEE Communications Magazine **52**(6), 210–217 (2014)
25. Jiang, J.W., Lan, T., Ha, S., Chen, M., Chiang, M.: Joint VM Placement and Routing for Data Center Traffic Engineering. In: INFOCOM. vol. 12 (2012)
26. Kim, M., Hiroyasu, T., Miki, M., Watanabe, S.: Spea2+: Improving the performance of the strength pareto evolutionary algorithm 2. In: International Conference on Parallel Problem Solving from Nature. pp. 742–751. Springer (2004)
27. Kounev, S.: Performance modeling and evaluation of distributed component-based systems using queueing petri nets. IEEE Transactions on Software Engineering **32**(7), 486–502 (2006)
28. Kounev, S., Buchmann, A.: SimQPN—A tool and methodology for analyzing queueing Petri net models by means of simulation. Performance Evaluation **63**(4-5) (2006)
29. Lalanda, P., McCann, J.A., Diaconescu, A.: Autonomic Computing. Springer (2013)
30. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: A survey. Operations research **14**(4) (1966)
31. Lukasiewycz, M., Glaß, M., Reimann, F., Teich, J.: Opt4J - A Modular Framework for Meta-heuristic Optimization. In: Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011). pp. 1723–1730. Dublin, Ireland (2011)
32. Luke, S.: Ecj then and now. In: GECCO (Companion). pp. 1223–1230 (2017)
33. Martí, R., Laguna, M., Glover, F.: Principles of scatter search. european Journal of operational Research **169**(2), 359–372 (2006)
34. Nguyen-Ngoc, A., Lange, S., Geissler, S., Zinner, T., Tran-Gia, P.: Estimating the Flow Rule Installation Time of SDN Switches when Facing Control Plane Delay. In: 19th International GI/ITG MMB Conference. Erlangen (2 2018)
35. Pawar, C.S., Wagh, R.: A review of resource allocation policies in cloud computing. World Journal of Science and Technology **2**(3), 165–167 (2012)
36. Przybylski, A., Gandibleux, X.: Multi-objective branch and bound. European Journal of Operational Research **260**(3), 856–872 (2017)
37. Qiu, T., Li, B., Qu, W., Ahmed, E., Wang, X.: TOSG: A topology optimization scheme with global small world for industrial heterogeneous Internet of Things. IEEE Transactions on Industrial Informatics **15**(6), 3174–3184 (2018)

38. Qu, L., Assi, C., Shaban, K.: Delay-aware scheduling and resource optimization with network function virtualization. IEEE Transactions on Communications **64**(9), 3746–3758 (2016)
39. Reyes-Sierra, M., Coello, C.C., et al.: Multi-objective particle swarm optimizers: A survey of the state-of-the-art. International journal of computational intelligence research **2**(3), 287–308 (2006)
40. Riley, G.F.: Simulation of Large Scale Networks II: Large-scale Network Simulations with GTNetS. In: Proceedings of the 35th conference on Winter simulation: driving innovation. Winter Simulation Conference (2003)
41. Rygielski, P., Seliuchenko, M., Kounev, S.: Modeling and Prediction of Software-Defined Networks Performance using Queueing Petri Nets. In: Proceedings of the Ninth International Conference on Simulation Tools and Techniques (SIMUTools 2016) (August 2016), http://dl.acm.org/citation.cfm?id=3021426.3021437
42. Sachs, K., Kounev, S., Buchmann, A.: Performance modeling and analysis of message-oriented event-driven systems. Software & Systems Modeling **12**(4), 705–729 (2013)
43. Sofi, I.B., Gupta, A., Jha, R.K.: Power and energy optimization with reduced complexity in different deployment scenarios of massive MIMO network. International Journal of Communication Systems (2019)
44. Sommer, J., Scharf, J.: IKR Simulation Library. In: Modeling and Tools for Network Simulation. Springer (2010)
45. Tajiki, M.M., Salsano, S., Chiaraviglio, L., Shojafar, M., Akbari, B.: Joint Energy Efficient and QoS-aware Path Allocation and VNF Placement for Service Function Chaining. IEEE TNSM (2018)
46. Tso, F.P., Pezaros, D.P.: Improving Data Center Network Utilization Using Near-Optimal Traffic Engineering. IEEE Transactions on Parallel and Distributed Systems **24**(6) (2013)
47. Varga, A.: A practical introduction to the OMNeT++ simulation framework. In: Recent Advances in Network Simulation, pp. 3–51. Springer (2019)
48. Varga, A., Hornig, R.: An Overview of the OMNeT++ Simulation Environment. In: SIMUtools '08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium (2008), http://dl.acm.org/citation.cfm?id=1416222.1416290
49. Wang, L., Zhang, F., Vasilakos, A.V., Hou, C., Liu, Z.: Joint Virtual Machine Assignment and Traffic Engineering for Green Data Center Networks. SIGMETRICS Perform. Eval. Rev. **41**(3), 107–112 (Jan 2014)
50. Züfle, M., Bauer, A., Lesch, V., Krupitzer, C., Herbst, N., Kounev, S., Curtef, V.: Autonomic Forecasting Method Selection: Examination and Ways Ahead. In: Proceedings of the International Conference on Autonomic Computing (ICAC). pp. 167–176 (2019)