WARSAW UNIVERSITY OF TECHNOLOGY

ENGINEERING AND TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Ph.D. Thesis

Dariusz Nogalski, M.Sc.

Optimizing Critical Function Placement to Ensure Resilient Network Services – Selected Problems

Supervisor
Prof. Konstanty Junosza Szaniawski, Ph.D., D.Sc.

Additional supervisor
Mariusz Mycek, Ph.D.

WARSAW 2025

Abstract

The dissertation addresses the optimization of the placement of critical functions in network nodes to ensure the resilience of network services. Network services providing users with data transmission capabilities over the network (e.g., voice, video, e-mail, web, distributed file system) are vulnerable to various cyber-attacks. The three research studies propose new methods of protecting network services and increasing their resilience to attacks on availability. All three research areas concern the deployment of critical network functions in network nodes in a way that increases the network's resilience to selected availability attacks.

The first research area concerns the placement of SDN (*Software Defined Network*) controllers in a multi-controller design model and the identification of nodes most vulnerable to attacks. The goal of the network operator is to maximize network survivability, while the adversary's goal is to minimize network survivability. The measure of network survivability is the number of attack-surviving switches, i.e., those for which there is a path to the attack-surviving controller.

The second research area focuses on placing sensors detecting DDoS attacks (*Distributed Denial of Service*) in network nodes. The goal of optimization is to find such sensor placement that ensures that the largest possible part of the traffic directed to the set of target nodes (from the set of source nodes) is subject to analysis.

The third research area addresses the simultaneous optimization of routing and placement of traffic processing functions (selection of data centers) while building network function chains (*service chains*) that form the network services infrastructure. Selected functions in the service chain may be protective against attacks (e.g., firewall or intrusion detection system).

The research uses graph theory, mathematical programming, and game theory. MIP models (Mixed Linear Programming) and heuristics for large-scale networks were developed. The algorithms were implemented, and based on computational experiments, their effectiveness was compared.

Keywords: *software defined network, control placement problem, network function virtualization, distributed denial of service, service function chaining*

Streszczenie

Praca dotyczy optymalizacji rozmieszczenia funkcji krytycznych w węzłach sieci dla zapewnienia odporności usług sieciowych. Usługi sieciowe (ang. *network services*) zapewniające użytkownikom możliwość przesyłania danych przez sieć (np. głos, wideo, e-mail, www, rozproszony system plików) są podatne na różnorodne ataki cybernetyczne. Badania prowadzi się w trzech obszarach proponując nowe metody ochrony usług sieciowych i zwiększania ich odporności (ang. *resilience*) na ataki na dostępność. Wszystkie trzy obszary badawcze dotyczą rozmieszczenia krytycznych funkcji sieciowych (ang. *network function*) w węzłach sieci, w sposób zwiększający odporność sieci na wybrane ataki na dostępność.

Pierwszy obszar badań dotyczy rozlokowania sterowników SDN (ang. *Software Defined Network*) w modelu wielu sterowników (ang. *multi-controller design*) oraz identyfikacji węzłów najbardziej podatnych na ataki. Celem operatora jest maksymalizacja przeżywalności sieci, podczas gdy celem atakującego jest minimalizacja przeżywalności sieci. Miarą przeżywalności sieci jest liczba przełączników, które przetrwały atak, czyli takich dla których istnieje ścieżka do kontrolera który przetrwał atak.

Drugi obszar badawczy koncentruje się na rozmieszczeniu czujników wykrywających ataki DDoS (*Distributed Denial of Service*) w węzłach sieciowych. Celem optymalizacji jest znalezienie takiego rozmieszczenia czujników, które zapewni analizę jak największej części ruchu kierowanego do zbioru węzłów docelowych (ze zbioru węzłów źródłowych).

Trzeci obszar badawczy podejmuje jednoczesną optymalizację routingu i rozmieszczenia funkcji przetwarzania ruchu (wybór centrów danych) przy budowie łańcuchów funkcji sieciowych (ang. *service chains*) tworzących infrastrukturę usług sieciowych. Wybrane funkcje w łańcuchu mogą mieć charakter ochrony przed atakami (np. firewall, czy system wykrywania intruzów (ang. *Intrusion Detection System*)).

W badaniach wykorzystano teorię grafów, programowanie matematyczne oraz teorię gier. Opracowano modele programowania liniowego MIP (ang. *Mixed Linear Programming*) oraz heurystyki dla sieci o dużej skali. Algorytmy zostały zaimplementowane i na podstawie eksperymentów obliczeniowych porównano ich efektywność.

Słowa kluczowe: *sieci sterowane programowo, rozmieszczenie sterowników, wirtualizacja funkcji sieciowych, rozproszony atak na dostępność usługi, rozmieszczenie łańcuchów funkcji sieciowych*

To my wife Anna, daughters Nina, Helena, Jaśmina, and my parents.

Acknowledgements

First of all, I would like to thank my supervisor, Prof. Konstanty Junosza-Szaniawski, for his great support, patience, openness, and availability during this scientific journey.

I would also like to thank Prof. Michał Pióro and Dr Mariusz Mycek for their support in our work on the control placement problem.

Finally, I would like to thank Dr Slim Abdellatif, Dr Dallal Belabed, Prof. Pascal Berthou, and Stanislas Pedebearn for their support on the resource allocation problem in the SOFTANET project.

Contents

Chapter 1

Introduction

1.1 Motivation and goal of the research

Network Service Systems are complex telecommunications systems primarily focused on delivering services that utilize the network's capability to transport data. These systems cover a wide range of solutions, from the simplest ones that provide point-to-point delivery of data (P2P), through more complex systems that offer isolated L2 (e.g., Ethernet) or L3 (IP) tenant networks based on shared infrastructure (e.g., Virtual Private Routed Networks (VPRN)), to the most complex, heterogeneous network service systems (such as Content Delivery Networks (CDN), Software-Defined Wide Area Networks (SD-WAN) or mobile communication networks) that deliver complete sophisticated service chains.

Although the general principles of designing and implementing network service systems are known, organizing these systems requires careful planning each time, considering both technical and environmental conditions, as well as the business goals of the system operator. A primary requirement is to estimate the type and size of customer demand and equip the system with the means (resources and algorithms) to satisfy it effectively. Furthermore, it is critical to ensure the uninterrupted operation of the system in the face of environmental threats recognized by the operator (failures, random events), as well as deliberate attacks on the network infrastructure of the system and its processing functions.

The research presented in this thesis focuses on the planning and optimizing network service systems. The main goal of the study is to develop models, methods, and tools that allow for the optimal placement (in network and processing nodes) of functions that

are critical to ensuring efficient and resistant to attacks (and random events) functioning of these systems. Many attacks lead to the degradation of network services, causing their limited availability. This work addresses selected attacks on availability and methods of mitigating the effects of these attacks. The research concentrates on three selected problems, described below:

The first problem – *Robust Controller Placement* – considers the state-of-the-art Software-Defined Networks (SDN). SDN separates the control plane from the data plane to enable the construction of logically centralized control functions (network controllers) and efficient traffic management. Conversely, such architecture becomes vulnerable to new attack vectors, such as attacks on the control plane (control nodes or other nodes participating in control plane flows). The objective is to position network controllers to maximize the availability of network services provided by the SDN, w.r.t. the considered classes of attacks.

The second problem – *Traffic Sentinel Placement* – refers to the general class of IP transport networks whose *raison d'être* is to provide the end-users with access to the resources (e.g., data, web, processing, and storage services) exposed by the selected set of network nodes (target nodes). The network service system must secure the target nodes against DDoS attacks to ensure the uninterrupted availability of end-user resources. For this purpose, the system is to deploy a (limited) set of traffic sentinels whose task is to analyze traffic and signal the threat of a DDoS attack as early as possible. The goal of optimization is to find such sentinel placement that ensures that the largest possible part of the traffic directed to the target nodes is subject to analysis.

The third problem – *Multi-Domain Service Function Chain Placement* – concerns service systems whose task is to dynamically create and provide Service Function Chains (SFC) that are compliant with user requirements in an environment composed of many cooperating administrative domains (served by many operators). The research aims to simultaneously optimize network and computation resources while satisfying SFC requirements. Selected functions in the service chain may be protective against attacks (e.g., firewall or intrusion detection system).

To solve the above optimization problems, we used graph theory, mathematical programming, and game theory. Graph theory naturally models network structures, while traffic flow is typically represented using linear programming, which seamlessly inte-

grates with decision variables to determine the optimal placement of critical network function – such as controllers, sensors, or processing functions. We also applied a game-theoretic approach to capture the opposing objectives of the network operator and the attacker. Given the complexity of large-scale networks, heuristic algorithms were developed to provide near-optimal solutions when exact methods become computationally infeasible. These heuristics are often supported by MILP solvers, which help refine solutions by leveraging mathematical optimization techniques. The developed algorithms were implemented and tested through computational experiments, allowing for a thorough comparison of their effectiveness across different network configurations and attack scenarios.

In the following subsections 1.1.1-1.1.3, we provide a more detailed introduction to the three selected problems and present an overview of the proposed solutions.

1.1.1 Robust Controller Placement

Software-defined networking (SDN) is a network architecture that separates the control plane (controllers) from the data plane (switching devices). Deploying a single controller for the whole network is not scalable (delays, controller capacity, etc.) and not resilient (single point of failure).

We assume that any network node can host a controller. Following the work (Tomaszewski et al. [117]), a node can be controlled either by the local controller placed in the same location or, if the location does not house a controller, a remote controller. In the latter case, a node (e.g., a switch) communicates via an existing network path with the remote controller.

Here, we deal with the *multi-controller design*, where a logically centralized control plane is deployed by placing multiple controllers at different locations. How many controllers and where in the topology to put them is known as the *controller placement problem (CPP)*. It is a facility location type of problem known to be NP-hard (Heller et al. [49]) (Non-deterministic Polynomial-time hard).

In general, the CPP may address various criteria, e.g., delay (Dou et al. [32], Mycek et al. [84], Santos et al. [110], Wang et al. [123], Li et al. [74], controller capacity (Ibrahim et al. [54], Rath et al. [106], Yao et al.[131]), dynamics of network and traffic conditions (Toufga et al. [118], He et al. [48], Bari et al. [11], Dixit et al. [31]), reliability and

failures (Wang and Chen [126], Santos et al. [108], Tohidi et al. [116], Chaudhary and Kumar [23], Killi and Rao [69], Perrot and Reynaud [96], Vizaretta et al. [122], Zhang et al. [135]) and scalability (Lange et al. [72]).

Our focus is the *CPP resilient to targeted attacks on nodes* (Calle et al. [19][20], Pióro et al. [100][99], Rueda et al. [107], Santos et al. [109]).

Attacks on SDN nodes can be modeled as a two-player zero-sum game between the network operator and the attacker (DeVos and Kent [30]). The operator places M controllers, while the attacker selects K nodes to attack (disrupt connectivity), with both players acting under uncertainty (they do not know each other's moves).

The network is represented as a graph, where an attack removes nodes, potentially fragmenting the network. Surviving nodes remain connected to a controller, while non-surviving nodes lose routing capability. The game-theoretic objective is to optimize payoff, defined as the number of surviving nodes – the attacker seeks to minimize it, while the operator aims to maximize it.

Initially, a single strategy (min-max) approach is considered, but if the network recovers over time, a mixed-strategy framework is more suitable. In this case, both players assign probabilities to their choices. The attacker minimizes the maximum average payoff, while the operator maximizes the minimum average payoff (DeVos and Kent [30]).

This work introduces two novel mathematical models – one from the operator's and one from the attacker's perspective – for optimizing these probability distributions. A key challenge is the exponential growth of the game matrix, making pre-computation impractical. The study aims to develop an efficient optimization procedure for large-scale game scenarios.

1.1.2 Traffic Sentinel Placement

Denial of Service (DoS) attacks are intended to stop legitimate users from accessing a specific network resource [133]. A DoS attack is an attack on availability, one of the three dimensions of the well-known CIA security triad - Confidentiality, Integrity, and Availability. Availability is a guarantee of reliable access to information by authorized people. In 1999, the Computer Incident Advisory Capability (CIAC) reported the first *Distributed DoS* (DDoS) attack incident [27]. In a DDoS attack, the attacker gains control of a large number of users through a virus and then simultaneously performs a large

number of requests to a victim server via infected machines. As a result of this large number of tasks, the victim server is overwhelmed, out of resources, and unable to provide services to legitimate users.

DDoS attacks are a problem not only on the Internet [104] but also in the context of a Smart Grid (Wang et al. [124], Cameron et al. [21] and Huseinovic et al. [52]), Cloud (Bonguet and Bellaiche [17]) and Control Systems (Cetinkaya et al. [22]). According to [21], availability is more critical than integrity and confidentiality for Smart Grid environments.

DDoS attacks are difficult to defend against because of the large number of machines that can be controlled by botnets and participate in an attack. In consequence, an attack may be launched from many directions. A single bot (compromised machine) sends a small amount of traffic, which looks legitimate, but the total traffic at the target from the whole botnet is very high. This leads to an exhaustion of resources and disruption to legitimate users (Mirkovic and Reiher [80], Ranjan et al. [105]). Another difficulty is that the attack pattern may be changed frequently. Typically, only a subset of botnet nodes conduct an attack at the same time (Belabed et al. [13]). After a certain time, the botnet commander switches to another subset of nodes that conduct the attack.

As pointed out by Zargar et al. [133], there are basically two types of DDoS *flooding attacks*:

i) Disruption of a legitimate user's connectivity by exhausting bandwidth, router processing capacity, or network resources. These are essentially *link-flooding* attacks. Within this group, we have *Coremelt attacks* (Studer and Perrig [112]), and *Crossfire attacks* (Kang et al. [65]). These attacks aim at intermediate network links between attack sources and targets. Traditional target-based defenses do not work with these types of attacks (Liaskos and Ioannidis [76], and Gkounis et al. [46]).

ii) Disruption of a legitimate user's service by exhausting server resources (e.g., CPU, memory, bandwidth). These are essentially *target-flooding* attacks conducted at the application layer.

This work addresses *target-flooding* attacks with the assumption that there are multiple targets.

Some other well-known attacks are: *Reflector attacks* (Ramanathan et al. [104]) - an attacker sends a request with a fake address (of a victim) to DNS server, and the server

responds to the victim; *Spoofed attacks* (Armbruster et al. [9]) - an attacker forges the true origin of packets. Detailed classifications of DDoS attacks are discussed in, e.g., Mirkovic and Reiher [80], Douligeris and Mitrokotsa [33], Peng et al. [94], Zargar et al. [133], Bonguet and Bellaiche [17], and Huseinovic et al. [52].

A detection algorithm for DDoS attacks and the identification of an attack signature are out of the scope of this research. Various works have been published in the literature in this field, many of which use Machine Learning (ML) or Artificial Intelligence (AI) techniques. A combination of various ML/AI techniques (e.g., Logistic Regression, Decision Trees, Random Forest, K-Nearest Neighbors, Support Vector Machines (SVM), Multilayer Perceptron, Naive Bayes, Ada Boost) is used in Garcia et al. [42], Maksimovic et al. [78] and Rios et al. [79]. Aljebreen et al. [5] use an ensemble of three Deep Learning (DL) approaches namely Long Short-Term Memory (LSTM), Bidirectional LSTM, and Deep Belief Network. Al Dunainawi et al. [4] develop a DL model based on one-dimensional Convolutional Neural Network (CNN). Almadhor et al. [6] concentrate on Explainable Artificial Intelligence together with Federated Deep Neural Networks. Saini and Somani [71] use the Artificial Neural Network and Random Forest. Daya et al. [29] incorporate graph-based features into ML. Other works focus on general methods of anomaly detection, including signature-based and profile-based methods. Huang et al. [51] propose a multi-channel network traffic anomaly detection method combined with multi-scale decomposition. Hwang et al. [53] present an anomaly traffic detection mechanism, which consists of a CNN and an unsupervised DL. Zang et al. [132] use the Ant Colony Optimization to construct the baseline profile of the normal traffic behavior. Other related works are present e.g., Liu et al. [77], Gera and Battula [44], Jiao et al. [59], Zekri et al. [134], Assis et al. [10], Kallitsis et al. [64], and Afek et al. [3]. Comprehensive surveys of DDoS detection are also available: Jafarian et al. [56] overview anomaly detection mechanisms in software-defined networks; Khala et al. [67] focus on the defense methods that adopt artificial intelligence and statistical approaches.

One of the ways to defend against a DDoS attack is to place traffic sentinels (sensors, in short) in the network that recognize and stop unauthorized demands (e.g., Defrawy et al. [35], Armbruster et al. [9], Jeong et al. [58], Islam et al. [55] and Fayaz et al. [39])). However, placing such sensors in every network node would be expensive and inefficient. We propose a method to solve it efficiently.

A DDoS attack can be modeled as a flow from multiple sources to a single target (single commodity flow). Defined are directed graph with a capacity function on edges, a set of sources (S), and a set of targets (T). An attacker can conduct an attack on any vertex $t \in T$. The strength of an attack is given by a value of a $\mathsf{maxflow}_G(S,t)$, i.e., the value of the maximum flow from S to t in the network G. In this DDoS defense approach, sensors are placed in network nodes to recognize and stop unwanted traffic. If a sensor is placed in a vertex $v \in V$, then all the edges incident to v are assumed to be controlled. A set $D \subseteq V$ is called a set of sensors. The goal of this defense is to limit maximum uncontrolled flow towards each $t \in T$. Having a placement D, a maximum uncontrolled flow is determined and easy to compute. For that purpose, for each $t \in T$ max-flow algorithm (see, for example (Goldberg and Tarjan [47])) can be used for a graph $G \setminus D$ ($|T|$ runs of the algorithm). A super vertex ss is added to G, connected with a directed edge to each $s \in S$. For each run of the algorithm, ($t \in T$) maximum flow from ss to t is computed. Finally, maximum uncontrolled flow as $\max_{t \in T} \mathsf{maxflow}_G(ss,t)$ is computed.

For computational reasons, two variants of the sensor placement problem are given. First, the PQ problem, where a tolerable amount $a \in \mathbb{R}$ of uncontrolled flow is set, and a minimum number of sensors needed to achieve it is required. Second is the PC problem, where the number of sensors is set, and the question of how much uncontrolled flow we can reduce with such a number of sensors is asked. The main result of this work is two mixed integer models describing PQ and PC problems of optimal sensor placement against DDoS attacks. Moreover, two efficient heuristics (one for each problem) are presented on a large scale.

1.1.3 Multi-Domain Service Function Chain Placement

With the advent of network function virtualization, software networks are no longer limited to providing connectivity (i.e., a path with QoS) but also offer computational capabilities (i.e., network functions) along the path. A network service can then be represented as an *Service Function Chain* (SFC) composed of a set of *Virtual Network Functions* (VNFs) and directional links that connect them (a flow of packets goes through the chain of VNFs composing the SFC). Selected functions in the chain may be protective against attacks (e.g., firewall or intrusion detection system). By constructing network services this way, we can achieve increased attack resistance. The problem of embedding service

chains onto the substrate network is called *SFC embedding*.

In the literature, multiple existing works are applied in the single-domain context. Addis et al. [2] formulate via mathematical programming the VNF Placement and Routing optimization problem, including compression constraints. Each demand requests a subset of VNFs; the order of network functions is not required (as opposed to this work). The formulation minimizes the maximum network link utilization (TE goal) and minimizes the number of cores (CPU) used by the instantiated VNFs. The two competing goals are prioritized. In Wion et al. [127], the SFC routing problem is formulated as the Integer Linear Program (ILP) model. The goal is to minimize the service function cost (proportional to the requests' bandwidth) and network link cost (static, proportional to the used bandwidth). Each demand requests a subset of VNFs, and the order of network functions is required (as in this work). The work compares centralized and decentralized computation of SFC paths. However, the bitrate of each demand flow doesn't change along a VNF chain. Peng and Di [95] maximize the compute resource utilization efficiency by jointly optimizing the VNF deployment, power, and spectrum resource allocation. Yang et al. [130] address the risk of network attacks and allocate SFC based on honeypots and backup technology to reduce the resource cost of protecting air traffic information networks while enhancing network security. They deploy SFC VNFs close to the shortest path between the source and destination endpoints, aiming to reduce SFC latency and save bandwidth. Murray et al. [83] presents a VNF placement and routing algorithm based on a column generation method that iterates between generating improved paths and optimizing VNF placement based on the generated paths. They optimize throughput, latency, and availability in a multi-layer radio access network (RAN). Ko et al. [70] assume a network of service nodes, where each service node is exposing SFs by means of network function virtualization (NFV). They propose an integer non-linear model that considers link latency and compute/storage resources (e.g., CPU, storage, and memory capacity). The goal is to optimize latency. For example, the chain for the latency-sensitive service, such as multimedia streaming and voice over IP (VoIP), has a tight latency requirement. In contrast, the chain for the file download service has a loose latency requirement. Popokh et al. [103] address efficient resource allocation for VNF placement while minimizing the communications latencies between VMs that are part of the VNF deployment.

A related problem to the SFC is virtual network embedding. Embedding virtual networks in a substrate network (SN) is the main resource allocation challenge in network virtualization. It is usually referred to as the *Virtual Network Embedding problem (VNE)* (Herrera et al. [45], Belbekkouche et al. [14]). VNE concentrates on the allocation of virtual resources both in nodes (mapping to substrate nodes and their compute resources) and links (mapping to substrate network resources - links/paths) (Botero et al. [18]). Computing optimal VNE is an NP-hard optimization problem (Herrera et al. [45], Fischer et al. [40]). Even in the case when we map only virtual links (without information about node resources, e.g., VNFs, CPUs, etc.) to the substrate network links, in the single-path setting (i.e., the flow of a user in the SN follows a single path), the problem is NP-complete. Such a problem reduces to the decision version of the m-commodity flow problem with fixed rates and a single-path setting which is NP-complete (Drwal [34], Junosza-Szaniawski and Nogalski [61]). This is proved by the reduction from the decision version of the bin-packing problem. Even et al. [37] show that even the decision version of the two-commodity integral flow problem is NP-complete by reduction from the boolean satisfiability problem (SAT) problem.

In literature, multiple existing works are applied in the large-scale network context. Following Beck and Botero [12], resource allocation algorithms for setting up virtual network services should scale with the size of the substrate network. A solution should be found in the range of minutes or seconds – even in larger scenarios [12]. Tastevin et al. [115] propose the ILP formulation for SFC. Each network node is a PoP (Point of Presence) and can possibly host VNFs. It also has a limited CPU capacity, thus, multiple PoPs should be needed to serve all traffic demands. The work minimizes OPEX composed of two components: i) the VNF deployment cost, which is directly linked to the number of PoPs hosting VNF instances, and ii) the cost of forwarding traffic, which is linked to the number of hops in a traffic request path and its bandwidth usage. The longer the SFC path is, the more it will cost. SFC request is an ordered logical sequence of VNFs. They also propose a graph-based heuristic that combines graph centrality and multi-stage graphs. Beck and Botero [12] propose an NFV resource allocation problem (NFV-RA), that is divided into two problem stages: 1) service chain composition, to find VNF-FGs (VNF Forwarding Graph, a directed acyclic graph representing the request) to be embedded in the substrate network and 2) service chain embedding (VNF-FG embedding

in the substrate network). They propose a heuristic method to solve the composition of VNF chains and their embedding into the substrate network, in one coordinated step. Obadia et al. [88] present ILP formulation of SFC placement problem. Since the problem is NP-complete, they provide a heuristic based on game theory and implemented in a best response algorithm. They assume different NFV operating costs: a) VNF license cost (operating cost) (some VNF can have a license cost that depends on whether the VNF is running or not) b) energy cost (operating cost) of having a server with the VNF software running in idle mode. c) processing cost, a piece-wise linear function of the load, d) link cost. This work addresses the SFC problem on a large scale by proposing efficient heuristic for large network instances. However, our formulation is different from that of the above-cited works.

In a multi-domain network, NFV may be deployed and hosted by different domains. These latter may belong to the same (or other) authority and correspond to different network technologies or segments (access, backhaul, core). In this work, we deal with SFC embedding in a multi-domain (multi-administrative) environment and compose E2E multi-domain SFC services based on available national network and compute resources (slices, VNFs). In particular, we consider the case of federated military coalition networks. Although the SFC embedding problem is largely investigated in the literature, only a few works address it in a sliced collaborative multi-administrative network federation. We provide an exact ILP model, which is different from previous works (Addis et al. [2], Wion et al. [127]). We use a different goal function (a combination of slice deployment cost and link utility minimization). We use a (de)compression VNF and SFC ordering. Each topology link can represent a domain-level slice. In addition, since the problem is NP-hard (Addad et al. [1]), we provide an efficient heuristic for a large scale.

1.2 Contribution

The original contributions of the dissertation consist of the following elaborations:

- Models and algorithm of the control placement against attacks on nodes. The algorithm is based on column generation and iterates to find an optimal set of placements of controllers and attacks (dual problem) together with probability functions. To our knowledge, this novel research is the first work on optimal mixed-strategy control

placement. Game theory is used to find these optimal mixed strategies for operator and attacker (dual problem).

- Models and algorithms for the placement of detection sensors to counter DDoS attacks. To our knowledge, only a few existing studies focus on the placement of detection sensors to defend against DDoS attacks, and our algorithms tackle this problem from a different perspective.

- Models and algorithms of the service function chain placement in multi-domain. The novel optimization model is one of the first to address service function chain embedding in a sliced multi-administrative network federation. Additionally, to show the future research directions on SFC embedding, we provide novel optimization algorithms to solve a simplified problem (only network resources, no compute resources) based on relaxation and rounding. The heuristics are iterative; we set the best path (demand) embedding in each iteration and continue until all demands are set.

1.3 Thesis of the dissertation

The proposed optimization framework provides an efficient modeling means for optimizing critical function placement to ensure resilient network services.

1.4 Layout

The following three chapters present the results of our studies.

- Chapter 2 presents the control placement against attacks on nodes.

This chapter is based on the following publications:
Michał Pióro, Mariusz Mycek, Artur Tomaszewski, Konstanty Junosza-Szaniawski and **Dariusz Nogalski**, “Finding optimal mixed strategies in a matrix game between the attacker and the network operator”, RNDM, 2023 - [101].
Konstanty Junosza-Szaniawski and **Dariusz Nogalski**, “Game-theoretic approach to attack planning and controller placement in software defined networks”, ICMCIS, 2023 - [62].

- Chapter 3 addresses the sensor placement against DDoS attacks

This chapter is based on the following publications:
Konstanty Junosza-Szaniawski, **Dariusz Nogalski** and Paweł
Rzążewski, “Exact and Approximation Algorithms for Sensor Placement Against DDoS Attacks”, AMCS, 32.1 (2022) - [63].
Konstanty Junosza-Szaniawski, **Dariusz Nogalski** and Agnieszka
Wójcik, “Exact and Approximation Algorithms for Sensor Placement Against DDoS Attacks”, FedCSIS WCO, 2020 - [60].

- Chapter 4 focuses on the service function chain placement in multi-domain.

This chapter is based on the following publications:
Dariusz Nogalski, Dallal Belabed, Alexandre Triollet, Konstanty Junosza-Szaniawski, Slim Abdellatif, Pascal Berthou, Stanislas Pedebearn, Adam Dudko, “Critical function placement based on service chains in multi-administrative federated networks”, JCOMSS, 21.1 (2025) - [85].
Dariusz Nogalski, Dallal Belabed, Alexandre Triollet, Konstanty Junosza-Szaniawski, Slim Abdellatif, Pascal Berthou, Stanislas Pedebearn, Adam Dudko, “Federated SFC Placement in Sliced Collaborative Multi-administrative Multi-Domain Networks”, SoftCOM, 2024 - [86].
Konstanty Junosza-Szaniawski, **Dariusz Nogalski**, “Exact and approximation algorithms for joint routing and flow rate optimization”, FedCSIS WCO, 2019 - [61].

All the above chapters have a similar structure: introduction to the problem (state of the art, motivation, contribution), problem description, model description, algorithm description, computational results, and finally conclusions and future works.

Chapter 2

Robust Controller Placement

2.1 Introduction

2.1.1 CPP against targeted attacks on nodes

Existing *Control Placement Problem* (CPP) studies (Calle et al. [20], Santos et al. [109]) distinguish two network states: the regular state - network with all nodes operational and delay requirements satisfied; and the failure state - one or more nodes are shut down, as a consequence of targeted attacks. In the latter state, the operator will not achieve the same level of availability as in the regular state. For example, in a failure state, he can sacrifice delay requirements but may still try to preserve node survivability (e.g., each switch has a connection to a surviving controller, most likely a different one than the one assigned for regular state (Santos et al. [109])). Depending on the topology and the size of an attack, preserving survivability may not be guaranteed, and the operator may only try to maximize it. The work (Calle et al. [20]) proposes the efficient placement of backup controllers for failure states.

As stated by Rath et al. [106], in SDN, the controllers can run either on virtual or physical machines. Placing controllers can be made possible by dynamically adding or removing controllers as needed, i.e., adding a controller by invoking a new Virtual Machine (VM) and removing a controller by shutting down the VM.

The dual to CPP problem is the *attack planning problem*. The problem is related to the Critical Node Detection (CND) problem, which aims to identify a set of optimal nodes (the critical nodes) on a given network that, if removed, minimize or restrict a given

metric of network connectivity (Faramondi et al. [38], Calle et al. [20]). Attack planning may be based on local, node-related, topological centrality measures (e.g., a node degree) or global measures (Calle et al. [20]). The problem of finding a critical attack for a pairwise connectivity measure was found NP-hard (Santos et al. [110]).

Attacks on SDN can be conducted via various means, e.g., fiber cuts, power outages, jamming, viruses and worms, configuration errors, and Distributed Denial of Service attack (DDoS) on SDN controller (Chapter 3, based on Szaniawski, Nogalski, Rzazewski [63]). In addition, as reported by Scott and Hayward [111], the control plane in SDN is exposed to various kinds of potentially malicious attacks.

Our focus is the CPP resilient to targeted attacks on nodes (Calle et al. [19][20], Pióro et al. [100][99], Rueda et al. [107], Santos et al. [109]).

Pióro et al. [99] introduce a set of probabilistic network availability measures and formulate an optimization model that determines the potentially most dangerous attacks the attacker might launch. They also formulate a counter-part optimization model that allows the network operator to derive the optimal placement of controllers, which maximizes availability of services with respect to a given set of network attacks.

Tomaszewski et al. [117] consider two complementary optimization problems. The first of them consists in finding a placement of a given number of controllers that maximizes the number of nodes that survive the worst case attack from a given list of attacks (max-min). The complementary problem, in turn, is to find an attack targeted at a given number of locations that minimizes the number of surviving nodes for any placement from a given list of controller placements (min-max).

2.1.2 Our proposal

Our placement approach (the operator's point of view) offers to protect the network against mixed-attack strategies (Szaniawski and Nogalski [62], Pióro et al. [101]).

Attacks against the SDN nodes can be viewed as a two-player game between the attacker and the network operator. The network operator chooses a set of nodes (for example an M-element set) where the controllers are placed while the attacker chooses a set (for example a K-element set) of nodes to attack. We assume that the attacker is unaware of where the controllers are actually placed and the operator is unaware of which nodes will be attacked. The network is modeled with a graph and the attack is

modeled by the set of nodes that are removed from the graph. The resulting graph may be disconnected and in general, not all of its components contain a controller. The nodes of a component containing a controller are called surviving nodes. All the nodes in a component without a controller are non-surviving nodes because they lose the capability of traffic routing.

We can model this situation with a zero-sum matrix game (DeVos and Kent [30]) where the payoff is the number of surviving nodes. The attacker strives to minimize his payoff whatever placement of M controllers is actually applied. The network operator, in turn, strives to maximize his payoff regardless of which K-node attack occurs. Such a min-max strategy corresponds to the pure (single) strategy in game theory.

However, if we assume that as a result of the attack, the network will fail only temporarily, and after some time it will return to its normal operation, then the next round of the game will start. For such a scenario, the so-called mixed strategy (DeVos and Kent [30]) is more appropriate. In such a strategy, each player assigns a probability to each possible move (by move we mean the choice of an attack or a controllers' placement), and in subsequent rounds, he draws his move according to its probability.

The attacker chooses the probability distribution of his attacks so as to minimize the maximum average payoff (over all possible placements), where the average payoff for a given placement is the sum (over all attacks) of the attacks' payoffs for the given placement, weighted by the attacks' probabilities. The operator, in turn, uses the probability distribution of his controllers' placements which maximizes the minimum average payoff (over all possible attacks), where the average payoff for a given attack is the sum (over all placements) of the placements payoffs for the given attack, weighted by the placements' probabilities.

In this chapter we introduce two novel mathematical models for optimizing the probability distributions of the mixed strategy. First model, $\mathbb{P}[\mathcal{S}, \mathcal{A}]$, from the operator's point of view, finds optimal probability distribution q^*, which maximizes average survivability of the network regardless of an attack $a \in \mathcal{A}$. Second model, $\mathbb{A}[\mathcal{A}, \mathcal{S}]$, from the attacker's point of view, finds optimal probability distribution p^*, which minimizes average survivability of the network regardless of a controllers' placement $s \in \mathcal{S}$. We denote $\mathcal{S}$ as the set of (allowable) controllers' placements, and $\mathcal{A}$ as the set of attacks targeted on network nodes. In compact formulation of the optimization procedure, the game matrix is fixed

and a parameter to the problem. Each element of the game matrix expresses the number of surviving nodes for a given placement and a given attack $\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$. Such approach however may be problematic for large networks, since it requires to generate payoff matrix of the size $\binom{V}{M} \times \binom{V}{K}$, where V is the number of network vertices (nodes), M number of vervices of a single controllers' placement, K number of vertices of a single attack.

Another novelty of the work is a column generation procedure that optimizes the considered distributions and can be applied to games with large game matrices. The procedure is based on two linear programming master problems (one for each player - above $\mathbb{P}[\mathcal{S},\mathcal{A}]$ and $\mathbb{A}[\mathcal{A},\mathcal{S}]$) and two corresponding integer programming pricing problems ($\mathbb{CP}[\mathcal{A},p^*]$ and $\mathbb{NA}[\mathcal{S},q^*]$). The efficiency of the proposed procedure is illustrated with a numerical study.

2.2 Problem description

2.2.1 Notation

Below we describe the notation (summarized in Table 2.1) used throughout the chapter. The considered network is represented by an undirected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, with the set of nodes $\mathcal{V} = \{1,2,\ldots,V\}$ and the set of links $\mathcal{E} \subseteq \{\{u,v\} \subseteq \mathcal{V} : u \neq v\}$ interconnecting the nodes. For each $e \in \mathcal{E}$, $\alpha(e), \beta(e) \in \mathcal{V}$ denote the end nodes of link e.

The set of (allowable) controllers' placements will be denoted by $\mathcal{S}$. Each placement $s \in \mathcal{S}$ is characterized by the set $\mathcal{V}(s)$, in which the controllers are actually placed. A typical example of such a set $\mathcal{S}$ is the set of all M-node placements, i.e., the set of all placements s with $|\mathcal{V}(s)| = M$; such a set will be denoted by $\mathcal{S}(M)$.

We consider a set $\mathcal{A}$ of attacks targeted on network nodes. Hence, each attack $a \in \mathcal{A}$ is characterized by the set $\mathcal{V}(a)$ of attacked node locations, which defines the set $\mathcal{C}(a)$ of (non-empty) connected components of graph $\mathcal{G}(a)$ resulting from attack a. For each component, $c \in \mathcal{C}(a)$, $\mathcal{V}(c)$ will denote the set of its nodes. A typical example of such a set $\mathcal{A}$ is the set of all K-node attacks, i.e., the set of all attacks a with $|\mathcal{V}(a)| = K$, for a given integer parameter K; such a set will be denoted by $\mathcal{A}(K)$.

When a node belongs to a component in $\mathcal{C}(a)$ that contains at least one controller, it is called a surviving node. All other nodes will not survive the attack because they are

either directly attacked (i.e., belong to $\mathcal{V}(a)$) or have no connection (via a path in the network graph) to any of the controllers that are not directly attacked.

For a given controllers' placement $s \in \mathcal{S}$ and a given attack $a \in \mathcal{A}$, we define $\mathcal{V}(s,a)$ – the set of nodes surviving attack a when controllers' placement s is deployed and $V(s,a) := |\mathcal{V}(s,a)|$ – the number of nodes surviving attack a when controllers' placement s is deployed. The quantity $V(s,a)$ will be used as the resilience measure in the considerations of this chapter.

Table 2.1: Summary of general notation

$\mathcal{V}, \mathcal{E}$	set of nodes and links ($V := \|\mathcal{V}\|, E := \|\mathcal{E}\|$)
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	network graph
$\alpha(e), \beta(e)$	end nodes of link $e \in \mathcal{E}$
$\mathcal{S}$	set of allowable controllers' placements
$\mathcal{S}(M)$	set of all placements composed of M controllers
$\mathcal{V}(s)$	set of controllers locations in placement $s \in \mathcal{S}$ ($V(s) := \|\mathcal{V}(s)\|$)
$\mathcal{A}$	set of expected attacks
$\mathcal{A}(K)$	set of all k-node attacks
$\mathcal{V}(a)$	set of nodes affected by attack $a \in \mathcal{A}$ ($V(a) := \|\mathcal{V}(a)\|$)
$\mathcal{G}(a)$	graph surviving attack a, $\mathcal{G} - \mathcal{V}(a)$
$\mathcal{C}(a)$	set of components of $\mathcal{G}(a)$ for $a \in \mathcal{A}$
$\mathcal{V}(c)$	set of nodes of component $c \in \mathcal{C}(a)$ ($V(c) := \|\mathcal{V}(c)\|$)
$\mathcal{V}(s,a)$	set of nodes that survive attack $a \in \mathcal{A}$ when placement $s \in \mathcal{S}$ is assumed ($V(s,a) := \|\mathcal{V}(s,a)\|$)
$\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$	payoff matrix

2.2.2 Game description

Consider the two-player zero-sum matrix game, where the move of the network operator is to choose the controllers' placement $s \in \mathcal{S}$, the move of the attacker is to choose the attack $a \in \mathcal{A}$, and both players are unaware of the opponent's choice. The payoff of the game, denoted by $V(s,a)$, is equal to the number of nodes surviving attack a for placement s. Hence $\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$ is the matrix of the game, hereinafter referred to as the *payoff matrix* (or *outcome matrix*). Clearly, the operator seeks to maximize the number

of surviving nodes while the attacker seeks to minimize it.

***Min-max* and *max-min* strategy**. One strategy the players can adopt is the so-called *min-max* (for the attacker) and *max-min* (for the defender). The **attacker** selects a single attack $a \in \mathcal{A}$ that minimizes the number of surviving nodes over all possible placements, assuming the operator simultaneously maximizes the availability measure. This is done by first selecting the maximum value for each column of $\mathbb{V}$, and then choosing the column (denoted as a^*) with the minimum value

$$V_{\min-\max} = \min_{a\in\mathcal{A}} \max_{s\in\mathcal{S}} V(s,a) = \max_{s\in\mathcal{S}} V(s,a^*). \tag{2.1}$$

Choosing such an attack a^* will be called *attacker's MM decision*. Note that there can be multiple attacks a^* satisfying equality (2.1), so the decision is generally not unique.

The **operator** selects a single placement $s \in \mathcal{S}$ that maximizes the number of surviving nodes over all possible attacks, assuming the attacker simultaneously minimizes the availability measure. This is done by first selecting the minimum value for each row of $\mathbb{V}$, and then choosing the row (denoted as s^*) with the maximum value

$$V_{\max-\min} = \max_{s\in\mathcal{S}} \min_{a\in\mathcal{A}} V(s,a) = \min_{a\in\mathcal{A}} V(s^*,a). \tag{2.2}$$

Choosing such a placement s^* will be called *defender's MM decision*; the decision defined this way may not be unique.

These strategies, considered in Tomaszewski et al. [117] and Junosza-Szaniawski and Nogalski [62], will be called *MM strategy* in short.

Given the way the outcomes of $\mathbb{V}$ identifying the values of $V_{\max-\min}$ and $V_{\min-\max}$ are found, it is not difficult to show that the inequality

$$V_{\max-\min} \leq V_{\min-\max} \tag{2.3}$$

always holds, and $V_{\max-\min} = V_{\min-\max}$ if, and only if, the outcome matrix contains at least one *saddle point*, i.e., an element (s^*, a^*) whose outcome $V(s^*, a^*)$ is the smallest in its row (i.e., in row s^*) and the largest in its column (i.e., in column a^*). It is also easy to show that $V(s^*, a^*) = V_{\max-\min} = V_{\min-\max}$ and thus the decision for the defender and attacker is to choose s^* and a^*, respectively. Moreover, when the outcome matrix contains multiple saddle points (which is possible), then their respective outcomes are equal to each other. Therefore, choosing any placement s^* and any attack a^* are *MM decisions*,

provided that s^* and a^* appear in some, perhaps different, saddle points (Figure 2.1 and Table 2.2 in Section 2.2.3).

Optimal mixed strategy. Here, we study another strategy, namely the *optimal mixed strategy*, where each player assigns to each of his possible moves (choices of a particular attack or controllers' placement, respectively) the probability with which the move is actually selected. Then, each player optimizes the expected payoff of the game, again assuming the worst, from his point of view, move of his opponent.

Thus, the **attacker** needs to find the probability distribution $p = (p_a : a \in \mathcal{A})$ according to which the attacks are launched (where $p \in \mathcal{P}(\mathcal{A}) := \{p : \sum_{a\in\mathcal{A}} p_a = 1; p_a \geq 0, a \in \mathcal{A}\}$), which minimizes the expected number of surviving nodes for the worst (from attacker's viewpoint) placements in $\mathcal{S}$. More precisely, assuming a placement $s \in \mathcal{S}$ as the move of the operator, the expected payoff of the attacker is

$$V(s, \mathcal{A}, p) = \textstyle\sum_{a\in\mathcal{A}} V(s, a) \cdot p_a,\ p \in \mathcal{P}(\mathcal{A}). \tag{2.4}$$

and the probability distribution in question minimizes the value of $V(s, \mathcal{A}, p)$ over all $s \in \mathcal{S}$. Hence, this minimized value, denoted by $V^*(\mathcal{A}, \mathcal{S})$, is expressed as follows

$$V^*(\mathcal{A}, \mathcal{S}) = \min_{p\in\mathcal{P}(\mathcal{A})} \max_{s\in\mathcal{S}} V(s, \mathcal{A}, p) \tag{2.5}$$

and distribution $p^* \in \mathcal{P}(\mathcal{A})$ is optimal if, and only if,

$$\max_{s\in\mathcal{S}} V(s, \mathcal{A}, p^*) = \min_{p\in\mathcal{P}(\mathcal{A})} \max_{s\in\mathcal{S}} V(s, \mathcal{A}, p). \tag{2.6}$$

Note that since $V^*(\mathcal{A}, \mathcal{S})$ is the minimum expected game outcome over all $p \in \mathcal{P}(\mathcal{A})$, the inequality $V^*(\mathcal{A}, \mathcal{S}) \leq V_{\min-\max}$ holds. This is because $V_{\min-\max} = \max_{s\in\mathcal{S}} V(s, \mathcal{A}, p)$ for any one-point distribution p such that $p_{a^*} = 1$ for some attack $a^* \in \mathcal{A}$ for which the minimum in formula (2.1) is reached and $p_a = 0$ for $a \in \mathcal{A} \setminus \{a^*\}$.

Analogously, the **operator** finds a probability distribution $q = (q_s : s \in \mathcal{S})$ according to which the controllers are placed (where $q \in \mathcal{Q}(\mathcal{S}) := \{q : \sum_{s\in\mathcal{S}} q_s = 1; q_s \geq 0, s \in \mathcal{S}\}$), which maximizes the expected number of surviving nodes for the worst (from operator's viewpoint) attacks in $\mathcal{A}$. Thus, for a given attack $a \in \mathcal{A}$, the expected payoff of the operator is equal to

$$V(\mathcal{S}, q, a) = \textstyle\sum_{s\in\mathcal{S}} V(s, a) \cdot q_s,\ q \in \mathcal{Q}(\mathcal{S}), \tag{2.7}$$

and the probability distribution in question maximizes the value $V(\mathcal{S}, q, a)$ over all $a \in \mathcal{A}$. Hence, the maximized value, denoted by $V^*(\mathcal{S}, \mathcal{A})$, is expressed as follows:

$$V^*(\mathcal{S}, \mathcal{A}) = \max_{q \in \mathcal{Q}(\mathcal{S})} \min_{a \in \mathcal{A}} V(\mathcal{S}, q, a). \tag{2.8}$$

and distribution $q^* \in \mathcal{Q}(\mathcal{S})$ is optimal if, and only if,

$$\min_{a \in \mathcal{A}} V(S, q^*, a) = \max_{q \in \mathcal{Q}(\mathcal{S})} \min_{a \in \mathcal{A}} V(S, q, a). \tag{2.9}$$

Note that since $V^*(\mathcal{S}, \mathcal{A})$ is the maximum expected game outcome over all $q \in \mathcal{Q}(\mathcal{S})$, the inequality $V^*(\mathcal{S}, \mathcal{A}) \geq V_{\max-\min}$ holds. This is because $V_{\max-\min} = \min_{a \in \mathcal{A}} V(\mathcal{S}, q, a)$ for any one-point distribution q such that $q_{s^*} = 1$ for some placement $s^* \in \mathcal{S}$ for which the maximum in formula (2.2) is reached and $q_s = 0$ for $s \in \mathcal{S} \setminus \{s^*\}$.

As shown in Section 2.3, the values of the two optimized expected outcomes are always equal, i.e., $V^*(\mathcal{S}, \mathcal{A}) = V^*(\mathcal{A}, \mathcal{S})$. Therefore, taking this equality and previous inequalities into account, we obtain the following relationship:

$$V_{\max-\min} \leq V^*(\mathcal{S}, \mathcal{A}) = V^*(\mathcal{A}, \mathcal{S}) \leq V_{\min-\max}. \tag{2.10}$$

Thus, if the outcome matrix has a saddle point, then the values of all four quantities in (2.10) are equal to each other. Otherwise, the above formula shows that using mixed strategies increases the payoffs of both players compared to the payoffs obtained with *MM* strategies; in fact this is the main reason for considering mixed strategies in general. Moreover, the numerical results presented in Section 2.5 show that the gain under consideration can be substantial.

Mathematical programming formulations optimizing the probability distributions in $\mathcal{P}(\mathcal{A})$ and $\mathcal{Q}(\mathcal{S})$, i.e., the payoffs $V^*(\mathcal{A}, \mathcal{S})$ and $V^*(\mathcal{S}, \mathcal{A})$, will be discussed in Section 2.3.

2.2.3 Examples

In this section we will illustrate the notions introduced above for three simple network topologies. Below, we will identify controllers placements and attacks with the sets of controller locations and the sets of attacked nodes, respectively.

Small network with saddle points. Consider a network in Figure 2.1, 2-node controller placements and 1-element attacks. In this case the outcome matrix contains four

saddle points $V(s,a)$ for placements s with $\mathcal{V}(s) \in \Big\{\{1,3\},\{1,4\},\{3,5\},\{4,5\}\Big\}$ and attack a with $\mathcal{V}(a) = \{2\}$ (Table 2.2). Hence, as discussed in the Section 2.2.2 there are four defender's *MM* deterministic decisions and one attacker's *MM* deterministic decision.

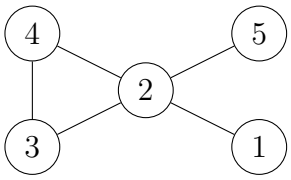


Figure 2.1: 5-node network with saddle point

	{1}	{2}	{3}	{4}	{5}	min
{1, 2}	4	1	4	4	4	1
{1, 3}	4	**3**	4	4	4	**3**
{1, 4}	4	**3**	4	4	4	**3**
{1, 5}	4	2	4	4	4	2
{2, 3}	4	2	4	4	4	2
{2, 4}	4	2	4	4	4	2
{2, 5}	4	1	4	4	4	1
{3, 4}	4	2	4	4	4	2
{3, 5}	4	**3**	4	4	4	**3**
{4, 5}	4	**3**	4	4	4	**3**
max	4	**3**	4	4	4	Attacker's min-max = 3 Defender's max-min = 3

Table 2.2: Game matrix for 5-node network with saddle point (bold).

Line network. Consider a line network consisting of V nodes ($\mathcal{V} = \{1, 2, \ldots, V\}$), assuming 1-node controllers placements ($\mathcal{S}(1)$) and 1-node attacks ($\mathcal{A}(1)$). In this case, there is no max-min strategy solution that would give the defender a positive payoff, simply because any location chosen for the controller can be attacked directly (hence $V_{\max-\min} = 0$). On the other hand, the attacker can secure the payoff $V_{\min-\max} = \lfloor \frac{V}{2} \rfloor$ by choosing the middle node for odd V (node 3 in Figure 2.2) and one of the two middle nodes for even V (node 3 or node 4 in Figure 2.3).

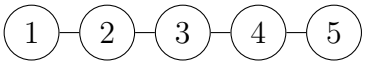


Figure 2.2: 5-node line network

	{1}	{2}	{3}	{4}	{5}	min	q_s^*	$V(s,p^*)$
{1}	0	1	2	3	4	0	0.5	2
{2}	4	0	2	3	4	0	0	2
{3}	4	3	0	3	4	0	0	0
{4}	4	3	2	0	4	0	0	2
{5}	4	3	2	1	0	0	0.5	2
max	4	3	2	3	4	Attacker's min-max = 2 Defender's max-min = 0	-	Attacker's $V^*(\mathcal{A},\mathcal{S}) = 2$
p_a^*	0	0	1	0	0	-	-	-
$V(q*,a)$	2	2	2	2	2	Defender's $V^*(\mathcal{S},\mathcal{A}) = 2$	-	-

Table 2.3: Game matrix for the 5-node line network, where $V(q^*,a) = V(\mathcal{S},q^*,a)$, and $V(s,p^*) = V(s,\mathcal{A},p^*)$

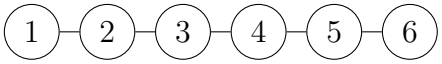


Figure 2.3: 6-node line network

	{1}	{2}	{3}	{4}	{5}	{6}	min	q_s^*	$V(s,p^*)$
{1}	0	1	2	3	4	5	0	0.5	2.5
{2}	5	0	2	3	4	5	0	0	2.5
{3}	5	4	0	3	4	5	0	0	1.5
{4}	5	4	3	0	4	5	0	0	1.5
{5}	5	4	3	2	0	5	0	0	2.5
{6}	5	4	3	2	1	0	0	0.5	2.5
max	5	4	3	3	4	5	Attacker's min-max = 3 Defender's max-min = 0	-	Attacker's $V^*(\mathcal{A},\mathcal{S}) = 2.5$
p_a^*	0	0	0.5	0.5	0	0	-	-	-
$V(q^*,a)$	2.5	2.5	2.5	2.5	2.5	2.5	Defender's $V^*(\mathcal{S},\mathcal{A}) = 2.5$	-	-

Table 2.4: Game matrix for the 6-node line network, where $V(q^*,a) = V(\mathcal{S},q^*,a)$, and $V(s,p^*) = V(s,\mathcal{A},p^*)$

An optimal solution for the mixed strategy is obtained as follows. To place a controller, the defender can choose the leftmost node with probability $\frac{1}{2}$ and the rightmost node with the same probability (no matter whether V is odd or even). Assuming an attack on node i, the expected value of the payoff is $\frac{i-1}{2} + \frac{V-i}{2} = \frac{V-1}{2}$, which is independent of the choice of $i \in \mathcal{V}$ (Table 2.3-2.4). In effect, the expected payoff for the defender increases to $V^*(\mathcal{S}(1),\mathcal{A}(1)) = \frac{V-1}{2}$. The attacker, in turn, can improve his strategy only in the case of even V by choosing one of the two middle nodes with probability $\frac{1}{2}$; this decreases the game outcome (and thus increases the attacker's payoff) to $V^*(\mathcal{A}(1),\mathcal{S}(1)) = \frac{V-1}{2}$ (Table 2.3-2.4).

To sum up, the solution obtained by optimizing the mixed strategy allows for a dramatic improvement of the defender's payoff, while improvement of the attacker's outcome is marginal (Table 2.3-2.4).

Cycle network. Now we move on to another example – a cycle network consisting of V nodes ($\mathcal{V} = \{1, 2, \ldots, V\}$), this time assuming 2-node placements ($\mathcal{S}(2)$) and 2-node attacks ($\mathcal{A}(2)$). In this case we have $V_{\max-\min} = 0$ (as before) and $V_{\min-\max} = V - 2$.

However, finding optimal solutions for the mixed strategy is more tricky. To illustrate this is point, let us consider a cycle network with $V = 4n$ for some $n \geq 4$.

It turns out that the safe probabilistic decisions for the defender is to place the controllers in a randomly selected set of 2 nodes from the following set: $\mathcal{S} = \left\{\{i, \frac{V}{2} + i\} : i = 1, 2, \ldots, \frac{V}{2}\right\}$ with probability $q(s) = \frac{2}{V}$ for $s \in \mathcal{S}$, and $q(s) = 0$ for $s \in \mathcal{S}(2) \setminus \mathcal{S}$. The attacker, in turn, should attack a randomly selected set of 2 nodes from the following set: $\mathcal{A} = \left\{\{i, (\frac{V}{4} + i + 1) (\mathrm{mod}\, V)\} : i = 1, 2, \ldots, V\right\}$ with probability $p(a) = \frac{1}{V}$, $a \in \mathcal{A}$, and $p(a) = 0$, $a \in \mathcal{A}(2) \setminus \mathcal{A}$. For the so defined optimal solutions the payoff equal $V^*(\mathcal{A}(2)) = V^*(\mathcal{S}(2)) = \frac{7}{8}V - 2$.

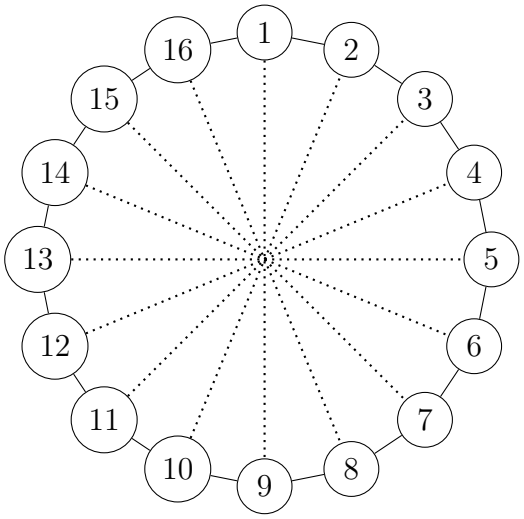


Figure 2.4: A 16-node cycle network and the selected placements

We will prove that the solution given above is optimal for both the defender and the attacker. To do this, we will show that the solution under consideration guarantees that the payoffs $V(\mathcal{S}, q, a) \geq \frac{7}{8}V - 2$ for any attack $a \in \mathcal{A}(2)$, and $V(\mathcal{A}, p, s) \leq \frac{7}{8}V - 2$ for any placement $s \in \mathcal{S}(2)$, and that the payoff $\frac{7}{8}V - 2$ (the same for the defender and the attacker) is reached by this solution.

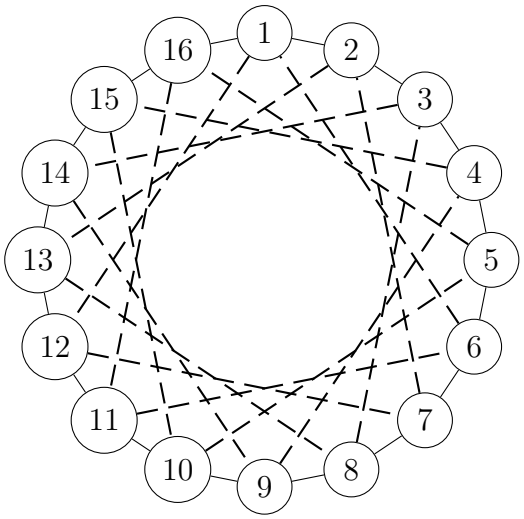


Figure 2.5: A 16-node cycle network and the selected attacks

First, we will show that $V(\mathcal{S}, q, a) \geq \frac{7}{8}V - 2$ for all $a \in \mathcal{A}(2)$. Due to the symmetry of set $\mathcal{S}$, without loss of generality, we can limit our considerations to attacks $a(j) \in \mathcal{A}(2)$, where $\mathcal{V}(a(j)) = \{1, j\}$, $j = 2, 3, \ldots, \frac{V}{2} + 1$, and consider two cases.

1. $j = \frac{V}{2} + 1$: For placement $\{1, \frac{V}{2} + 1\}$, the payoff is equal 0 and this happens with probability $\frac{2}{V}$. For the remaining placements in $\mathcal{S}$ (i.e., for placements $\{i, \frac{V}{2} + i\}$, $i = 2, 3, \ldots, \frac{V}{2}$), the payoff is equal to $V - 2$, and this happens with probability $1 - \frac{2}{V}$. Hence, the expected payoff is equal to $(V - 2)(1 - \frac{2}{V}) = V - 4 + \frac{4}{V}$, and this value is greater than $\frac{7}{8}V - 2$ for all $V \geq 14$.

2. $2 \leq j \leq \frac{V}{2}$: For placements $\{i, \frac{V}{2} + i\}$, $i = 2, 3, \ldots, j - 1$, the payoff is equal to $V - 2$ and this happens with probability $\frac{2(j-2)}{V}$. For placements $\{i, \frac{V}{2} + i\}$ for $i = 1$ or $i = j, j + 1, \ldots, \frac{V}{2}$, in turn, the payoff is equal to $V - j$ and this happens with probability $\frac{2(\frac{V}{2} - j + 2)}{V}$. So the expected payoff is equal to

$$(V - 2)\tfrac{2(j-2)}{V} + (V - j)\tfrac{2(\frac{V}{2} - j + 2)}{V} = \\ = \tfrac{2}{V}j^2 - \left(\tfrac{8}{V} + 1\right)j + V + \tfrac{8}{V}.$$

The above expression is minimized for $j = \frac{V}{4} + 2$ and its minimal value is $\frac{7}{8}V - 2$.

Hence, the considered solution guarantees the payoff at least $\frac{7}{8}V - 2$ for any attack $a \in \mathcal{A}(2)$.

Note that also in this case the solution obtained by optimizing the mixed strategy allows for a dramatic improvement in the defender's payoff (from 0 to $\frac{7}{8}V - 2$), while

the improvement in the attacker's payoff (from $V-2$ to $\frac{7}{8}V-2$) becomes asymptotically equal to 12.5%.

The so defined optimal 2-node placements and 2-node attacks are illustrated in Figure 2.4 and Figure 2.5, respectively, for $V = 16$ (i.e., $n = 4$). In both cases, the two nodes that make up a given 2-node set $\mathcal{V}(s)$, $s \in \mathcal{S}$, or $\mathcal{V}(a)$, $a \in \mathcal{A}$, are joined by a dotted line. It is easy to see that for any pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, the payoff $V(s, a)$ is either 10 or 14, and in each column and each row of the outcome matrix $\mathbb{V} = [V(s, a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$ exactly half of the elements (i.e., 4 elements in each column, and 8 elements in each row) are equal to 10 and the remaining half to 14. And since $q(s) = \frac{1}{8}$ for all $s \in \mathcal{S}$ and $p(a) = \frac{1}{16}$ for all $a \in \mathcal{A}$, $V(s, \mathcal{A}, p) = \frac{1}{16}(80 + 112) = 12$, and for all $a \in \mathcal{A}$, $V(a, \mathcal{S}, q) = \frac{1}{8}(40 + 56) = 12$. Hence, $V^*(\mathcal{S}(2)) = V^*(\mathcal{A}(2)) = 12 = \frac{7}{8}16 - 2$, as it should be.

Notice that optimal mixed strategies are not unique. Another example of optimal mixed strategies are given in the Figure 2.6 and in the table 2.5.

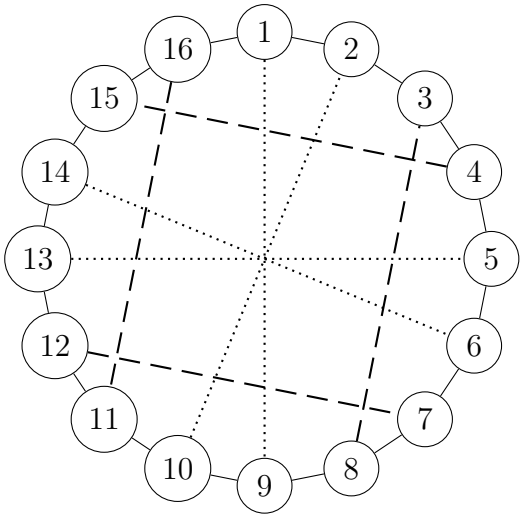


Figure 2.6: A 16-node cycle network and the selected four placements (doted lines) and four attacks (dashed lines)

Yet another example of optimal mixed strategies are given in the Figure 2.7 and in the Table 2.6.

	$\{3,8\}$	$\{4,15\}$	$\{7,12\}$	$\{11,16\}$	q_s^*	$V(s,p^*)$
$\{1,9\}$	10	14	14	10	0.25	12
$\{2,10\}$	10	14	14	10	0.25	12
$\{5,13\}$	14	10	10	14	0.25	12
$\{6,14\}$	14	10	10	14	0.25	12
p_a^*	0.25	0.25	0.25	0.25	-	Attacker's $V^*(\mathcal{A},\mathcal{S})=12$
$V(q^*,a)$	12	12	12	12	Defender's $V^*(\mathcal{S},\mathcal{A})=12$	-

Table 2.5: Game matrix for the 16-node cycle network, where $V(q^*,a)=V(\mathcal{S},q^*,a)$, and $V(s,p^*)=V(s,\mathcal{A},p^*)$

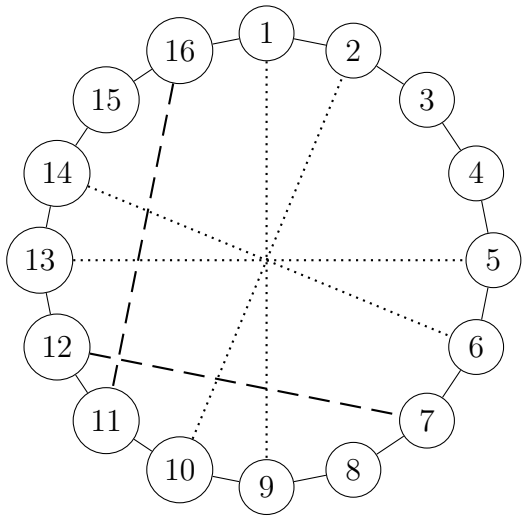


Figure 2.7: A 16-node cycle network and the selected four placements (doted lines) and two attacks (dashed lines)

	$\{7,12\}$	$\{11,16\}$	q_s^*	$V(s,p^*)$
$\{1,9\}$	14	10	0.25	12
$\{2,10\}$	14	10	0.25	12
$\{5,13\}$	10	14	0.25	12
$\{6,14\}$	10	14	0.25	12
p_a^*	0.5	0.5	-	Attacker's $V^*(\mathcal{A},\mathcal{S})=12$
$V(q^*,a)$	12	12	Defender's $V^*(\mathcal{S},\mathcal{A})=12$	-

Table 2.6: Game matrix for the 16-node cycle network, where $V(q^*,a)=V(\mathcal{S},q^*,a)$, and $V(s,p^*)=V(s,\mathcal{A},p^*)$

2.3 Model description

2.3.1 Optimizing p and q with compact formulations

Below we present two effective linear programming formulations that find an optimal probability distribution in $\mathcal{P}(\mathcal{A})$. The problem (abbreviated by $\mathbb{A}[\mathcal{A},\mathcal{S}]$) of finding an optimal

probability distribution p^*, for which $\max_{s\in\mathcal{S}} V(s,\mathcal{A},p^*) = V^*(\mathcal{S})$, can be formulated as a linear program (LP) in the following way.

$$\mathbb{A}[\mathcal{A},\mathcal{S}]: \min\ x \tag{2.12a}$$

$$[y]\ \ \textstyle\sum_{a\in\mathcal{A}} p_a = 1 \tag{2.12b}$$

$$[q_s \geq 0]\ \ \textstyle\sum_{a\in\mathcal{A}} V(s,a)p_a \leq x \qquad s\in\mathcal{S} \tag{2.12c}$$

$$x\in\mathbb{R};\ p_a\in\mathbb{R}_+ \qquad a\in\mathcal{A}. \tag{2.12d}$$

The (continuous) variables used in the formulation, i.e., x and $p = (p_a,\ a\in\mathcal{A})$ are primal variables, and the (continuous) variables y and $q = (q_s,\ s\in\mathcal{S})$ are dual variables.

Note that in any vertex solution of $\mathbb{A}[\mathcal{A},\mathcal{S}]$, no more (and usually less) than $|\mathcal{S}|+1$ of p_a variables are positive (because $|\mathcal{S}|+1$ is the number of constraints in (2.12)).

The problem (abbreviated by $\mathbb{P}[\mathcal{S},\mathcal{A}]$) of finding an optimal probability distribution q^*, for which $\min_{a\in\mathcal{A}} V(\mathcal{S},q^*,a) = V^*(\mathcal{A})$, is formulated as the following LP.

$$\mathbb{P}[\mathcal{S},\mathcal{A}]: \max\ y \tag{2.13a}$$

$$[x]\ \ \textstyle\sum_{s\in\mathcal{S}} q_s = 1 \tag{2.13b}$$

$$[p_a \geq 0]\ \ y \leq \textstyle\sum_{s\in\mathcal{S}} V(s,a)q_s \qquad a\in\mathcal{A} \tag{2.13c}$$

$$y\in\mathbb{R};\ q_s\in\mathbb{R}_+ \qquad s\in\mathcal{S}. \tag{2.13d}$$

This time, for the same reason as before, in any vertex solution of (2.13), no more (and usually less) than $|\mathcal{A}|+1$ of q_s variables are positive.

Clearly, problem $\mathbb{P}[\mathcal{S},\mathcal{A}]$ is the dual of $\mathbb{A}[\mathcal{A},\mathcal{S}]$ and vice versa (this why the primal variables x,p in $\mathbb{A}[\mathcal{A},\mathcal{S}]$ become dual variables in $\mathbb{P}[\mathcal{S},\mathcal{A}]$ and vice versa). This means that for any $p\in\mathcal{P}(\mathcal{A})$ and $q\in\mathcal{Q}(\mathcal{S})$, the following relation holds:

$$\begin{aligned}&\max_{s\in\mathcal{S}} V(s,\mathcal{A},p) \geq V^*(\mathcal{S}) = x^* = \\ &= y^* = V^*(\mathcal{A}) \geq \min_{a\in\mathcal{A}} V(\mathcal{S},q,a),\end{aligned} \tag{2.14}$$

where x^* and y^* denote the optimal values of the objective functions of $\mathbb{A}[\mathcal{A},\mathcal{S}]$ and $\mathbb{P}[\mathcal{S},\mathcal{A}]$.

The inequalities in (2.14) imply that

- the mixed strategy guaranties the same payoff for both players: $V^* = V^*(\mathcal{S}) = V^*(\mathcal{A})$

- in the MM strategy the operator's payoff is never greater than the attacker's payoff: $V_{\max-\min} \leq V_{\min-\max}$
- the common payoff of the mixed-strategy is between $V_{\max-\min}$ and $V_{\min-\max}$.

As we have seen in Section 2.2.3, the inequality in the second item above can be sharp and the resulting gap can be significant. We will come back to this observation in Section 2.5.

Thus, the mixed strategy is more efficient than its max-min counterpart when we think of the game composed of multiple rounds, where the attacker will launch each consecutive attack using distribution p^*, while the operator will rearrange the controllers' locations cyclically or after each attack. This will equalize the average payoffs for both players, at the same time increasing the payoff of the operator and decreasing the attacker's payoff, which is beneficial for both of them. This is illustrated in Section 2.2.3 and in the numerical study in Section 2.5.

2.3.2 Optimizing p and q through column generation

Having the payoff matrix $\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$ in hand, it is easy to determine $V^*(\mathcal{S})$ (and, for that matter, $V^*(\mathcal{A})$) since computing a single entry $V(s,a)$ of the payoff matrix for given $s \in \mathcal{S}, a \in \mathcal{A}$ is very fast and can be accomplished by applying a depth-first search algorithm. However, in general calculating the entire payoff matrix $\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$ is challenging since both $|\mathcal{S}|$ and $|\mathcal{A}|$ can grow exponentially with the number of nodes V. In the following sections, we show how to alleviate this issue using column generation when simultaneously solving formulations $\mathbb{A}[\mathcal{A}(K), \mathcal{S}(M)]$ and $\mathbb{P}[\mathcal{S}(M), \mathcal{A}(K)]$ for a given attack size K and a controllers' placement size M.

2.3.3 Pricing problem for $\mathbb{A}[\mathcal{A}, \mathcal{S}]$

Suppose we have solved the linear program $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ for a given set of attacks $\mathcal{A} \subset \mathcal{A}(K)$ and a given set of placements $\mathcal{S} \subset \mathcal{S}(M)$ to obtain an optimal primal solution x^*, p^* and an optimal dual solution y^*, q^* ($x^* = y^*$). (Note that the optimal dual solution is easily obtained from the optimal primal solution and vice versa.) Then, formulation (2.13) of the dual $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ implies that if there exists an attack $a' \in \mathcal{A}(K) \setminus \mathcal{A}$ for which constraint (2.13c) is broken, i.e., if $y^* > \sum_{s\in\mathcal{S}} V(s,a')q_s^*$, then adding this attack to $\mathcal{A}$ (and inequality $y \leq \sum_{s\in\mathcal{S}} V(s,a')q_s$ to formulation (2.13)) will cut off the current optimal solution from

the domain of $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ and in this way will likely decrease the maximum y^* of the dual objective function (2.13a). But since $y^* = x^*$, this will decrease the minimum x^* of the primal objective function (2.12a), i.e., will improve the payoff of the attacker. If such a' does not exist, the current solution of $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ solves $\mathbb{A}[\mathcal{A}(K), \mathcal{S}]$.

Finding such an attack a' is called the pricing problem for $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ (which in this context is called the master problem). Using binary variables a_v, z_v^s for $s \in \mathcal{S}, v \in \mathcal{V}$, can be formulated as an integer program (IP) as follows.

Attack generation problem $\mathbb{NA}[\mathcal{S}, q^*]$:

$$\min \ \textstyle\sum_{s \in \mathcal{S}'} q_s^* \cdot F_s \tag{2.15a}$$

$$\textstyle\sum_{v \in V} a_v = K \tag{2.15b}$$

$$z_v^s = 1 - a_v \qquad s \in \mathcal{S}', \ v \in \mathcal{V}(s) \tag{2.15c}$$

$$z^s_{\beta(e)} \geq z^s_{\alpha(e)} - a_{\beta(e)} \qquad e \in \mathcal{E}, \ s \in \mathcal{S}' \tag{2.15d}$$

$$z^s_{\alpha(e)} \geq z^s_{\beta(e)} - a_{\alpha(e)} \qquad e \in \mathcal{E}, \ s \in \mathcal{S}' \tag{2.15e}$$

$$F_s = \textstyle\sum_{v \in \mathcal{V}} z_v^s \qquad s \in \mathcal{S}' \tag{2.15f}$$

$$a_v, z_v^s, \in \mathbb{B} \qquad s \in \mathcal{S}', \ v \in \mathcal{V}, \tag{2.15g}$$

$$F_s \in \mathbb{Z}_+ \qquad s \in \mathcal{S}', \tag{2.15h}$$

where $\mathcal{S}' = \{s \in S : \ q^*(s) > 0\}$. In the formulation, $a_v = 1$ when node v is attacked, and $z_v^s = 1$ means that node v is surviving the constructed attack when $s \in \mathcal{S}$ is used. Then, integer variable F_s expresses the number of surviving nodes when attack a with $\mathcal{V}(a) = \{v \in \mathcal{V} : \ a_v = 1\}$ against placement $s \in \mathcal{S}'$ is applied, i.e., $V(s, a)$.

2.3.4 Pricing problem for $\mathbb{P}[\mathcal{S}, \mathcal{A}]$

The pricing problem for $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ is obtained in an analogous way. Using the same notation as above, we notice that if there exists a placement $s' \in \mathcal{S}(M) \setminus \mathcal{S}$ for which constraint (2.12c) is broken, i.e., if $\sum_{a \in \mathcal{A}} V(s', a) p_a^* > x^*$, then adding this placement to $\mathcal{S}$ (and inequality $\sum_{a \in \mathcal{A}} V(s', a) p_a \leq x$ to formulation (2.12)) will cut off the current optimal solution from the domain of $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ and in this way will likely increase the minimum x^* of the primal objective function (2.12a). And since $x^* = y^*$, this will increase the maximum y^* of the dual objective function (2.13a), i.e., will improve the payoff of the operator. If such s' does not exist, the current solution of $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ solves $\mathbb{P}[\mathcal{S}(M), \mathcal{A}]$.

Finding such a placement s' is the pricing problem for the master problem $\mathbb{P}[\mathcal{S}, \mathcal{A}]$, which, using binary variables s_v, y_v^a for $v \in \mathcal{V}, a \in \mathcal{A}$, can be formulated as the following IP.

Controllers' placement generation problem $\mathbb{CP}[\mathcal{A}, p^*]$:

$$\max \sum_{a \in A} p_a^* \cdot Y_a \tag{2.16a}$$

$$\sum_{v \in \mathcal{V}} s_v = M \tag{2.16b}$$

$$y_v^a = 0 \qquad a \in \mathcal{A},\ v \in \mathcal{V}(a) \tag{2.16c}$$

$$\sum_{v \in \mathcal{V}(c)} y_v^a \leq V(c) \cdot \sum_{v \in \mathcal{V}(c)} s_v \qquad a \in \mathcal{A},\ c \in \mathcal{C}(a) \tag{2.16d}$$

$$Y_a = \sum_{v \in \mathcal{V}} y_v^a \qquad a \in \mathcal{A} \tag{2.16e}$$

$$s_v, y_v^a \in \mathbb{B} \qquad v \in \mathcal{V},\ a \in \mathcal{A} \tag{2.16f}$$

$$Y_a \in \mathbb{Z}_+ \qquad a \in \mathcal{A}. \tag{2.16g}$$

Above, $s_v = 1$ when is chosen to host a controller, $y_v^a = 1$ means that node v is surviving attack a when the constructed placement with $\mathcal{V}(s) = \{v \in \mathcal{V} :\ s_v = 1\}$ is used, and Y_a expresses the total number of nodes surviving attack a.

2.4 Algorithm description

2.4.1 Column generation procedure for finding optimal p and q

Below we formulate a pseudo-code of the column generation procedure for finding p and q. The Algorithm 1 solves the problem iteratively (in a *while* loop, line 3-17). In a single iteration, the algorithm executes the following models in the following order:

- first, in line 5, it solves the model $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ (Section 2.3.1) to find the optimal payoffs $y^* = x^*$ and the optimal probability distributions q^* and p^*, for the current sets $\mathcal{S}$ and $\mathcal{A}$.

- Second, in line 6, it solves the model $\mathbb{CP}[\mathcal{A}, p^*]$ (Section 2.3.4) to generate a new placement $s' \in \mathcal{S}(M)$. If the newly generated placement s' improves the payoff x^* computed in line 5 (the check is done in line 7), such placement s' is added to the set $\mathcal{S}$ (line 8).

- Third, in line 11, it solves the model $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ (Section 2.3.1) to find the optimal payoff $x^* = y^*$ and the optimal probability distribution p^* and q^*, for the current sets $\mathcal{A}$ and $\mathcal{S}$.

- Forth, in line 12, it solves the model $\mathbb{NA}[\mathcal{S}, q^*]$ (Section 2.3.3) to generate a new attack $a' \in \mathcal{A}(K)$. If the newly generated attack a' improves the payoff y^* computed in line 11 (the check is done in line 13), such attack a' is added to the set $\mathcal{A}$ (line 14).

Algorithm 1 FMS(M, K)

1: Initialize $\mathcal{S}, \mathcal{A}$;
2: *continue* := *true*;
3: **while** *continue* **do**
4: *continue* := *false*;
5: Solve $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ to get x^*, p^* and y^*, q^*;
6: Solve $\mathbb{CP}[\mathcal{A}, p^*]$ to get $s' \in \mathcal{S}(M)$;
7: **if** $\sum_{a \in \mathcal{A}} V(s', a) p_a^* > x^*$ **then**
8: $\mathcal{S} := \mathcal{S} \cup \{s'\}$;
9: *continue* := *true*
10: **end if**
11: Solve $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ to get y^*, q^* and x^*, p^*;
12: Solve $\mathbb{NA}[\mathcal{S}, q^*]$ to get $a' \in \mathcal{A}(K)$;
13: **if** $y^* > \sum_{s \in \mathcal{S}} V(s, a') q_s^*$ **then**
14: $\mathcal{A} := \mathcal{A} \cup \{a'\}$
15: *continue* := *true*;
16: **end if**
17: **end while**
18: **return** $\mathcal{A}, \mathcal{S}, x^*, p^*, y^*, q^*$

Our algorithm can be modified by repeatedly performing the addition of a placement (and an attack correspondingly) in a single iteration. We could generate new placements in an internal *while* loop (repeat lines 5-10 in the *while* loop) until there is no further improvement of the payoff with a given set $\mathcal{A}$. We could also generate new attacks in an internal *while* loop (repeat lines 11-16 in the *while* loop) until there is no further improvement of the payoff with a given set $\mathcal{S}$. This approach yields four versions of the

algorithm: (1) single placement and single attack, (2) multiple placements with a single attack, (3) multiple placements with multiple attacks, and (4) single placement with multiple attacks. Since numerical experiments showed very similar results in terms of algorithm runtime across these variations, we chose to proceed with the single placement, single attack version.

2.5 Computational results

For the purpose of numerical experiments, we implemented Algorithm 1 in AMPL and run it on a standard laptop using the CPLEX 22.1 package for solving the optimization problems.

2.5.1 Experiments of "*cost266*" network

Below we present computational results for the average payoff obtained by Algorithm 1 for the "*cost266*" mesh network composed of 37 nodes and 57 links shown in Figure 2.8 (see SNDlib (Orlowski et al. [90])) for $M = 1, 2, \ldots, 15$ and $K = 2, 3, \ldots, 6$.

We conducted two experiments cases to see the impact of the algorithm start point:

- case 1: start with 100 worst topological attacks computed separately for each $K = 2, 3, 4, 5, 6$. The topological attacks were computed according to the model given in Appendix A.2 of the work by Pióro et al. [102]. Received results are given in Table 2.7-2.11.

- case 2: start with 1 random attack and 1 random placement. Received results are given in Table 2.12-2.16.

The two experiment cases, demonstrate that the payoff does not depend on the start point (Table 2.7 (case 1), Table 2.12 (case 2)). For each combination (M, K), the common operator' and attacker's payoff obtained with the mixed strategy (denoted by V^*) is compared with the payoff of the operator $V_{max-min}$ and the payoff the attacker ($V_{min-max}$) obtained with the MM strategy.

Thus, each element of the table in given row M shows the values $V_{\max-\min}$, $V^*(\mathcal{A}(K)) = V^*(\mathcal{S}(M))$, $V_{\min-\max}$, for each K under consideration. The payoffs of the operator and the

Table 2.7: Network "*cost266*" (case 1 - start from topological attacks): operator's and attacker's payoffs resulting from the MM strategy and the mixed strategy

$M \setminus K$	2	3	4	5	6	payoffs
1	0	0	0	0	0	$V_{max-min}$
	29	19	15.55	12.36	9.88	V^*
	29	19	17	13	13	$V_{min-max}$
2	0	0	0	0	0	$V_{max-min}$
	33.58	31.01	26.5	23.1	19.08	V^*
	34	32	27	25	20	$V_{min-max}$
3	34	0	0	0	0	$V_{max-min}$
	34.14	32.18	29.79	26.97	23.83	V^*
	35	34	31	29	26	$V_{min-max}$
4	34	29	0	0	0	$V_{max-min}$
	34.29	32.69	30.51	28.18	25.7	V^*
	35	34	32	31	29	$V_{min-max}$
5	34	32	29	0	0	$V_{max-min}$
	34.43	32.92	30.84	28.84	26.81	V^*
	35	34	33	32	30	$V_{min-max}$
6	34	32	29	25	0	$V_{max-min}$
	34.57	33.04	31.08	29.19	27.41	V^*
	35	34	33	32	31	$V_{min-max}$
7	34	32	29	26	19	$V_{max-min}$
	34.71	33.09	31.27	29.49	27.78	V^*
	35	34	33	32	31	$V_{min-max}$
8	34	32	30	26	23	$V_{max-min}$
	34.86	33.14	31.39	29.71	28.05	V^*
	35	34	33	32	31	$V_{min-max}$
9	35	33	30	27	25	$V_{max-min}$
	35	33.19	31.5	29.88	28.26	V^*
	35	34	33	32	31	$V_{min-max}$
10	35	33	30	28	25	$V_{max-min}$
	35	33.24	31.61	30.04	28.42	V^*
	35	34	33	32	31	$V_{min-max}$
11	35	33	30	28	26	$V_{max-min}$
	35	33.29	31.72	30.17	28.56	V^*
	35	34	33	32	31	$V_{min-max}$
12	35	33	31	29	26	$V_{max-min}$
	35	33.33	31.82	30.28	28.7	V^*
	35	34	33	32	31	$V_{min-max}$
13	35	33	31	29	27	$V_{max-min}$
	35	33.38	31.92	30.38	28.82	V^*
	35	34	33	32	31	$V_{min-max}$
14	35	33	31	29	27	$V_{max-min}$
	35	33.42	32.01	30.47	28.94	V^*
	35	34	33	32	31	$V_{min-max}$
15	35	33	31	30	27	$V_{max-min}$
	35	33.46	32.07	30.57	29.06	V^*
	35	34	33	32	31	$V_{min-max}$

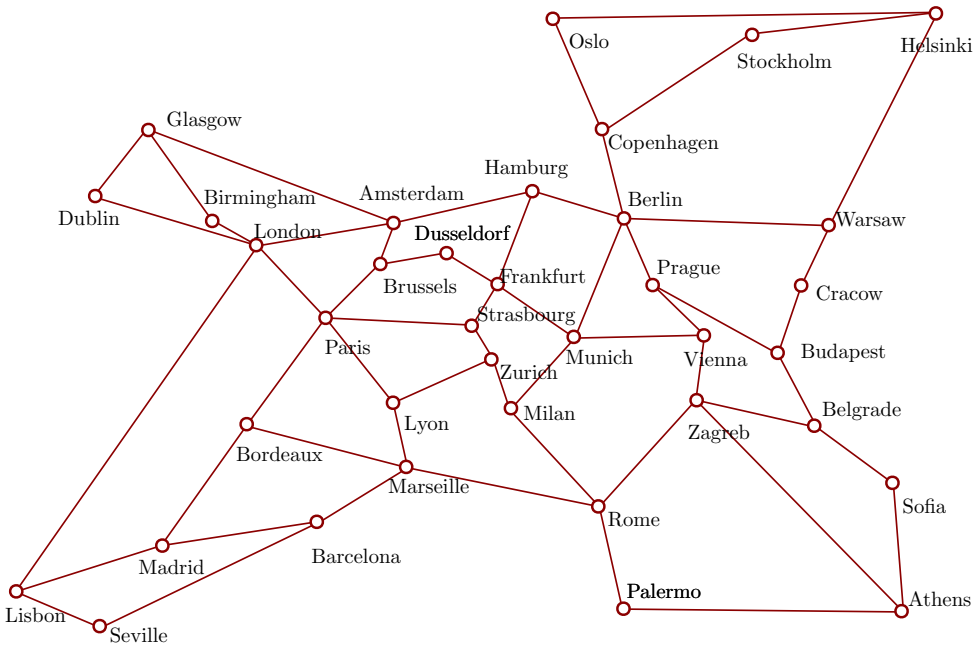


Figure 2.8: Network "*cost266*"

payoffs of the attacker in the MM strategy were obtained with the algorithms described in Tomaszewski et al. [117].

The relationship between these three payoff values is illustrated in Figure 2.9 for attack size fixed to $K = 6$ and the full range of placements sizes $M = 1, 2, \ldots, 15$. The three functions depicted there show that the mixed strategy improves (i.e., reduces) the attacker's payoff obtained by the MM strategy by no more than 12% for $M = 2, \ldots, 15$ (24% for the special case $M = 1$). The operator's analogous profit is much clearer, especially in the range of M, where the MM strategy payoff is equal to 0 (i.e., for $M = 1, \ldots, 6$). For $M = 7, 8$ the mixed strategy improves the operator payoff (compared to MM) by $46\%, 22\%$ respectively. In range $M = 9, \ldots, 15$ the improvement is up to 14%. Since similar relationship is observed for the remaining values of K, we can conclude that although both players benefit from the mixed strategy, the operator's profit is clearly better.

The computational times are presented in Table 2.8 (case 1) and 2.13 (case 2). The computation time of whole matrix Table 2.13 (case 2) is about 30% higher than of matrix Table 2.8 (case 1). It clearly shows the benefit of using the worst topological attacks as input (assuming we have them precomputed). For $K = 6$ we show the difference depending on the start point, for the two cases (Figure 2.10).

The number of iterations of Algorithm 1 are presented in Table 2.9 (case 1), Ta-

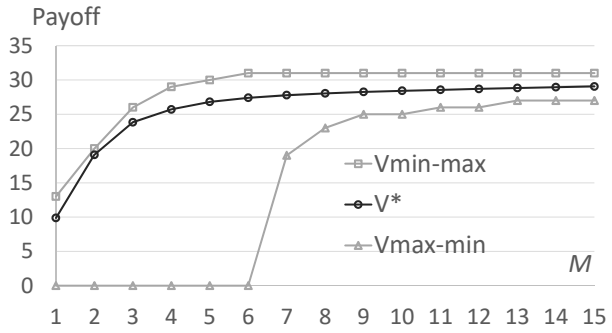


Figure 2.9: Network "*cost266*": optimized payoffs as functions of M (for fixed attack size $K = 6$)

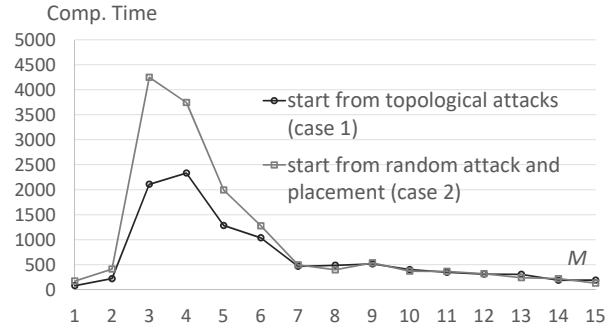


Figure 2.10: Network "*cost266*": $K = 6$, computational times [s] of Algorithm 1 depending on starting point

Table 2.8: Network "*cost266*" (case 1 - start from topological attacks): computational times [s] of Algorithm 1

$M \setminus K$	2	3	4	5	6
1	19	12	23	63	80
2	41	587	193	498	221
3	15	189	1315	1797	2108
4	16	77	550	1003	2332
5	17	85	121	761	1284
6	17	253	115	270	1037
7	30	220	128	270	467
8	31	146	143	269	489
9	1	156	127	307	515
10	1	138	133	352	402
11	1	118	159	361	349
12	1	114	114	233	312
13	1	107	142	134	306
14	1	90	309	132	188
15	1	91	288	213	190

ble 2.14 (case 2).

Table 2.10 (case 1) and Table 2.15 (case 2) show the sizes ($|\mathcal{S}| \times |\mathcal{A}|$) of the payoff

Table 2.9: Network "*cost266*" (case 1 - start from topological attacks): number of iterations of Algorithm 1

$M \setminus K$	2	3	4	5	6
1	20	16	22	45	52
2	34	86	82	91	66
3	16	60	175	199	161
4	17	37	124	168	262
5	19	38	49	151	168
6	18	61	43	77	163
7	30	61	48	70	94
8	31	53	52	72	93
9	2	61	52	76	98
10	2	60	55	82	88
11	2	56	63	89	84
12	2	58	53	73	87
13	2	57	60	58	85
14	2	52	92	58	71
15	2	53	90	75	75

matrices $\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$ for the sets $\mathcal{S}$ and $\mathcal{A}$ generated by Algorithm 1. These sizes are considerably smaller than the sizes of the full payoff matrices which are equal to $\binom{37}{M} \times \binom{37}{K}$ for $M = 1, 2, \ldots, 15$ and $K = 2, \ldots, 6$. The fact that in the case 1, a 100 topological attacks is given as input is reflected in the sizes of the payoff matrices (see Table 2.10, as compared to Table 2.15 (case 2)).

It should be noted, however, that the number of non-zero strategies computed for both players, the operator and the attacker (Table 2.11 (case 1) and Table 2.16 (case 2)), is much smaller than the size of the payoff matrix (Table 2.10 (case 1) and Table 2.15 (case 2)), for any $M = 1, 2, \ldots, 15$ and $K = 2, \ldots, 6$.

2.5.2 Experiments of "*coronet conus*" network

Below we present computational results for the average payoff obtained by Algorithm 1 for the CORONET continental United States topology (in short "*coronet conus*") mesh network composed of 75 nodes and 99 links shown in Figure 2.11 (Amorim and Pavani [8], Clapp et al. [26]) for $M = 1, 2, \ldots, 10$ and $K = 2, 3, 4$.

The Algorithm 1 was started with 100 worst topological attacks computed separately for each $K = 2, 3, 4$. The topological attacks were computed according to the model given in Appendix A.2 of the work by Pióro et al. [102].

In Table 2.17, for each combination (M, K), the common operator' and attacker's payoff obtained with the mixed strategy (denoted by V^*) is compared with the payoff

Table 2.10: Network "*cost266*" (case 1 - start from topological attacks): size $|\mathcal{S}| \times |\mathcal{A}|$ of the payoff matrix for $\mathcal{S}$ and $\mathcal{A}$ generated by Algorithm 1

$M \setminus K$	2	3	4	5	6
1	17 × 115	11 × 114	21 × 114	24 × 143	25 × 150
2	33 × 127	59 × 185	37 × 181	66 × 190	49 × 165
3	15 × 100	58 × 152	106 × 274	117 × 298	108 × 260
4	16 × 100	36 × 114	108 × 223	120 × 267	129 × 361
5	18 × 100	37 × 115	48 × 146	132 × 250	127 × 267
6	17 × 100	60 × 127	42 × 139	76 × 175	152 × 262
7	29 × 100	60 × 130	46 × 146	69 × 167	86 × 193
8	30 × 100	52 × 132	49 × 151	71 × 169	88 × 190
9	1 × 100	60 × 132	51 × 148	75 × 168	92 × 197
10	1 × 100	59 × 131	53 × 149	81 × 160	84 × 187
11	1 × 100	55 × 130	62 × 155	88 × 178	78 × 183
12	1 × 100	57 × 130	52 × 147	72 × 170	75 × 186
13	1 × 100	56 × 128	59 × 145	57 × 156	77 × 184
14	1 × 100	51 × 131	91 × 156	57 × 156	64 × 170
15	1 × 100	52 × 126	89 × 154	74 × 166	66 × 174

Table 2.11: Network "*cost266*" (case 1 - start from topological attacks): the number of non-zero strategies for the operator and the attacker generated by Algorithm 1

$M \setminus K$	2	3	4	5	6
1	14 × 1	7 × 1	15 × 9	20 × 17	22 × 21
2	26 × 6	44 × 15	24 × 4	31 × 24	26 × 22
3	7 × 7	21 × 5	33 × 10	27 × 14	33 × 23
4	7 × 7	14 × 11	21 × 10	26 × 13	21 × 14
5	6 × 7	15 × 12	14 × 12	29 × 18	24 × 20
6	7 × 7	28 × 26	18 × 11	25 × 23	28 × 22
7	6 × 7	28 × 25	21 × 11	26 × 25	28 × 23
8	7 × 7	24 × 22	24 × 12	25 × 23	27 × 26
9	1 × 1	25 × 22	19 × 12	27 × 24	27 × 25
10	1 × 1	24 × 22	22 × 12	29 × 25	29 × 26
11	1 × 1	24 × 22	20 × 13	26 × 24	33 × 23
12	1 × 1	25 × 22	16 × 13	29 × 22	26 × 18
13	1 × 1	26 × 24	19 × 13	25 × 22	25 × 22
14	1 × 1	24 × 24	28 × 21	27 × 24	25 × 19
15	1 × 1	24 × 24	30 × 21	32 × 32	24 × 13

of the operator $V_{max-min}$ and the payoff the attacker ($V_{min-max}$) obtained with the MM strategy.

The relationship between these three payoff values is illustrated in Figure 2.12 for attack size fixed to $K = 4$ and the full range of placements sizes $M = 1, 2, \ldots, 10$. The three functions depicted there show that the mixed strategy improves (i.e., reduces) the attacker's payoff obtained by the MM strategy by no more than 7% for all placement sizes considered. The operator's analogous profit is much clearer, especially in the range of M, where the MM strategy payoff is equal to 0 (i.e., for $M = 1, \ldots, 4$). For $M = 5, \ldots, 10$

Table 2.12: Network "*cost266*" (case 2 - start from random attack and placement): operator's and attacker's payoffs resulting from the MM strategy and the mixed strategy

$M \setminus K$	2	3	4	5	6	payoffs
1	0	0	0	0	0	$V_{max-min}$
	29	19	15.55	12.36	9.88	V^*
	29	19	17	13	13	$V_{min-max}$
2	0	0	0	0	0	$V_{max-min}$
	33.58	31.01	26.5	23.1	19.08	V^*
	34	32	27	25	20	$V_{min-max}$
3	34	0	0	0	0	$V_{max-min}$
	34.14	32.18	29.79	26.97	23.83	V^*
	35	34	31	29	26	$V_{min-max}$
4	34	29	0	0	0	$V_{max-min}$
	34.29	32.69	30.51	28.18	25.7	V^*
	35	34	32	31	29	$V_{min-max}$
5	34	32	29	0	0	$V_{max-min}$
	34.43	32.92	30.84	28.84	26.81	V^*
	35	34	33	32	30	$V_{min-max}$
6	34	32	29	25	0	$V_{max-min}$
	34.57	33.04	31.08	29.19	27.41	V^*
	35	34	33	32	31	$V_{min-max}$
7	34	32	29	26	19	$V_{max-min}$
	34.71	33.09	31.27	29.49	27.78	V^*
	35	34	33	32	31	$V_{min-max}$
8	34	32	30	26	23	$V_{max-min}$
	34.86	33.14	31.39	29.71	28.05	V^*
	35	34	33	32	31	$V_{min-max}$
9	35	33	30	27	25	$V_{max-min}$
	35	33.19	31.5	29.88	28.26	V^*
	35	34	33	32	31	$V_{min-max}$
10	35	33	30	28	25	$V_{max-min}$
	35	33.24	31.61	30.04	28.42	V^*
	35	34	33	32	31	$V_{min-max}$
11	35	33	30	28	26	$V_{max-min}$
	35	33.29	31.72	30.17	28.56	V^*
	35	34	33	32	31	$V_{min-max}$
12	35	33	31	29	26	$V_{max-min}$
	35	33.33	31.82	30.28	28.7	V^*
	35	34	33	32	31	$V_{min-max}$
13	35	33	31	29	27	$V_{max-min}$
	35	33.38	31.92	30.38	28.82	V^*
	35	34	33	32	31	$V_{min-max}$
14	35	33	31	29	27	$V_{max-min}$
	35	33.42	32.01	30.47	28.94	V^*
	35	34	33	32	31	$V_{min-max}$
15	35	33	31	30	27	$V_{max-min}$
	35	33.46	32.07	30.57	29.06	V^*
	35	34	33	32	31	$V_{min-max}$

Table 2.13: Network "*cost266*" (case 2 - start from random attack and placement): computational times [s] of Algorithm 1

$M \setminus K$	2	3	4	5	6
1	20	28	116	154	173
2	240	1523	568	636	412
3	16	417	1951	2530	4248
4	17	63	1218	1636	3743
5	16	65	102	873	1994
6	19	192	83	322	1279
7	16	189	144	284	497
8	28	140	138	218	397
9	19	111	101	224	536
10	21	107	118	313	369
11	11	84	109	289	366
12	18	90	123	174	319
13	11	84	134	145	241
14	12	88	223	155	220
15	8	66	256	177	131

Table 2.14: Network "*cost266*" (case 2 - start from random attack and placement): number of iterations of Algorithm 1

$M \setminus K$	2	3	4	5	6
1	31	45	117	137	142
2	139	226	197	160	149
3	23	154	256	291	320
4	24	40	238	256	334
5	23	39	55	209	270
6	28	69	55	113	231
7	25	65	76	99	109
8	40	64	67	86	110
9	29	60	57	87	131
10	31	60	62	108	108
11	19	58	71	99	113
12	30	60	69	74	116
13	20	60	79	77	95
14	21	65	92	88	103
15	16	56	108	85	78

the mixed strategy improves the operator payoff (compared to MM) by up to 5%. Since similar relationship is observed for the remaining values of K, we can conclude that although both players benefit from the mixed strategy, the operator's profit is clearly better.

The computational times are presented in Table 2.18.

The number of iterations of Algorithm 1 are presented in Table 2.19.

Table 2.20 shows the sizes ($|\mathcal{S}| \times |\mathcal{A}|$) of the payoff matrices $\mathbb{V} = [V(s,a)]_{s\in\mathcal{S}}^{a\in\mathcal{A}}$ for the sets $\mathcal{S}$ and $\mathcal{A}$ generated by Algorithm 1. These sizes are considerably smaller than the

Table 2.15: Network "*cost266*" (case 2 - start from random attack and placement): size $|\mathcal{S}| \times |\mathcal{A}|$ of the payoff matrix for $\mathcal{S}$ and $\mathcal{A}$ generated by Algorithm 1

$M \setminus K$	2	3	4	5	6
1	20 × 31	21 × 45	36 × 117	36 × 137	35 × 142
2	135 × 139	120 × 226	80 × 197	104 × 160	108 × 149
3	22 × 14	146 × 154	170 × 256	181 × 291	213 × 320
4	23 × 13	39 × 34	206 × 238	193 × 255	193 × 334
5	22 × 13	38 × 31	53 × 55	197 × 209	223 × 270
6	27 × 13	68 × 47	54 × 54	105 × 113	217 × 231
7	24 × 11	64 × 38	74 × 73	96 × 98	94 × 109
8	39 × 13	63 × 45	62 × 67	85 × 86	102 × 110
9	28 × 11	59 × 46	56 × 57	86 × 83	121 × 131
10	30 × 14	60 × 45	61 × 54	107 × 94	103 × 108
11	18 × 11	57 × 44	70 × 66	98 × 85	106 × 113
12	29 × 10	59 × 40	68 × 61	74 × 72	100 × 116
13	19 × 10	59 × 45	78 × 68	76 × 77	81 × 95
14	20 × 10	64 × 48	92 × 64	87 × 87	88 × 103
15	15 × 10	55 × 38	107 × 61	84 × 78	69 × 77

Table 2.16: Network "*cost266*" (case 2 - start from random attack and placement): the number of non-zero strategies for the operator and the attacker generated by Algorithm 1

$M \setminus K$	2	3	4	5	6
1	13 × 1	8 × 1	16 × 9	21 × 17	22 × 21
2	28 × 6	41 × 15	20 × 4	31 × 24	23 × 22
3	8 × 7	22 × 5	34 × 10	30 × 14	42 × 23
4	7 × 7	13 × 13	21 × 10	27 × 13	20 × 14
5	7 × 7	14 × 12	13 × 12	26 × 18	29 × 20
6	7 × 7	26 × 26	16 × 11	26 × 23	25 × 22
7	5 × 7	25 × 25	21 × 11	30 × 25	30 × 24
8	7 × 7	24 × 22	21 × 12	25 × 23	29 × 25
9	1 × 1	26 × 22	18 × 12	27 × 24	28 × 25
10	1 × 1	25 × 22	23 × 12	27 × 25	31 × 26
11	1 × 1	24 × 22	21 × 13	26 × 24	29 × 29
12	1 × 1	25 × 22	21 × 14	26 × 22	26 × 18
13	1 × 1	25 × 24	20 × 13	29 × 22	29 × 22
14	1 × 1	26 × 24	27 × 21	27 × 24	26 × 19
15	1 × 1	25 × 24	30 × 21	30 × 25	26 × 13

sizes of the full payoff matrices which are equal to $\binom{75}{M} \times \binom{75}{K}$ for $M = 1, 2, \ldots, 10$ and $K = 2, 3, 4$.

It should be noted, however, that the number of non-zero strategies computed for both players, the operator and the attacker (Table 2.21), is much smaller than the size of the payoff matrix (Table 2.20), for any $M = 1, 2, \ldots, 10$ and $K = 2, 3, 4$.

Table 2.17: Network "*coronet conus*" (start from topological attacks): operator's and attacker's payoffs resulting from the MM strategy and the mixed strategy

$M \setminus K$	2	3	4	payoffs
1	0	0	0	$V_{max-min}$
	68	62.39	35.03	V^*
	68	64	36	$V_{min-max}$
2	0	0	0	$V_{max-min}$
	71.11	67.89	63.35	V^*
	73	70	66	$V_{min-max}$
3	71	0	0	$V_{max-min}$
	71.35	68.52	65.12	V^*
	73	72	69	$V_{min-max}$
4	71	67	0	$V_{max-min}$
	71.55	68.95	66.15	V^*
	73	72	71	$V_{min-max}$
5	71	67	64	$V_{max-min}$
	71.73	69.32	66.67	V^*
	73	72	71	$V_{min-max}$
6	71	68	64	$V_{max-min}$
	71.91	69.63	67.13	V^*
	73	72	71	$V_{min-max}$
7	71	68	64	$V_{max-min}$
	72.03	69.87	67.52	V^*
	73	72	71	$V_{min-max}$
8	71	68	65	$V_{max-min}$
	72.09	70.07	67.83	V^*
	73	72	71	$V_{min-max}$
9	71	68	65	$V_{max-min}$
	72.14	70.21	68.08	V^*
	73	72	71	$V_{min-max}$
10	71	68	66	$V_{max-min}$
	72.19	70.34	68.29	V^*
	73	72	71	$V_{min-max}$

Table 2.18: Network "*coronet conus*" (start from topological attacks): computational times [s] of Algorithm 1

$M \setminus K$	2	3	4
1	313	1052	431
2	2255	33211	72824
3	67	1763	34317
4	109	315	9025
5	102	457	3891
6	104	679	2911
7	598	763	3834
8	392	1914	3916
9	802	1528	6267
10	797	1617	5777

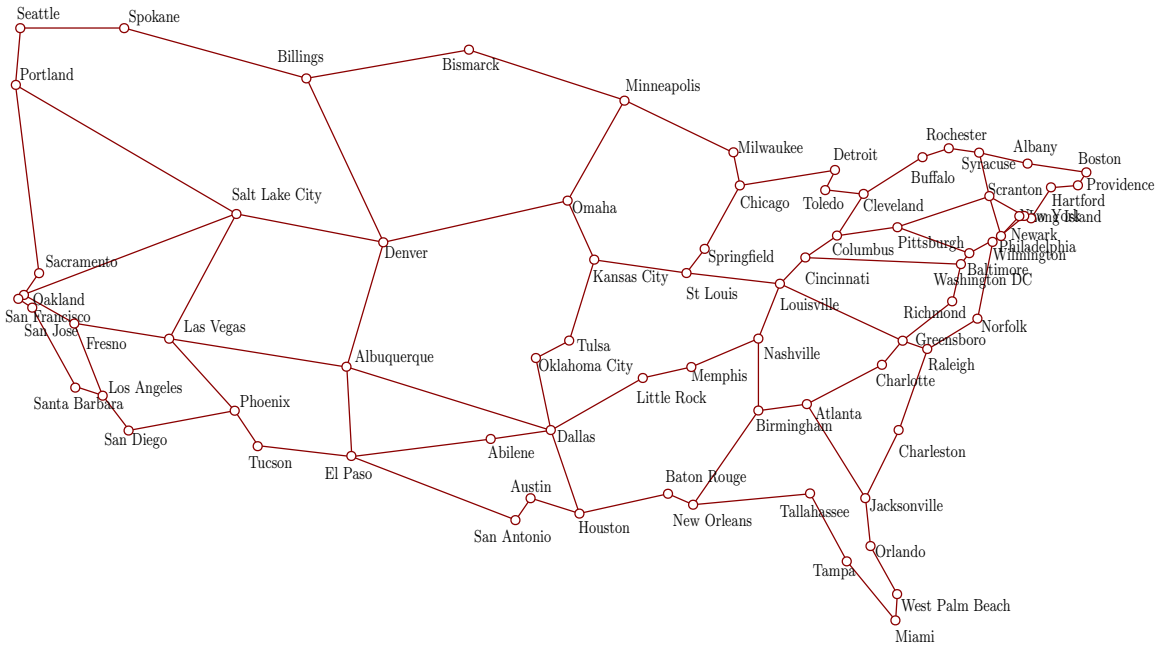


Figure 2.11: Network "*coronet conus*"

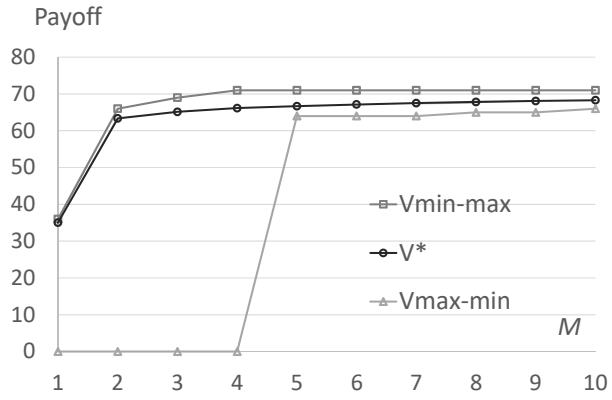


Figure 2.12: Network "*coronet conus*": optimized payoffs as functions of M (for fixed attack size $K = 4$)

Table 2.19: Network "*coronet conus*" (start from topological attacks): number of iterations of Algorithm 1

$M \setminus K$	2	3	4
1	47	67	109
2	103	144	287
3	26	92	313
4	28	29	270
5	28	36	84
6	34	44	85
7	72	47	91
8	67	66	99
9	80	67	109
10	81	74	119

Table 2.20: Network "*coronet conus*" (start from topological attacks): size $|\mathcal{S}| \times |\mathcal{A}|$ of the payoff matrix for $\mathcal{S}$ and $\mathcal{A}$ generated by Algorithm 1

$M \setminus K$	2	3	4
1	42 × 145	46 × 166	44 × 208
2	99 × 184	112 × 243	110 × 386
3	25 × 100	91 × 188	180 × 412
4	27 × 100	28 × 118	234 × 369
5	27 × 100	35 × 125	66 × 183
6	33 × 100	43 × 133	84 × 180
7	71 × 100	46 × 138	89 × 190
8	66 × 100	64 × 153	91 × 198
9	79 × 100	66 × 161	108 × 202
10	80 × 100	73 × 166	118 × 207

Table 2.21: Network "*coronet conus*" (start from topological attacks): the number of non-zero strategies for the operator and the attacker generated by Algorithm 1

$M \setminus K$	2	3	4
1	34 × 1	34 × 32	36 × 35
2	45 × 10	63 × 46	52 × 4
3	13 × 10	26 × 9	34 × 5
4	16 × 12	17 × 9	25 × 9
5	15 × 12	19 × 12	24 × 10
6	12 × 12	23 × 19	25 × 20
7	25 × 24	23 × 19	26 × 20
8	26 × 24	30 × 27	29 × 24
9	33 × 24	35 × 27	38 × 33
10	33 × 27	31 × 27	42 × 36

2.5.3 Algorithm convergence results

In this section we present several examples on how fast the Algorithm 1 converges to the optimal payoff, in function of the iteration number. First, we explain the search of the solution on a simple network "*cycle16*" (16-node cycle graph), for $K = 2$ and $M = 2$ (Figure 2.13). In this case, the Algorithm 1 was started with 1 random attack and 1 random placement.

In a single iteration the algorithm executes the following models in the following order: first, it solves the model $\mathbb{A}[\mathcal{A}, \mathcal{S}]$ (Section 2.3.1) to find the optimal payoffs $y^* = x^*$ and the optimal probability distributions q^* and p^*. Second, it solves the model $\mathbb{CP}[\mathcal{A}, p^*]$ (Section 2.3.4) to generate a new placement s'. Third, it solves the model $\mathbb{P}[\mathcal{S}, \mathcal{A}]$ (Section 2.3.1) to find the optimal payoff $x^* = y^*$ and the optimal probability distribution p^* and q^*. Forth, it solves the model $\mathbb{NA}[\mathcal{S}, q^*]$ (Section 2.3.3) to generate a new attack a'. Thus, in a single iteration we plot four measurements on the diagram.

In Figure 2.13, a single iteration of the algorithm is marked with vertical dashed lines. The top (blue) series shows the payoff obtained by the placement generation procedure. The bottom (red) series shows the payoff obtained by the attack generation procedure. The middle (green) series shows the payoff values for the optimal probability distribution, p^* and q^*, for current sets $\mathcal{A}$, $\mathcal{S}$. For the middle (green) series, in each iteration there're two measurements shown: first, the result of the model $\mathbb{A}[\mathcal{A}, \mathcal{S}]$, and second, the result of the model $\mathbb{P}[\mathcal{S}, \mathcal{A}]$. As one can observe, within the first iteration, the addition of the new placement increases the following middle (green) payoff value from 7 to 14. On the other hand, the addition of the attack (at the end of the first iteration) decreases the following middle (green) payoff value from 14 to 7. In the second iteration, the situation is different. Adding a new placement results in a top (blue) payoff of 14, but re-computation of placement probabilities procedure gives the middle (green) payoff of 7. However, in general, the addition of a new placement in most of the cases increases the middle (green) payoff, while the addition of a new attack decreases the middle (green) payoff, what is reflected in the figure. Nevertheless, the addition of a new placement never decreases, and the addition of a new attack never increases, the middle (green) payoff. As expected, the middle (green) payoff (the payoff we are searching for) cannot exceed the top payoff, and will always be not lower than the bottom payoff.

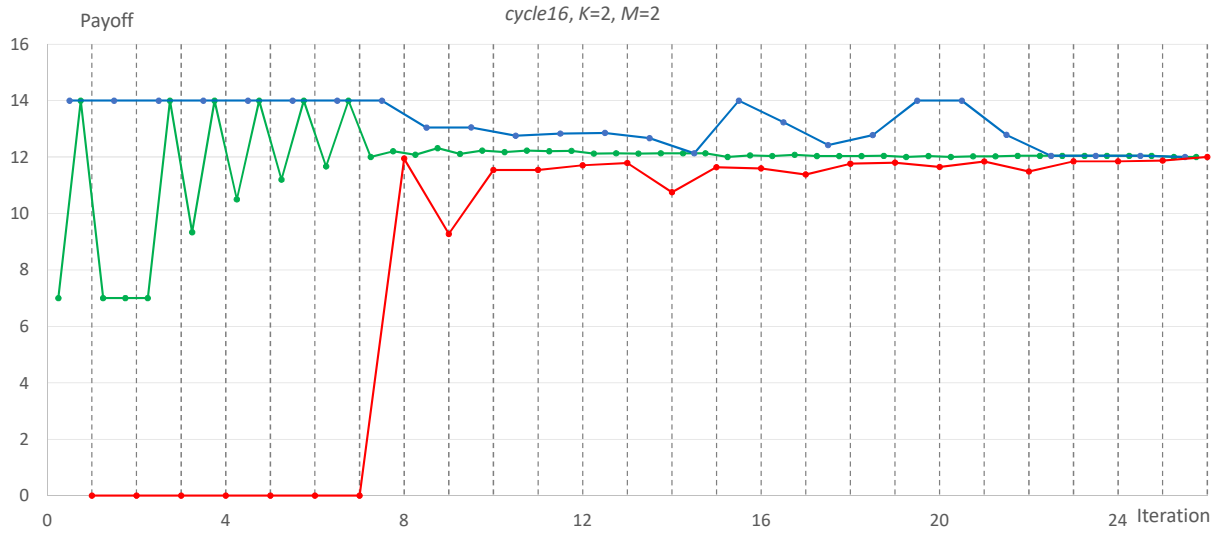


Figure 2.13: Network "*cycle16*": convergence of the Algorithm 1 for $K = 2$, $M = 2$

Figure 2.14 presents search of the solution for network "*cost266*" for $K = 3$ and

$M = 3$, while Figure 2.15 presents search of the solution for network "*cost266*" for $K = 4$ and $M = 3$.

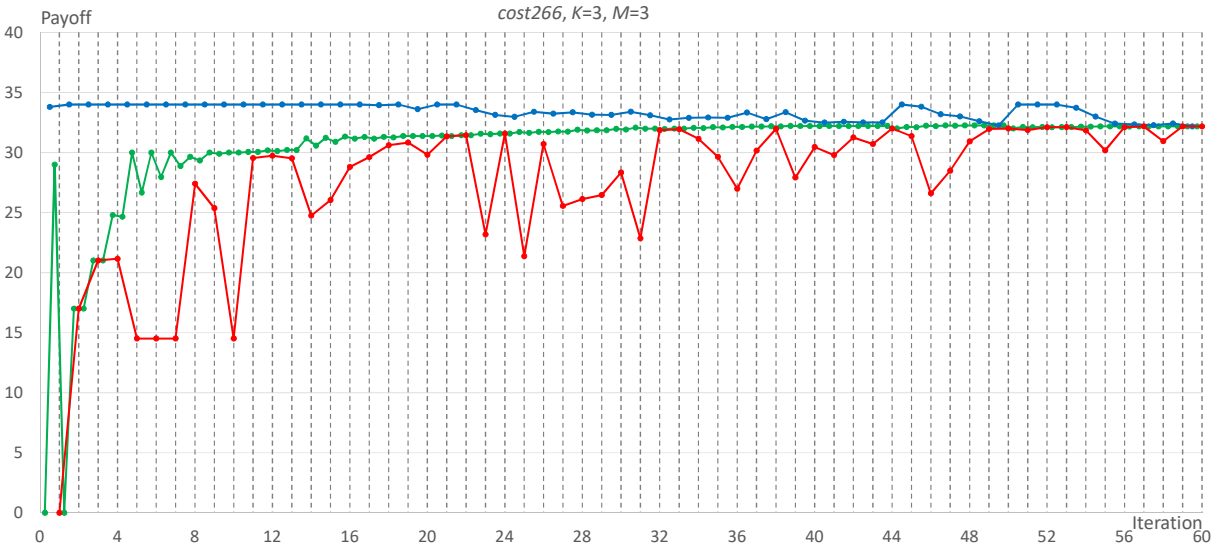


Figure 2.14: Network "*cost266*": convergence of the Algorithm 1 for $K = 3$, $M = 3$

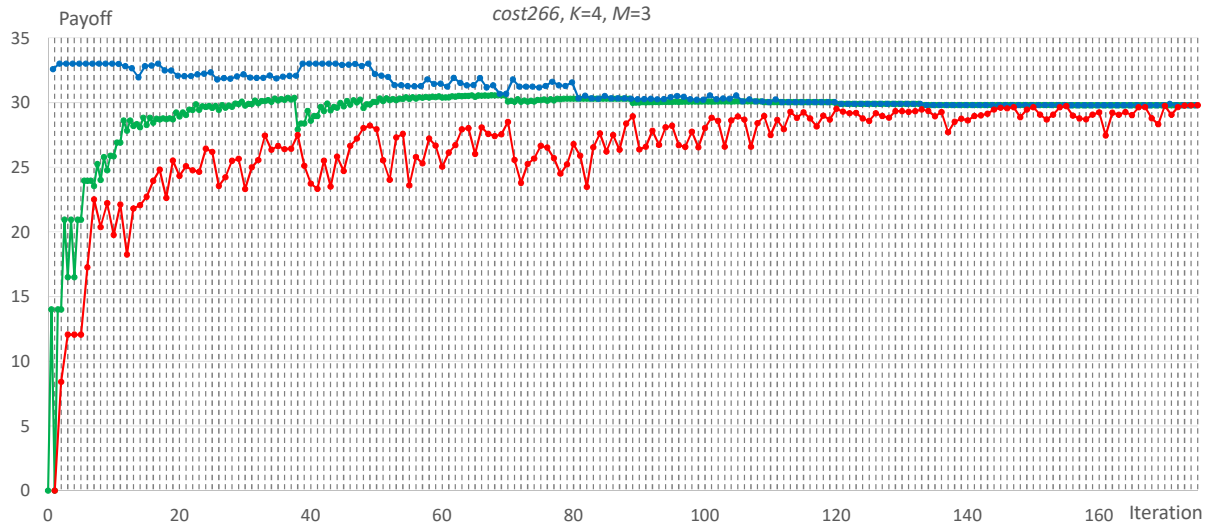


Figure 2.15: Network "*cost266*": convergence of the Algorithm 1 for $K = 4$, $M = 3$

Figure 2.16 presents search of the solution for network "*coronet conus*" for $K = 3$ and $M = 2$, while Figure 2.17 presents search of the solution for network "*coronet conus*" for $K = 3$ and $M = 3$.

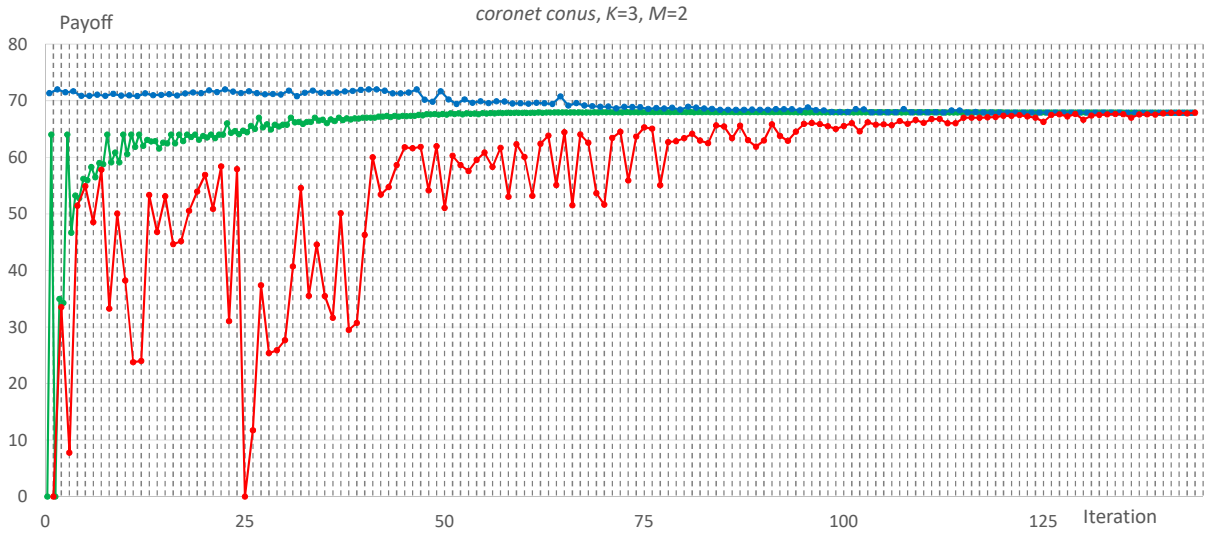


Figure 2.16: Network "*coronet conus*": convergence of the Algorithm 1 for $K = 3$, $M = 2$

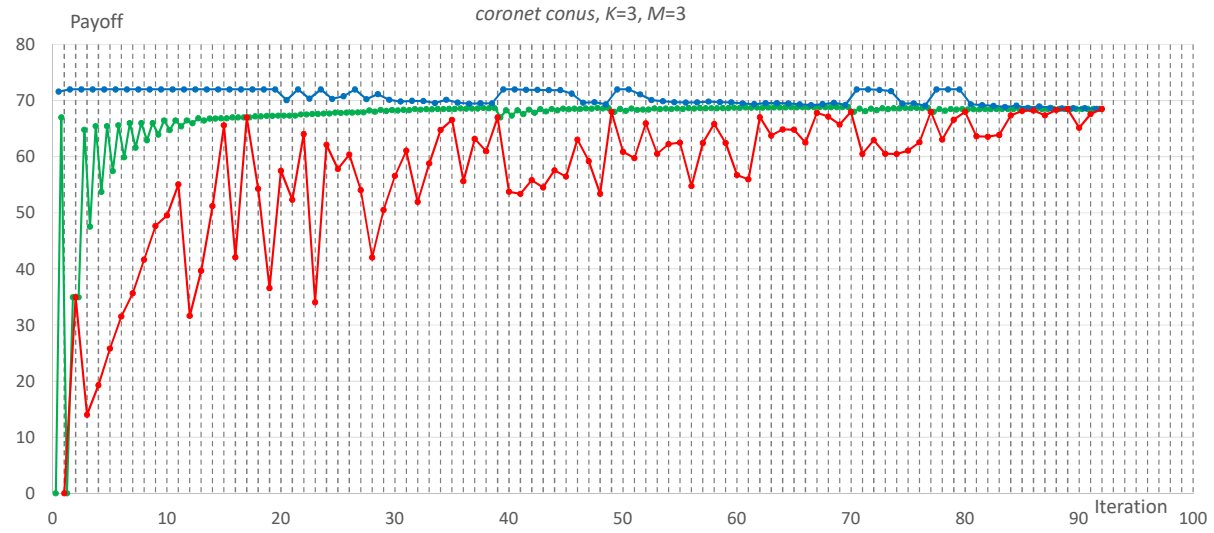


Figure 2.17: Network "*coronet conus*": convergence of the Algorithm 1 for $K = 3$, $M = 3$

2.6 Conclusions

The computational results reveal that:

- Optimal mixed strategies are more efficient than the MM strategy for both players, especially for the network operator.
- The average payoffs and probabilities for which they are achieved can be effectively computed with the column generation approach.

- The sizes of the sets of placements and attacks generated by Algorithm 1 are much smaller than $|\mathcal{S}(M)| = \binom{V}{M}$ and $|\mathcal{A}(K)| = \binom{V}{K}$. Additionally, it should be noted, that the number of non-zero strategies computed for both players, the operator and the attacker, is much smaller than the size of the payoff matrix, for any M and K.

- Starting the algorithm with the worst topological attacks reduces the computation time compared to starting with a random attack and random placement. The computed payoff does not depend on initial attacks/placements.

As for future work, the following extensions to our optimization model are worth considering:

- Adding other availability measures, such as the total number of node pairs in surviving components.

- Consideration of constraints on node-to-controller and controller-to-controller delays.

- Making a distinction between primary and backup controllers.

The other elements might also be further studied:

- The convergence results of the algorithm suggest that the algorithm reaches a payoff close to optimal quite quickly, but this is an invalid conclusion, and interrupting the algorithm for one of the players (and allowing the other player to search) could lead to a solution far from the one obtained by our algorithm. However, the stop condition may be analyzed in more detail to reduce the number of algorithm iterations (e.g. by slightly compromising the precision).

- Analyzing the consequences of abandoning a low-probability strategy (e.g. close to zero), for the players, the attacker, and the operator.

Chapter 3

Traffic Sentinel Placement

3.1 Introduction

3.1.1 Sensor placement against DDoS attacks

One of the ways to defend against a DDoS attack is to place traffic sentinels (sensors, in short) in the network which recognize and stop unauthorized demands. However, placing such sensors in every node of the network would be very expensive and inefficient. Commercial IPS (Intrusion Prevention Systems)/Firewalls solutions that detect and eliminate DDoS attacks have a high acquisition price (Fayaz et al. [39], Blazek et al. [16]). Hence, a natural question arises concerning what the number of sensors should be, and where they should be placed. The detection precision may be higher closer to attack sources since it is easier to detect spoofed addresses and other anomalies. On the other hand, the traffic closer to targets is large enough to accurately recognize an actual flooding attack. In order to efficiently control the flooding, sensors should be placed in the core of the network, where most of the traffic can be observed. A taxonomy of defense mechanisms against DDoS flooding attacks - including source-based, destination-based, network-based, and hybrid (a.k.a. distributed) defense mechanisms is discussed by Zargar et al. [133].

Defrawy et al. [35] formulate the problem of the optimal allocation of DDoS filters. They model single-tier filter allocation as a 0-1 knapsack problem and two-tier filter allocation as a cardinality-constrained knapsack. However, both models assume a single victim, while the models in this study allow for multiple victims.

Armbruster et al. [9] analyze the problem of packet filter placement to defend a

network against spoofed denial of service attacks. They examine the optimization problem (NP-hard) of finding a minimum cardinality set of nodes (filter placements) that filter packets so that no spoofed packet (with forged origin) can reach its destination. They relate the problem to the vertex cover problem and identify topologies and routing policies for which a polynomial-time solution to the minimum filter placement problem exists. They prove that under certain routing conditions a greedy heuristic for the filter placement problem yields an optimal solution. The paper addresses specific version of DDoS - a *Spoofed attack*.

Jeong et al. [58] and Islam et al. [55] minimize the number of sensors such that every path of a given length (r) contains a sensor. Any node less than r hops away is permitted to attack another node, since the impact of the attack is regarded as low, especially for a low r. This paper considers the problem of sensor placement under a different assumption.

Fayaz et al. [39] propose a Bohatei system for DDoS defense within a single ISP (Internet Service Provider). They use modern network architectures - software-defined networking (SDN) and network function virtualization (NFV) and develop the system orchestration capability to defend against a DDoS. The system addresses a resource management problem (NP-hard) to determine the number and location of defense VMs (Virtual Machines). These VMs detect and block attack traffic. After VMs are fixed, the system routes the traffic through these VMs. The goal of the resource manager is to efficiently assign available network resources to the defense, (1) minimizing the latency experienced by legitimate traffic, and (2) minimizing network congestion. The authors formulate an Integer Linear Program (ILP) to solve the resource management problem. However, due to the long computation time they apply hierarchical decomposition as well. For that purpose, they designed two heuristics, the first for data-center selection, and the second for server selection at the data-center. When it comes to routing, this paper doesn't assume any specific routing protocol, it simply assumes that it is multi-path. Additionally, traffic is not steered through a network; it is assumed that routing is an independent problem.

Mowla et al. [82] assume SDN architecture for their proposal. They propose a cognitive detection and defense mechanism to distinguish DDoS attacks and Flash Crowd traffic. The detection sensors are placed in the OpenFlow Switches, where approaching traffic is identified and specific features are extracted. The extracted data is handed over to the SDN controller for analysis and production of security rules to defend against the attack.

They use two classification techniques, namely SVM and Logistic Regression. It must be noted that such an approach has its drawbacks specifically, a centralized SDN controller is a potential single-point-of-failure (security risk).

Ramanathan et al. [104] propose a collaboration system SENSS to protect against DDoS. SENSS enables the victim of an attack to request an attack monitoring and filtering on demand from an ISP. Requests can be sent both to the immediate and to remote ISPs, where SENSS servers are located. The victim drives all the decisions, such as what to monitor and which actions to take to mitigate attacks (e.g., monitor, allow, filter). The number and location of monitoring sensors is not thoroughly analyzed in the research. For certain types of attack (direct floods without transport/network signature), the article suggests a location-based filtering approach that compares traffic volumes for ISP-ISP links during normal operation and during an attack.

Monnet et al. [81] place control nodes (CN) in a clustered WSN (Wireless Sensor Network). CN detects abnormal behavior (DoS) and reports it to a cluster leader up in the WSN hierarchy. The authors propose three methods of CN placement. The first uses a distributed self-election process. A node chooses a pseudo-random number, checks the number against the threshold and potentially self-elects itself as a CN. The second method is based on the residual energy of nodes. Cluster heads select nodes with the highest residual energy. The third method is based on democratic election. Nodes vote for the nodes that will be selected as a CN.

A related problem, the design of sensor networks for measuring the surrounding environment (natural floods, pollution etc.), is addressed in many works. Khapalov [68] addresses source location and sensor placement in environmental monitoring. The first problem here is linked to finding an unkown contamination source. The second concerns the placement of sensors to obtain adequate data. Ucinski [120] focuses on the design of a monitoring sensor network to provide proper diagnostic information about the functioning of a distributed parameter system. Patan [91] determines a scheduling policy for a sensor network monitoring a spatial domain in order to identify unknown parameters of a distributed system. Suchanski et al. [113] study the dependency between density of a sensor network and map quality in the radio environment map concept. There have been a large number of works on developing methods and technology of person's activity recognition and monitoring. Some use wearable devices to collect vital sign signals, some use video

analysis and an accelerometer to recognize the activity pattern, other use thermal sensors. Chou et al. [25] develop a framework to measure gait velocity (walking speed) using distributed tracking services deployed indoors (home, nursing institute). The work aims to minimize the sensing errors caused by thermal noise and overlapping sensing regions. The other goal is to minimize the data volume to be stored or transmitted. One fundamental question is how many sensors should be deployed and how these sensors work together seamlessly to provide accurate gait velocity measurements.

In the literature there is well-known class of interdiction problems, which can be related to our DDoS problem. Altner et al. [7] study the Maximum Flow Network Interdiction Problem (MFNIP). In the MFNIP a capacitated $s-t$ (directed) network is given, where each arc has a cost of deletion, and a budget for deleting arcs. The objective is to choose a subset of arcs to delete, without exceeding the budget, that minimizes the maximum flow that can be routed through the network induced on the remaining arcs. The special case of the MFNIP when an the interdictor removes exactly k arcs from the network to minimize the maximum flow in the resulting network is known as the Cardinality Maximum Flow Network Interdiction Problem (CMFNIP) (Wood [128]). One of the recent works on the interdiction problem addresses a two-stage defender-attacker game that takes place on a network whose nodes can be influenced by competing agents (Hemmati et al. [50]). A more general problem on graphs was proposed by Omer and Mucherino [89], and it includes the interdiction problem. In our DDoS problem we delete vertices instead of arcs in the CMFNIP.

Defense mechanisms against DDoS flooding attacks address specific attack types: *link-flooding* (Studer and Perrig [112], Kang et al. [65]) or *target-flooding* (Zargar et al. [133]). *Link-flooding* attacks aim at intermediate network links located between attack sources and targets. *Target-flooding* directly attack targets. This research concentrates on the latter one. The attacks may use *reflection* (Ramanathan et al. [104]), *spoofing* (Armbruster et al. [9]) or other techniques (Zargar et al. [133]). The existing works concentrate on single-target while we concentrate on multiple-target attacks. The defense mechanisms against DDoS are complex systems. They need to address: identification of attack signatures and detection algorithms (out of scope of this paper), placing the detection sensors, and stopping/filtering illegitimate traffic (Ramanathan et al. [104]) (out of scope of this paper). Some defense approaches use attack load distribution (re-routing of traffic) to

limit the effect on targets (Belabed et al. [13]). In this paper, the focus is on the placing of detection sensors. There are several works in this field: Jeong et al. [58] and Islam et al. [55] minimize the number of sensors such that every path of a given length (r) contains a sensor; Armbruster et al. [9] analyze the problem of packet filter placement to defend a network against spoofed denial of service attacks; Monnet et al. [81] place control nodes in clustered WSN to save the energy of nodes; Fayaz et al. [39] address the resource management problem to determine the number and location of defense VMs, which combines detection node placement with a re-routing strategy. This paper concentrates on the costly deployment of detection sensors (probes) against multiple-target flooding attacks. There is no assumption of any specific routing protocol, though it is assumed that it is multi-path. Additionally, traffic is not steered through a network; it is assumed that routing is an independent problem. Future work may address sensor placement with a knowledge of a specific routing protocol to increase performance in a network.

3.1.2 Our proposal

A DDoS attack can be modeled as a flow from multiple sources to a single target (single commodity flow). Defined are directed graph with a capacity function on edges, a set of sources (S) and a set of targets (T). An attacker can conduct an attack on any vertex $t \in T$. The strength of an attack is given by a value of a $\mathsf{maxflow}_G(S,t)$, i.e., the value of the maximum flow from S to t in the network G.

Within this DDoS defense approach sensors are to be placed in network nodes to recognize and stop unwanted traffic. If a sensor is placed in a vertex $v \in V$ then all the edges incident to v are assumed controlled. A set $D \subseteq V$ is called a set of sensors. The goal of this defense is to limit maximum uncontrolled flow towards each $t \in T$. Having a placement D, a maximum uncontrolled flow is determined and easy to compute. For that purpose, for each $t \in T$ max-flow algorithm (see for example (Goldberg and Tarjan [47])) can be used for a graph $G \setminus D$ ($|T|$ runs of the algorithm). A super vertex ss is added to G, connected with a directed edge to each $s \in S$. For each run of the algorithm ($t \in T$) maximum flow from ss to t is computed. Finally, maximum uncontrolled flow as $\max_{t \in T} \mathsf{maxflow}_G(ss,t)$ is computed.

In this chapter, the proof is given of the decision problem as to whether d sensors

suffice to reduce uncontrolled flow to defined amount $a \in \mathbb{R}$. When there is just one protected node, the proof is based on a reduction from Cardinality Maximum Flow Network Interdiction Problem (CMFNIP) (Wood [128]). When the number of pairs (S, t_i) is more than one, the reduction goes from Multiway Cut (Garg et al. [43]).

For computational reasons, two variants of the sensor placement problem are given. First, the PQ problem, where a tolerable amount $a \in \mathbb{R}$ of uncontrolled flow is set, and a minimum number of sensors needed to achieve it is required. Second, the PC problem, where the number of sensors is set, and the question as to how much uncontrolled flow we can reduce with such a number of sensors is asked.

The main result of this work, presented in this chapter, besides the proofs of NP-hardness, are two mixed integer models describing PQ and PC problems of optimal sensor placement against DDoS attacks. Moreover, two efficient heuristics (one for each problem) are presented. Finally, an experimental comparison of solutions given by the heuristics and the mixed-integer programming solvers is given.

3.2 Problem definition

3.2.1 The problem of optimal sensor placement

Network model: It is assumed that the network is modeled as a directed graph without multiple edges. The node (vertex) set and the edge set are denoted, respectively, by V and E. Every directed edge has a nonnegative capacity assigned by the function c. Each node in the network can be interpreted as a router or an autonomous system.

Protected nodes: Let $T \subseteq V$ denote a set of *protected* nodes (a.k.a target nodes) in the network. Each node $v \in T$ contains a protected resource and is a target of a possible malicious flow.

Attack sources: We assume that network flooding targeted at protected nodes $t \in T$ can start from any network node (*source*) $s \in V \setminus T$. In a practical scenario, however, it may be desirable to limit our attention to a set of sources $S \subseteq V \setminus T$. The selection may be based on a node's risk analysis. It is simply a case of choosing the vertices with unacceptable risk.

Attacks: It is not assumed which traffic from a source $s \in S$ is legitimate and which is hostile. Every potential attack starts from S and is modeled as a single-commodity flow

to some target $t \in T$. Routing policies allow multi-path transmissions from any $s \in S$ to t.

Sensors: When a sensor is placed at a node $v \in V$, then all the incoming and outgoing edges are assumed controlled. A set of nodes where sensors are placed is called D. For clarity of NP-completeness proofs it is assumed that the set D is disjoint with $S \cup T$. However, in practice this assumption can be easily omitted by adding artificial copies for each source and target and joining it with the original vertex (see Figure 3.2 and 3.3).

Definition 1. Attack flow For $t \in T$, a function $f_t : E \to [0, \infty)$ is called an *attack flow on* $t \in T$ (or just flow, if t is clear from the context) if both the following conditions are satisfied:

$$\forall_{u \in V \setminus (S \cup \{t\})} \sum_{(v,u) \in E} f_t(v,u) = \sum_{(u,w) \in E} f_t(u,w), \tag{3.1}$$

and

$$\forall_{e \in E}\ f_t(e) \leq c(e). \tag{3.2}$$

The attack flow value is given by

$$f_t = \sum_{(v,t) \in E} f_t(v,t) - \sum_{(t,w) \in E} f_t(t,w). \tag{3.3}$$

The maximum value of an attack flow on t is denoted by $\mathsf{maxflow}_G(S,t)$.

Definition 2. G \ D For an instance $G = (V, E, c, S, T)$ and a set $D \subseteq V \setminus (S \cup T)$ of sensors, by $G \setminus D$ we denote the instance $G' = (V, E, c', S, T)$, where $c' : E \to [0, \infty)$ is defined as follows:

$$c'(e) = \begin{cases} 0, & \text{if } e \in E_D, \\ c(e), & \text{otherwise,} \end{cases}$$

where E_D is the set of edges incident to a node in D.

Definition 3. Uncontrolled flow For an instance G and a set D of sensors, an *uncontrolled flow to* $t \in T$ is a flow to t in $G \setminus D$ with positive value.

For example, in Figure 3.1 all edges incident to nodes 5 and 7 are controlled. However, there still exists an uncontrolled flow f_8 in $G \setminus \{5, 7\}$.

In order to defend against a DDoS attack, sensors in a network should be placed in such a way that they can observe all or most of the traffic coming from sources S to targets

T. Placing sensors in every node of the network would be very expensive and inefficient. Having a limited number of sensors available, it is necessary to find a placement such that uncontrolled flows are "distributed" among all $t \in T$. The situation in which some targets are left unprotected and receive a high portion of an uncontrolled traffic, so in consequence are vulnerable to DDoS attacks, should be avoided.

In the optimization variant two models PQ (Placement with required Quality) and PC (Placement with required Cardinality) are considered. In the ***PQ*** model, we want to minimize the number k of sensors under the assumption that the amount of uncontrolled flow does not exceed a given value. Formally, for a given number $a \in \mathbb{Q}$, it is asked what a minimum integer k is such that there exists a k-element set $D \subseteq V \setminus (S \cup T)$ such that

$$\max_{t \in T} \mathsf{maxflow}_{G \setminus D}(S, t) \leq a.$$

For $a = 0$ the question follows: what is the minimum number of sensors that guarantees the total control in the network.

In the second model, denoted by ***PC***, it is assumed the number k of sensors and the task is to find a k-element set $D \subseteq V \setminus (S \cup T)$ such that $\max_{t \in T} \mathsf{maxflow}_{G \setminus D}(S, t)$ is minimum. Such a model is important from a practical perspective. In many cases the number of available sensors is limited and one needs to find an optimal placement.

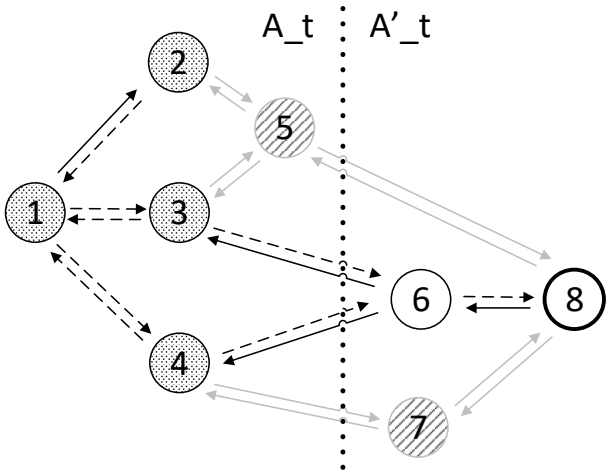


Figure 3.1: An instance G with source (attack) nodes $S = \{1, 2, 3, 4\}$, protected nodes $T = \{8\}$ and sensors $D = \{5, 7\}$. The dotted vertical line denotes a possible *cut* for $t = 8 \in T$. Dashed lines denote the *uncontrolled flow* f_8.

3.2.2 Complexity of the optimal sensor placement

For the complexity analysis a decision problem FLOW PREVENTION is defined:

Input: directed graph $G = (V, E)$, capacity function $c : E \to [0, \infty)$, disjoint sets $S, T \subseteq V$, integer k, real number a,

Question: Does there exist a set $D \subseteq V \setminus (S \cup T)$ of size at most k, such that for every $t \in T$ it holds that $\mathsf{maxflow}_{G \setminus D}(S, t) \leq a$?

The problem has several natural parameters, including k, a, $|S|$, and $|T|$. Its complexity is studded under different combinations of these parameters.

First, simple boundary cases. If $a = 0$, then the problem asks for an S-T-separator of size at most k and thus can be solved in polynomial time using standard flow techniques. If k is a constant, then the problem can be solved in polynomial time by exhaustive enumeration combined with finding the maximum flow.

Now, consider the case that $|T| = 1$. This will show a reduction from CMFNIP, which is known to be NP-hard (Wood [128]). An instance of this problem is a graph $G = (V, E)$ with edge capacities $c : E \to [0, \infty)$, two distinct distinguished vertices $s, t \in V$, an integer k and a real a. The question is whether we can remove at most k edges so that the maximum s-t-flow in the resulting graph is at most a. Observe that the difference between this problem and FLOW PREVENTION is that *nodes*, not *edges*, are removed.

Theorem 1. *FLOW PREVENTION is* NP*-complete, even if* $|S| = |T| = 1$.

Proof. Let $(G = (V, E), c, s, t, a, k)$ be an instance of CMFNIP. Let $\bar{G} = (\bar{V}, \bar{E})$ be the graph obtained from G in the following way. For every $v \in V$ we create its $k + 1$ copies $v_1, v_2, .., v_{k+1}$. For every arc $e = (u, v) \in E$ we define two vertices e_u, e_v and edges: $u_1e_u, u_2e_u, \ldots, u_{k+1}e_u, e_ue_v, e_vv_1, e_vv_2, \ldots, e_vv_{k+1}$. Moreover we add vertices s_0, t_0 and edges $s_0s_1, s_0s_2, \ldots, s_0s_{k+1}$, $t_1t_0, t_2t_0, \ldots, t_{k+1}t_0$. We set $S = \{s_0\}$ and $T = \{t_0\}$. Finally, we define the capacity function $\bar{c}$ as follows. For $e \in E$, we set $\bar{c}(e_ue_v) = c(e)$, and the capacities of all other arcs of $\bar{G}$ are set to some large integer, e.g., $\sum_{e \in E} c(e)$. Observe that $\mathsf{maxflow}(\bar{G}, s, t) = \mathsf{maxflow}(G, s, t)$. Furthermore, since our budget is only k, it makes no sense to remove any copy of a vertex v of G, and there will always be at least one copy left. Finally, for $e = (u, v) \in E$, removing e_u or e_v in $\bar{G}$ corresponds to removing e in G, and it is sufficient to remove one of these vertices. Summing up, it is straightforward

to verify that $(\bar{V}, \bar{E}, \bar{c}, S, T, k, a)$ is a yes-instance of Flow Prevention if and only if (G, c, s, t, k, a) is a yes-instance of CMFNIP. □

Now consider the case that $|T| \geq 2$. This time we will reduce from Node Multiway Cut with 2 terminals, which is known to be NP-hard (Garg et al. [43]). In this problem we are given a directed graph G with two distinguished vertices x, y and an integer k. We ask whether we can remove at most k vertices to destroy all x-y- and all y-x-paths.

Theorem 2. *Flow Prevention is* NP*-complete, even if* $a = 1$, $|S| = |T| = 2$, *and all capacities are unit. Furthermore, it is even* NP*-hard to distinguish yes-instances and those for which, for every set* D' *of size at most* k, *it holds that*

$$\max_{t \in T} \mathsf{maxflow}_{G \setminus D'}(S, t) = 2.$$

Proof. Let $G = (V, E)$, x, y, k, be an instance of Node Multiway Cut with 2 terminals. We may safely assume that G contains a directed $x - y$-path and a directed $y - x$-path, as otherwise the problem can be solved in polynomial time by finding a minimum vertex separator.

We construct an instance of Flow Prevention as follows. We start with a graph G. Next we add two new vertices x' and y', and edges $x'x, y'y$ with unit capacity. We set $S = \{x', y'\}$ and $T = \{x, y\}$.

We observe that for every $t \in T$ it holds that $\mathsf{maxflow}_G(S, t) = 2$, as G contains a directed $x - y$-path and a directed $y - x$-path. Furthermore, for $D \subseteq V \setminus (S \cup T)$, it holds that $\max_{t \in T} \mathsf{maxflow}_{G \setminus D}(S, t) = 1$ if and only if D is a multiway cut in G. □

Corollary 1. *The following optimization problem admits no polynomial-time 2-approximation algorithm, unless* P = NP.

Input: *directed graph* $G = (V, E)$, *disjoint sets* $S, T \subseteq V$, *integer* k,

Question: *What is the minimum* a, *for which there is some* $D \subseteq V \setminus (S \cup T)$ *of size at most* k, *such that for every* $t \in T$ *it holds that* $\mathsf{maxflow}_{G \setminus D}(S, t) \leq a$?

Finally, let us consider parameterization by k. The problem is clearly in XP (i.e., can be solved in polynomial time if k is fixed), so it is interesting if the problem is FPT (i.e., can be solved in time $f(k) \cdot n^{O(1)}$ on instances of size n, where f is some computable

function) and, if so, if it admits a polynomial kernel. See Gygan et al. [28] for more information about parameterized complexity classes.

Let us point out that a natural generalization of the problem is not in FPT under standard complexity assumptions. Consider a variant of Flow Prevention where to each sink $t \in T$ we have assigned a possibly distinct set S_t of sources, and we ask if there is a set $D \subseteq V \setminus \bigcup_{t \in T}(S_t \cup \{t\})$ of size at most k, such that for every $t \in T$ it holds that $\mathsf{maxflow}_{G \setminus D}(S_t, t) \leq a$. It turns out that this problem is W[1]-hard, even if $a = 0$, $|T| = 4$, and $|S_t| = 1$ for every $t \in T$. Indeed, one can readily verify that the problem is equivalent to the well known Node Multicut problem. The instance of this problem is a directed graph G, a set of pairs of vertices $(s_i, t_i)_{i=1}^{p}$ called terminals, and an integer k. The question is whether we can remove at most k nonterminal vertices so that in the resulting graph there is no s_i-t_i path, for any i. As shown by Pilipczuk and Wahlström [98], this problem is W[1]-hard even for $p = 4$. This is a strong evidence that the problem is not in FPT (Cygan et al. [28]).

3.3 Model description

3.3.1 Basic formulation of *PQ* and *PC* models

To solve the problem of optimal sensor placement in the sense of models PQ and PC we use mix-integer programming. Our solution is based on a well-known Ford-Fulkerson Theorem [41] stating that the maximum flow cannot exceed the minimum cut and actually, in our solution the min-cuts are minimized. To compute minimum cuts for every target $t \in T$ we introduce a set A_t such that any edge u, v is in a cut for t if and only if $u \in A_t$ and $v \notin A_t$ (Figure 3.1). The set $D \subseteq V$ denotes the set of vertices in which sensors are placed.

Formally, we define the following variables:

- For every $v \in V$ a binary variable $d[v]$ with the meaning $d[v] = 1$ if and only if $v \in D$ (there is a sensor in the vertex v).
- For every $t \in T$ and $v \in V$ a binary variable $a[t, v]$ with the meaning $a[t, v] = 1$ if and only if $v \in A_t$. The sets A_t allow us to compute a cut for the target $t \in T$.

- For every $t \in T, e \in E$ a binary variable $cutT[t, e]$ with the meaning $cutT[t, e] = 1$ if and only if $e \in E$ belongs to a cut in $G \setminus D$ for t.
- A real variable $M \in \mathbb{R}$, that denotes the value of the minimum cut in $G \setminus D$.

In ***PQ*** model, a function to minimize is $\sum_{v \in V} d[v]$ with respect to the below restrictions (3.4)(3.5)(3.6)(3.7)(3.8)(3.9). The meaning of restrictions is as follows. For every target $t \in T$ each vertex $s \in S$ belongs to A_t (3.4). For every target $t \in T$ the vertex t does not belong to A_t (3.5). The restriction (3.6) guarantees that an edge belongs to a cut if none of its ends is in a set D, the first vertex is in A_t and the second vertex is not. The equation (3.7) bounds the value of the cut with $a = (1 - q) \cdot \max_{t \in T} \mathsf{maxflow}_G(t)$, where $q \in [0, 1]$ is a quality factor (parameter to the problem), $q = 1$ signifies total control (100% traffic controlled), $q = 0$ signifies no control (zero sensors placed); and $\max_{t \in T} \mathsf{maxflow}_G(t)$ is equal to the value of max minimum cut M_t in G. The restrictions (3.8)(3.9) make sure that sensors cannot be placed in either $s \in S$ or $t \in T$. Obviously, the above statement which assumes 100% control of traffic ($q = 1$) gives a theoretical value, while in practice it depends on the volume of traffic flowing via links, and on the processing capacity of a detection sensor (technology).

$$\forall_{t \in T} \ \forall_{s \in S} \ \ a[t, s] == 1 \tag{3.4}$$

$$\forall_{t \in T} \ \ a[t, t] == 0 \tag{3.5}$$

$$\begin{aligned} &\forall_{t \in T} \ \forall_{(u,v) \in E} \\ &cutT[t, u, v] \geq a[t, u] - a[t, v] - d[u] - d[v] \end{aligned} \tag{3.6}$$

$$\forall t \in T \sum_{(u,v) \in E} cutT[t, u, v] \cdot c[u, v] \leq a \tag{3.7}$$

$$\forall s \in S \ \ d[s] = 0 \tag{3.8}$$

$$\forall t \in T \ \ d[t] = 0 \tag{3.9}$$

In ***PC*** model, a function to minimize is just M with respect to the restrictions (3.4)(3.5)(3.6)(3.8)(3.9)(3.10)(3.11). The meaning of restrictions is as follows. The restriction (3.10) makes sure that the number of sensors is fixed, and given as parameter k to the problem. The equation (3.11) bounds the value of the cut with M.

$$\sum_{v \in V} d[v] = k \tag{3.10}$$

$$\forall t \in T \quad \sum_{(u,v) \in E} cutT[t,u,v] \cdot c[u,v] \leq M \tag{3.11}$$

As shown in Section 3.5 (Computational results), the above models are very efficient in terms of the number of deployed sensors and a volume of uncontrolled flow. On the other hand, when the number of vertices is high (large scale networks) the models may suffer from increased execution time. That is why we designed and implemented two efficient heuristics (one for each model, Section 3.4). They are reasonably efficient in terms of a goal value, but much faster than the models.

3.4 Algorithm description

3.4.1 Relaxed formulation of *PQ* and *PC* models

In this formulation we relax two types of variables to allow the fractional sensor placement (first bullet) and fractional traffic control (second bullet). Let us notice that fractional sensor placement is an artificial concept without physical interpretation and defined only as an intermediate step, not present in the final step of the algorithm.

- For every $v \in V$ a real variable $d[v] \in [0,1]$
- For every $t \in T, e \in E$ a real variable $cutT[t,e] \in [0,1]$.

In the basic model formulation (Section 3.3) when an edge u, v is in a cut for some t ($u \in A_t$ and $v \notin A_t$), placing a sensor in either u or v classifies such an edge as fully controlled. When no sensor is placed in either u nor v such an edge is uncontrolled. However, in the relaxed formulation we allow fractional sensor placement (d variables) and fractional control of edges in a cut ($cutT$ variables).

To solve the *PQ* and *PC* problems, additionally to our two models (Section 3.3), we have designed and implemented two algorithms:

- *PQIterativeBestSensor* (see Algorithm 2)
- *PCIterativeBestSensor* (see Algorithm 3).

Both algorithms assume the following common *input* parameters: G graph representing a network with c capacity function, T set of targets and S set of sources. Additionally, *PQIterativeBestSensor* heuristic takes q (quality factor) as input and *PCIterativeBestSensor* heuristic k (number of sensors) as input.

3.4.2 PQ Iterative Best Sensor Placement

The preparatory step of the algorithm *PQIterativeBestSensor* is a computation of the value of $a = (1 - q) \cdot \max_{t \in T} \mathsf{maxflow}_G(t)$ (line 1). In each while loop, linear program relaxation is solved (line 5). From the relaxed LP solution a subset of vertices L is selected from the set $V \setminus D$ such that $d[v] \neq 0$ and $d[v] == \max\{d[j]\}_{j \in V \setminus D}$ (line 6). Among the $|L|$ best sensor locations, the single best (max) one $v_{\max}$ is selected and added to the model as a constraint (line 8). The constraint fixes a sensor in the location $v_{\max}$ in the next iterations.

3.4.3 PC Iterative Best Sensor Placement

The algorithm *PCIterativeBestSensor* constitutes $k + 1$ iterations. In each $\{1, .., k\}$ iteration, linear program relaxation is solved (line 4). From the relaxed LP solution a subset of vertices L is selected from the set $V \setminus D$ such that $d[v] \neq 0$ and $d[v] == \max\{d[j]\}_{j \in V \setminus D}$ (line 5). Among the $|L|$ best sensor locations, the single best (max) one $v_{\max}$ is selected and added to the model as a constraint (line 7). The constraint fixes a sensor in the location $v_{\max}$ in the next iterations.
In the last iteration, the LP relaxation is solved assuming fixed sensor placements for all $v \in D$ (line 10) to compute the final value of M.

We show that the algorithm *PCIterativeBestSensor* may give a result $2 \cdot OPT$. In Figure 3.2 we compare the optimal solution OPT given by *PC* model (a) to the solution given by *PCIterativeBestSensor* (b)(c). We assume two sources $S = \{1, 2\}$ and two targets

Algorithm 2 PQIterativeBestSensor

Require: G, c, T, S, q

1: Compute a value of $a = (1 - q) \cdot \max_{t \in T} \mathsf{maxflow}_G(t)$
2: Create the relaxed *PQ problem* (Section 3.4.1) with goal *minimize* $\sum_{v \in V} d[v]$. Add constraints {(3.4),(3.5),(3.6),(3.7),(3.8),(3.9)} to the *problem*
3: Initiate a set of vertices in which we place sensors $D = \emptyset$
4: **while** $(\exists t \in T \sum_{(u,v) \in E} cutT[t, u, v] \cdot c[u, v] > (1 - q) \cdot \max_{t \in T} \mathsf{maxflow}_G(t))$ **do**
5: Solve the *problem*
6: Let $L = \{v, \text{ s.t. } v \in V \setminus D \text{ and } d[v] \neq 0 \text{ and } d[v] == \max\{d[j]\}_{j \in V \setminus D}\}$
7: Choose randomly $v_{\max} \in L$, where probability of selecting an element $v_{\max}$ equals $\frac{1}{|L|}$
8: Add constraint $d[v_{\max}] == 1$ to the *problem*
9: $D = D \cup \{v_{\max}\}$
10: **end while**
11: **return** D

$T = \{7, 8\}$, and we require to place $k = 1$ sensors. The optimal solution is $M = 1$ (a). Then one fractional solution given by the heuristic with its corresponding rounding is given. The (b)(c) results in a sub-optimal solution $M = 2$, which is equal to $2 \cdot OPT$. An additional example where the algorithm *PCIterativeBestSensor* gives a result $\frac{3}{2} \cdot OPT$, is given in Figure 3.3.

However, for practical scenarios the heuristic exposes a solid ratio (see Chapter 3.5 Computational results).

3.5 Computational results

3.5.1 Experiment Setup

The following experiments compare efficiency of the models with the algorithms. The *PQ* model is compared with *PQIterativeBestSensor* algorithm, and the *PC* model with the *PCIterativeBestSensor* algorithm. The comparison assumes ideal (theoretical) sensors, which means that if a sensor is placed in a node it controls 100% of in/out traffic. However, in practice it depends on the volume of traffic flowing via links, and on the pro-

Algorithm 3 PCIterativeBestSensor

Require: G, c, T, S, k

1: Create the relaxed *PC problem* (Section 3.4.1) with goal *minimize* M. Add constraints {(3.4),(3.5),(3.6),(3.8),(3.9),(3.10),(3.11)} to the *problem*.
2: Initiate a set of vertices in which we place sensors $D = \emptyset$
3: **for** $i = 1, .., k$ **do**
4: Solve the *problem*
5: Let $L = \{v, \text{s.t. } v \in V \setminus D \text{ and } d[v] \neq 0 \text{ and } d[v] == \max\{d[j]\}_{j \in V \setminus D}\}$
6: Choose randomly $v_{\max} \in L$, where probability of selecting an element $v_{\max}$ equals $\frac{1}{|L|}$
7: Add constraint $d[v_{\max}] == 1$ to the *problem*
8: $D = D \cup \{v_{\max}\}$
9: **end for**
10: Solve the *problem* to compute M
11: **return** (D, M)

cessing capacity of a detection sensor (technology). In practice, for high volume networks, typically only selected samples are analyzed due to processing limitations.

The two models *PQ* and *PC* and two algorithms *PQIterativeBestSensor* and *PCIterativeBestSensor* were run with the use of CPLEX 12.10 for Python. Python 3.7 was utilized to implement heuristics and automate simulations. The simulations were run on a personal computer with 1.9GHz CPU, 16GB RAM and 64-bit Windows platform.

The experiments were conducted on the following types of grid networks: "*gridnet*V", where $V = \{64, 81, 100, 121, 144, 169, 196, 225, 256, 289\}$ indicates the number of vertices in a network. All these networks are directed graphs, with a single edge in each direction u, v and v, u. An example of a small grid network is demonstrated in Figure 3.4. Each vertex in a graph may correspond to a router or an autonomous system in a telecommunication network.

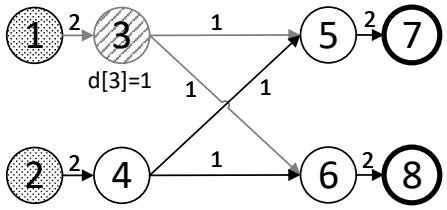
For simulation scenarios, for each network type, four random instances of each network type were generated, each with randomly selected capacities (c). Each edge capacity was randomly selected from the range $c(e)_{e \in E} \in [100, 200]$ (random selection with uniform distribution). Additionally, for each simulation scenario, four random instances of target locations ($T_{i=1..4} \subseteq V$) were generated (all vertices V have equal probabilities). For each

target instance T_i, four random instances of source locations were generated ($S_{j=1..4} \subseteq V \setminus T_i$) (all vertices $V \setminus T_i$ have equal probabilities). As a result, each value (volume of uncontrolled flow; execution time) presented on each diagram is an average computed from 64 measurements. Finally, we assumed the following number of targets and sources: Scenario1-4: $|T| = 10$, $|S| = 40$; Scenario1b,2b: $|T| = 10$; Scenario3b,4b: $|T| = 20$.

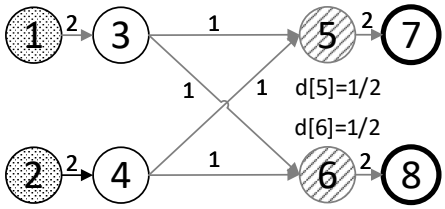
3.5.2 Scenario1 *PC* problem, "*gridnet100*" network, increasing number of sensors

The experiments were conducted for the grid network "*gridnet100*". The number of sensors was increasing from $k = 0$ to $k = 10$.

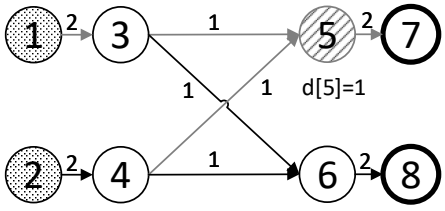
The diagram Figure 3.5 (a) demonstrates the average volume of uncontrolled traffic



(a) The model *PC*: M=1 (OPT)



(b) The heuristic *PCIterativeBestSensor* before rounding: M=1

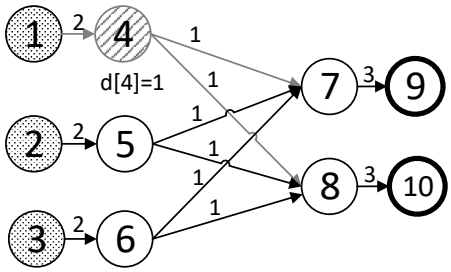


(c) The heuristic *PCIterativeBestSensor* after rounding: M=2

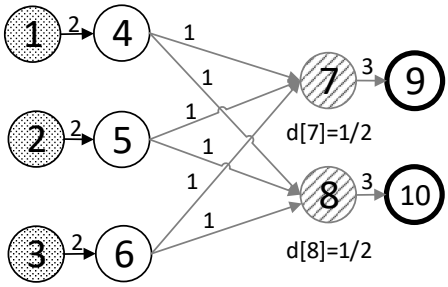
Figure 3.2: The algorithm *PCIterativeBestSensor* gives a result $2 \cdot OPT$ (solution (b)(c)), where M is a value of uncontrolled flow, $S = \{1, 2\}$, $T = \{7, 8\}$, $k = 1$, and D is defined by gray striped circles.

(y axis) depending on the number of sensors. As the number of sensors increases, the average volume of uncontrolled traffic decreases to zero (for $k = |T|$), for both *PC* model and *PCIterativeBestSensor* heuristic. The observed average objective values of *PCIterativeBestSensor* are higher than those of *PC* by up to 8%.

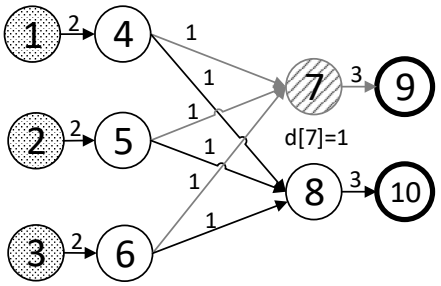
The diagram Figure 3.5 (b) demonstrates the average time of execution (y axis). The observed average values of execution time of *PC* are up to 10 times higher than those of *PCIterativeBestSensor*.



(a) The model *PC*: M=2 (OPT)



(b) The heuristic *PCIterativeBestSensor* before rounding: M=1,5



(c) The heuristic *PCIterativeBestSensor* after rounding: M=3

Figure 3.3: The algorithm *PCIterativeBestSensor* gives a result $\frac{3}{2} \cdot OPT$ (solution (b)(c)), where M is a value of uncontrolled flow, $S = \{1, 2, 3\}$, $T = \{9, 10\}$, $k = 1$, and D is defined by gray striped circles.

3.5.3 Scenario2 *PC* problem, increasing size of the grid "*gridnet64*", "*gridnet81*", ... , "*gridnet169*"

The experiments were conducted for the grid networks: "*gridnet64*", "*gridnet81*", "*gridnet100*", "*gridnet121*", "*gridnet144*", "*gridnet169*". The number of sensors was fixed $k = 5$.

The diagram Figure 3.5 (c) demonstrates the average time of execution (y axis) as the size of the network increases ($|V|$). As $|V|$ grows, the gap between *PCIterativeBestSensor* and *PC* increases significantly in favor of the heuristic.

3.5.4 Scenario3 *PQ* problem, "*gridnet196*" network, increasing value of quality factor

The experiments were conducted for the grid network "*gridnet196*". The value of quality factor was increasing $q \in \{0.1, 0.2, ..., 1.0\}$.

The diagram Figure 3.5 (d) demonstrates the average number of sensors (y axis) required to control the q-factor of the network traffic (x axis). As the value of q-factor increases, the number of required sensors increases on average, for both *PQ* model and *PQIterativeBestSensor* heuristic. However, at a certain point sensor usage becomes saturated. In the worst observed cases *PQIterativeBestSensor* required approximately one sensor more than *PQ* to achieve the same quality.

The diagram Figure 3.5 (e) demonstrates the average time of execution (y axis). The observed average values of execution time of *PQ* are up to 5 times higher than those of *PQIterativeBestSensor*.

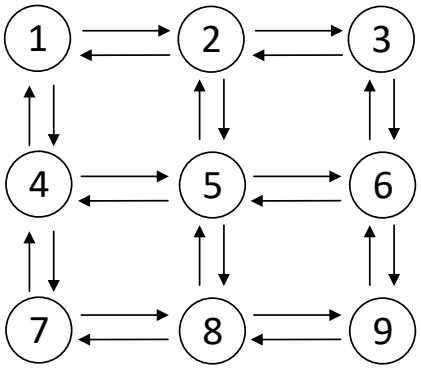


Figure 3.4: An example of a small grid network $|V| = 9$

3.5.5 Scenario4 *PQ* problem, increasing size of the grid "*gridnet121*", "*gridnet144*", ... , "*gridnet256*"

The experiments were conducted for the grid networks: "*gridnet121*", "*gridnet144*", "*gridnet169*", "*gridnet196*", "*gridnet225*", "*gridnet256*". The quality factor was fixed $q = 0.5$.

The diagram Figure 3.5 (f) demonstrates the average time of execution (y axis) as the size of the network increases ($|V|$). As $|V|$ grows, the gap between *PQIterativeBestSensor* and *PQ* increases significantly in favor of the heuristic.

3.5.6 Scenario1b-4b super source formulation

In general, we would like to assume, that network flooding targeted at protected nodes $t \in T$ can start from any network node (*source*) $s \in V \setminus T$. In practical scenarios however, we may want to limit attention to a set of sources $S \subseteq V \setminus T$. For example, after conducting a network risk analysis, we may know that some sources (autonomous systems, sub-networks) are more hostile than others. For experiment purpose, we applied two methods of source selection.

First, explicit selection, as used in the experiments $1 - 4$ (Section 3.5.2, 3.5.3, 3.5.4 and 3.5.5). We selected subsets of vertices as sources $|S| = 40$. The sources were selected randomly with uniform distribution from set $V \setminus T$.

Second, instead of selecting a set of sources S explicitly, we can limit the portion of traffic we want to monitor from each source $s \in V \setminus T$ based on risk analysis $R : V \to [0, 1]$ (see below *single super source formulation* for details). This method was applied within scenarios $1b-4b$. The experiments $1b-4b$ were conducted with the following assumptions: Scenario1b: "*gridnet100*", the number of sensors from $k = 0$ to $k = 10$; Scenario2b: $k = 5$, size of the grid "*gridnet64*", "*gridnet81*", ... , "*gridnet169*"; Scenario3b: "*gridnet289*", the value of quality factor $q \in \{0.1, 0.2, ..., 1.0\}$; Scenario4b: $q = 0.5$, size of the grid "*gridnet144*", "*gridnet169*", ... , "*gridnet256*".

Algorithms efficiency demonstrated in experiments $scenario1b - 4b$ (Figure 3.6) is similar to that demonstrated in experiments $scenario1 - 4$ (Figure 3.5).

Single super source formulation: With a standard trick the problem can be reduced to an equivalent one, with a single source. Having a graph $G = (V, E)$ and a risk analysis as a function $R : V \to [0, 1]$, we create a new graph $G' = (V \cup \{ss\}, E \cup$

$\{(ss, v)\}_{v \in V \setminus T}$), where ss is an artificial super vertex, and capacities of edges in $\{(ss, v)\}_{v \in V \setminus T}$) are given by:

$$\forall_{v \in V \setminus T} \; c(ss, v) = R(v) \cdot \sum_{u:(v,u) \in E} c(v, u). \tag{3.12}$$

For the graph G' we assume a single attack source $S = \{ss\}$. Within G' we simply limit vertex production (possible outgoing flow value) according to its risk value.

In case this formulation is used to characterize the attack sources we need to add additional restriction (Equation 3.13) to both PQ and PC models (models described in Section 3.3). This is required since the super source vertex ss in graph G' is an artificial vertex and in fact a sensor can not be placed in it. The same restriction (3.13) applies to both algorithms *PQIterativeBestSensor* and *PCIterativeBestSensor* (Section 3.4).

$$d[ss] = 0 \tag{3.13}$$

3.5.7 Summary of simulation results

The PC algorithm simulations led to a number of observations. Firstly, for all test networks, as the number of sensors increases, the volume of uncontrolled traffic decreases to zero, for both PC model and *PCIterativeBestSensor* heuristic. Secondly, the observed average objective values of *PCIterativeBestSensor* are higher than those of PC by up to 8% for tested networks. Finally, as the size of the grid network increases, for fixed k, the execution time gap between *PCIterativeBestSensor* and PC increases significantly in favor of the heuristic.

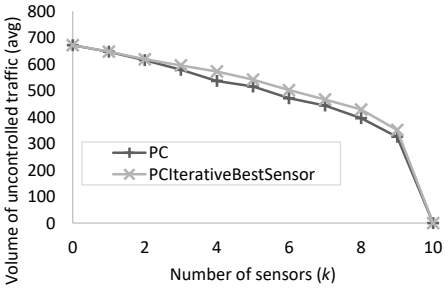
The PQ algorithm simulations led to the following observations. Firstly, as the quality factor increases, the number of sensors increases on average, however, at a certain point sensor usage becomes saturated, for both PQ model and *PQIterativeBestSensor* heuristic. Secondly, in the worst observed cases the *PQIterativeBestSensor* required approximately one sensor more than PQ to achieve the same quality. Finally, as the size of the grid network increases, for fixed q, the execution time gap between *PQIterativeBestSensor* and PQ increases significantly in favor of the heuristic.

3.6 Conclusions

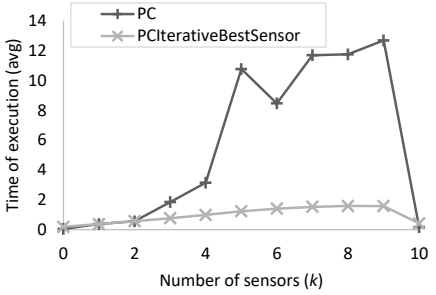
We give a proof that the sensor placement problem is NP-complete. Additionally, we prove that the optimization problem admits no polynomial-time 2-approximation algorithm, unless $P \neq NP$. So, few natural questions arise: is there a better exact algorithm than brute-force? Can the number of sensors be approximated with any constant?

Although the problem is computationally hard it can be efficiently solved with the use of a mixed integer programming solver for medium-sized networks. As demonstrated for the tested grid networks, computation time is not high and qualifies both PC and PQ models for practical applications. The models respond to the challenges of the real DDoS problem. One challenge is that an attack can be conducted from any network node. The other is that sensors are expensive and placing them in all network nodes is not possible in many cases. Sensors can be placed dynamically, based on perceived network indicators (e.g., risk factor). The models expose a highly desirable feature, such that dislocation of relatively small number of sensors (proportional to the number of protected nodes) can obtain a significant quality. Both models lead to a trade-off between the number of deployed sensors and the volume of uncontrolled flow.

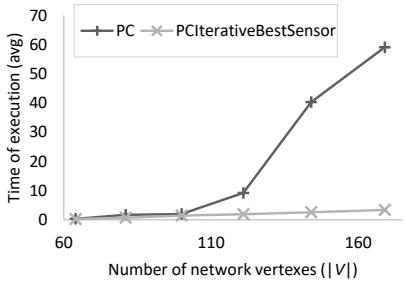
Additionally to two models, we designed two efficient solver-based heuristics (one for each problem). For large networks, the execution time gap between the two models and their corresponding heuristics increases significantly in favor of the heuristics.



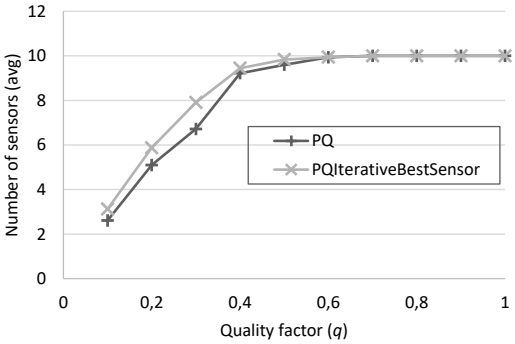
(a) Scenario1: Average volume of uncontrolled traffic



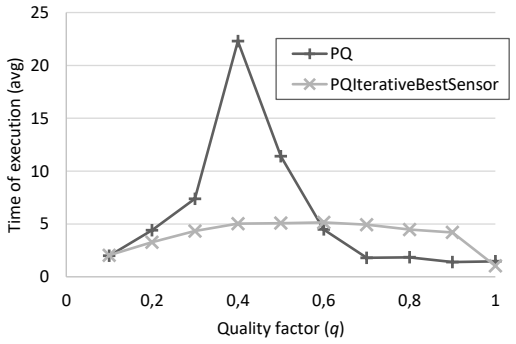
(b) Scenario1: Average time of execution (sec.)



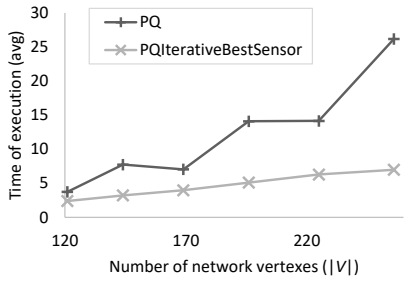
(c) Scenario2: Average time of execution (sec.)



(d) Scenario3: Average number of sensors

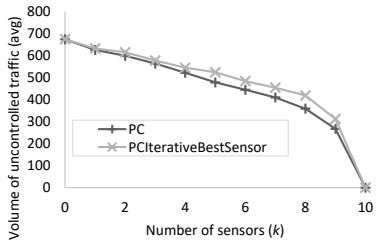


(e) Scenario3: Average time of execution (sec.)

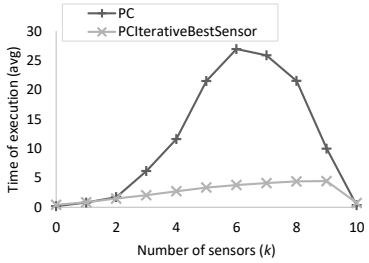


(f) Scenario4: Average time of execution (sec.)

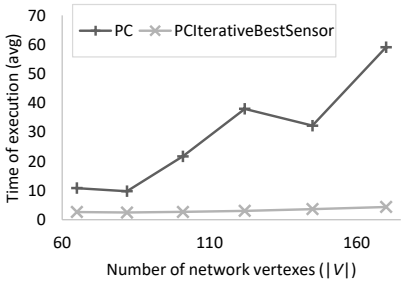
Figure 3.5: Scenario1-4



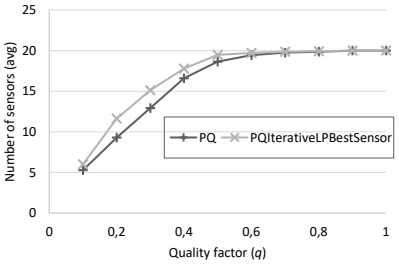
(a) Scenario1b: Average volume of uncontrolled flow



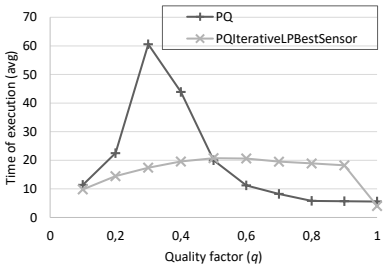
(b) Scenario1b: Average time of execution (sec.)



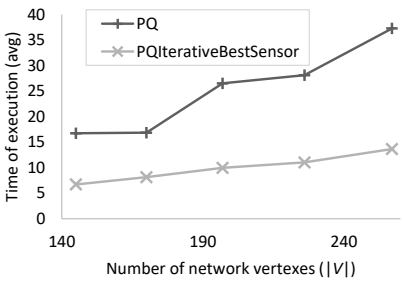
(c) Scenario2b: Average time of execution (sec.)



(d) Scenario3b: Average number of sensors



(e) Scenario3b: Average time of execution (sec.)



(f) Scenario4b: Average time of execution (sec.)

Figure 3.6: Scenario1b-4b

Chapter 4

Multi-Domain Service Function Chain Placement

4.1 Introduction

4.1.1 SFC placement in multi-domain

In recent years, due to the development of network systems based on Software Defined Networking (SDN) and Network Function Virtualization (NFV), computer networks do not rely only on transmission resources, but also on computational resources. In a multi-domain network, NFV may be deployed and hosted by different domains, and may belong to the same (or other) authority.

In this work, we deal with SFC embedding in a multi-domain (multi-administrative) environment and compose E2E multi-domain SFC services based on available national network and compute resources (slices, VNFs). In particular, we consider the case of federated military coalition networks.

In multi-domain context, the literature presents various studies to address slice embedding, particularly within the 5G/6G (Addad et al. [1]). The centralized orchestration requires a global view of the infrastructure, which raises scalability concerns. The desire to reduce communication costs has led to developing distributed slice embedding solutions. Other related works on multi-domain SFC address, e.g., reduction of deployment time (El Amine et al. [36]), dynamic orchestration (Wu and Zhou [129]), and SFC placement with limited visibility (Toumi et al. [119]).

In coalition networks, by taking advantage of new technologies SDN/NFV, various network operators (nations, in the context of military networks) can provide efficient and attack-resistant network services by sharing and complementing each other's network cyber-defense capabilities (Li and Wang [75]) provided in the form of network functions. Our idea goes in the direction of building cooperative defenses for network protection.

4.1.2 Our proposal

Our goal is to assure attack-resistant end-to-end (E2E) services. To achieve this, in addition to network resources, we require additional security functions (also referred to as critical functions in this work) that carry out threat detection and mitigation to prepare the network against cyber-attacks. Such critical protection and mitigation security functions could be, e.g., Firewall (FW), Intrusion Detection System (IDS), Distributed Denial of Service (DDoS) attack sensor (Junosza-Szaniawski, Nogalski and Rzazewski [63], Blazek et al. [16]), traffic scrubber (Fayaz et al. [39]), DDoS mitigation (Belabed et al. [13]), encryption/decryption, traffic filtering, etc. In addition, there are other crucial functions of network protection (of Quality of Service (QoS) nature), e.g., compression, decompression, traffic shaping, traffic optimizer, and load balancer (Beck and Botero [12]).

We define resilient-to-attack network service as a Service Function Chain (SFC) composed of a set of Virtual Network Functions (VNFs) and the directional virtual links that connect them (the flow of packets passes through the chain of VNFs that constitute the SFC). In a multi-domain network, NFV can be deployed and hosted by different domains.

We address the embedding of service chains (*SFC embedding* in short) in a multi-domain context. When embedding a multi-domain SFC in a multi-administrative domain federation, the initiating domain relies on the compact views of each domain's network topology (the abstract topologies) to form an E2E SFC. Typically, domains limit the disclosure of topology information (Toumi et al. [119]). Classically, the abstract topology shared with other domains consists of border nodes and abstract links connecting them. Some domains may also share compute nodes with a set of VNFs they can host. Our work also proposes sharing other elements, e.g., domain-level slices and non-border transit nodes (Figure 4.1).

The SFC embedding algorithm executed on these abstracted topologies is referred to as *federated-level SFC embedding*. It assigns network and computing resources exposed

by domains to the requested multi-domain SFC. All of these resources should be checked and confirmed, as some aspects of the abstractions may be out of date, even though they were valid at the time of announcement. Domain-level sub-SFC embeddings are then triggered on the selected domains. Without loss of generality, in this work we focus on federated-level embedding and implicitly assume that all advertised abstractions remain valid at the time of provisioning of the requested multi-domain SFC.

Although the SFC embedding problem is largely investigated in the literature, only few works address it in a sliced collaborative multi-administrative network federation. An example of SFC embedding at the federated level within a two-domain network is presented in the Figure 4.1. Domain 1 exposes two data centers (each exposes two VNFs: FW, IDS). Domain 2 exposes one data center (VNFs: FW, IDS). Both domains expose edge nodes (green) and non-edge transit nodes (black). Some links represent domain-level slices (dotted). We consider the embedding of a single user demand in the form of E2E SFC. The user demand (blue) requires a chain of two VNFs (FW, IDS). For simplicity, bandwidth and latency characteristics are omitted here.

Our work provides important insights for adopting slicing in a federated network for the following reasons. First, a single nation may not have all the necessary network capabilities to define an E2E SFC (e.g., degradation due to a DDoS attack (Chapter 3) or a control plane attack (Chapter 2). Second, it may support several interesting security scenarios e.g.: (1) Following Fayaz et al. [39], traffic scrubbing by remote VNFs. As soon as we open military networks toward the Internet of Things the risk of volumetric attacks will increase (Chapter 3). (2) Following Li and Wang [75], MEC (Multi-Access Edge Computing) (5G) resilience to attacks from end-user networks can be increased by using shared IDS (Intrusion Detection System) resources.

To sum up, our contribution consists of the following key elements:

- First, we deal with SFC embedding in a multi-domain context. The structure of a federated topology is described in more detail in our previous work (Pedebearn et al. [93]). Users are allowed to compose E2E SFC services spanning different domains.

- Second, we address the multi-domain SFC embedding by proposing an Integer Linear Programming (ILP) model that solves VNF and link embedding simultaneously (in existing works, they are usually solved separately). Our ILP model differs from

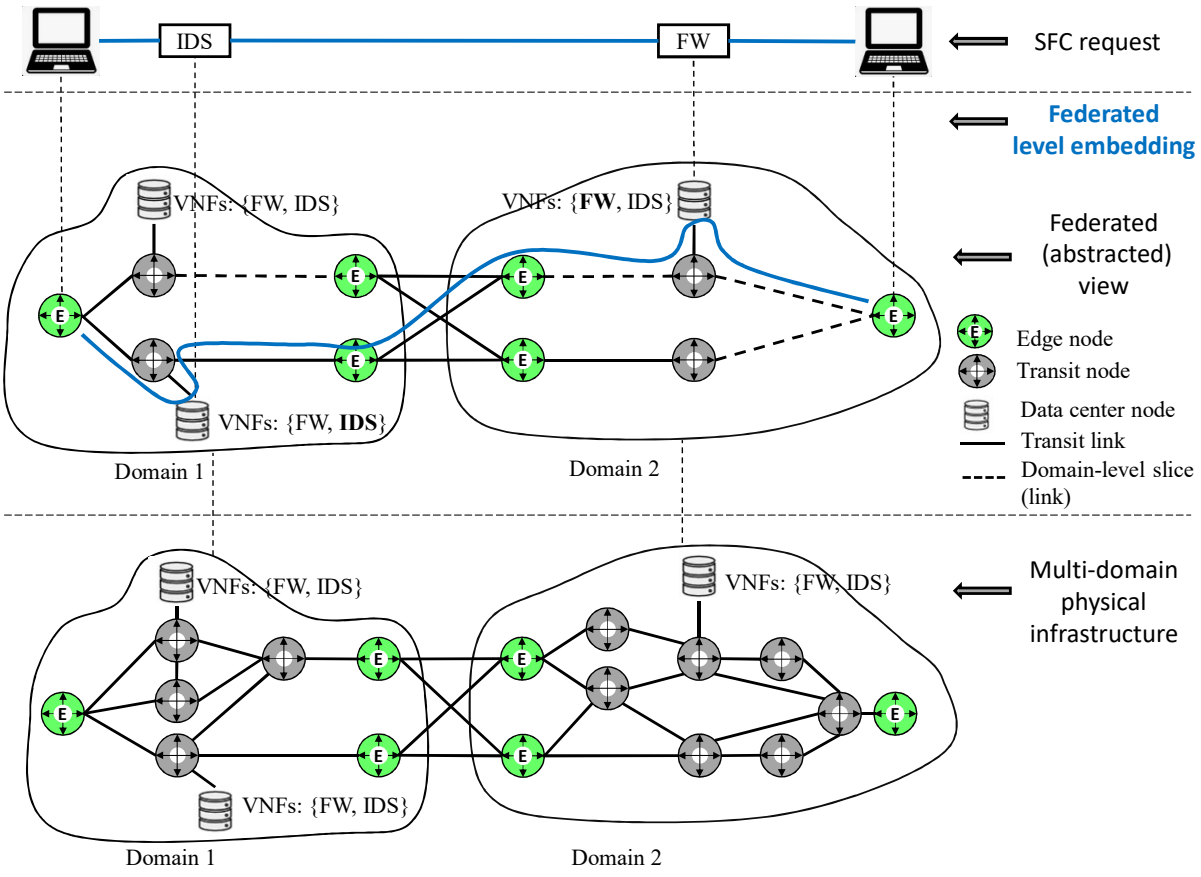


Figure 4.1: SFC embedding in sliced multi-administrative multi-domain network (federated-level view)

the models present in the literature (Addis et al. [2], Wion et al. [127]). We use a different goal function (a combination of slice deployment cost and link utility minimization). We use a (de)compression VNF and SFC ordering. Each topology link can represent domain-level slice. In addition, since the problem is NP-hard (Addad et al. [1]), we provide an efficient heuristic.

- Third, we performed extensive experiments on known topologies. We demonstrate that the ILP model efficiently solves SFC problems of a practical size, while the heuristic can solve large-scale problems very fast with decent efficiency. This leads to a trade-off between the two.

4.2 Problem description

4.2.1 Notation

The notation describes parameters (Table 4.1) and variables (Table 4.2) that define the network and demand model. This is a general notation, common for both the ILP model (Section 4.3.1) and greedy heuristic (Section 4.4.1).

We model the federated-level abstract network (abstracted view) for the purpose of federated-level SFC embedding as a directed graph $G = (N, A)$, where N is a set of nodes, and A is a set of directed links. A node can be: an edge node or a non-edge transit node. A data center (hosting VNFs) can be co-located with either of them. A link $a \in A$ can be a slice link $a \in A_s$ (it represents a domain-level slice) or a classical transit link. An example federated level network graph is shown in the middle part of Figure 4.1.

The central part of the demand model is the definition of the service chains. The tuple $(f^{k,1}, .., f^{k,T_k})$ (list of ordered services) is defined for each k-th demand $k \in D$. Each E2E demand is represented by a single SFC. The user may define more than one demand, e.g., a separate one for data and control plane purposes.

Additionally to Table 4.1, since for each $f \in F$ we might have an individual compression factor μ_f, we define b_k^p as follows:

$$b_k^p = \begin{cases} b_k, & p = 0 \\ b_k \cdot \prod_{q=1}^{p} \mu_{f^{k,q}}, & p \in \{1, \ldots, T_k\} \end{cases} \tag{4.1}$$

The b_k^p denotes bandwidth used by demand $k \in D$ after the first p services $f^{k,1}, .., f^{k,p}$ are performed.

4.3 Model description

4.3.1 The ILP Model

The model takes two types of federated-level parameters as input. First, the network model: topology, bandwidth, latency, VNF types, and locations (node). Second, the demand model: bitrate, delay, and SFC requirement (Table 4.1-4.2). A detailed description

Table 4.1: Notation I

Parameters

N	Set of nodes
A	Set of links (arcs)
$A_s \subseteq A$	Subset of links (arcs) which are slice-able
$\gamma_{i,j}^{max}$	Maximum capacity of the arc (i,j)
$\gamma_{i,j}$	Available capacity of the arc $(i,j) \in A$. 'Available' since we consider iterative allocations (for heuristic purpose)
$l_{i,j}$	Latency of the link $(i,j) \in A$, which is an upper bound that includes the transmission delay and the propagation delay
F	Set of service types (VNF types)
D	Set of demands
o_k, t_k	$o_k, t_k \in N$ origin and target of k-th demand $k \in D$
b_k, L_k	Bandwidth and max latency of k-th demand $k \in D$
$f^{k,p}$	$f^{k,p} \in F$ is p-th service step of k-th demand $k \in D$
T_k	For k-th demand it is the length of its service chain $(f^{k,1}, .., f^{k,T_k})$
μ_f	Compression/decompression factor for service $f \in F$, where $\mu_f = 1$ no compression/decompression; $\mu_f \leq 1$ compression VNF; $\mu_f \geq 1$ decompression VNF
N_f	$N_f \subset N$ set of nodes which provide a service (VNF) of type $f \in F$
N_F	$N_F = \bigcup_{f \in F} N_f$ set of nodes which provide a service (VNF) of any type
b_k^p	Bandwidth used by demand $k \in D$ after the first p services $f^{k,1}, .., f^{k,p}$ are performed

of federated-level abstraction can be found in our previous work Pedebearn et al. [93]. The optimization problem is to find:

- the optimal domain cost of the slice deployment (by minimizing the number of deployed national slices)
- the optimal link utilization (by minimizing maximum link utilization - to increase future demand admissibility)

subject to the following constraints:

Table 4.2: Notation II
Decision Variables

$\phi_{i,j}^{k,p}$	Continuous variable, represents the flow on the arc $(i,j) \in A$ of demand $k \in D$ for the p-th service step. The p-th service step is the state of demand processing after p-th service ($f^{k,p}$) is performed and before $p+1$-th service ($f^{k,p+1}$) is performed. For the initial service step $p = 0$, the flow $\phi_{i,j}^{k,0}$ denotes state before the first service ($f^{k,1}$) is performed
$x_{i,j}^{k,p}$	Binary ($x_{i,j}^{k,p} \in \{0,1\}$), $x_{i,j}^{k,p} = 1$ iff $\phi_{i,j}^{k,p} > 0$ (the arc $(i,j) \in A$ is used by the flow $\phi_{i,j}^{k,p}$)
$e_{i,j}$	Binary ($e_{i,j} \in \{0,1\}$), $e_{i,j} = 1$ iff the arc $(i,j) \in A$ is used by at least one flow $\phi_{i,j}^{k,p}$ for $k \in D, p \in \{0,\dots,T_k\}$ ($e_{i,j} = \max\{x_{i,j}^{k,p} : k \in D, p \in \{0,\dots,T_k\}\}$)
$z_i^{k,p}$	Binary ($z_i^{k,p} \in \{0,1\}$), $z_i^{k,p} = 1$ iff for demand $k \in D$ the p-th service ($f^{k,p} \in F$) is executed at node $i \in N_{f^{k,p}}$
U	Continuous variable, represents maximum link utilization rate

- Flow conservation laws are defined via the relationship between ϕ and z variables. We distinguish the following main cases: $p = 0$, $p = T_k$, and otherwise.

  For $p = 0$ (unprocessed flow of demand $k \in D$), we have two sub-cases. First, if $i = o_k$ (4.2a), then the flow balance (outgoing flow value - incoming flow value) equals b_k^0, which is b_k. Second, if $i = N - \{o_k\}$ (4.2b), we have two states: either $z_i^{k,1} = 0$ ($f^{k,1}$ service is not placed at i) then the flow balance is 0 or $z_i^{k,1} = 1$ then flow $\phi^{k,0}$ terminates in the node i and thus the flow balance is $-b_k^0$.

  For $p = T_k$ (flow of demand $k \in D$ processed by service f^{k,T_k}) we have two sub-cases. First, if $i = t_k$ (4.2e), then the flow balance equals $-b_k^{T_k}$. Second, if $i = N - \{t_k\}$ (4.2d), we have two states: either $z_i^{k,T_k} = 0$ (f^{k,T_k} service is not placed at i) then the flow balance is 0 or $z_i^{k,T_k} = 1$ then the flow ϕ^{k,T_k} starts at the node i and thus the flow balance is $b_k^{T_k}$.

  Otherwise, for $p > 0$ and $p < T_k$ (4.2c) we have three states. If $z_i^{k,p} = 0$ and $z_i^{k,p+1} = 0$, then the flow balance is 0. If $z_i^{k,p} = 1$ and $z_i^{k,p+1} = 0$, then the flow $\phi^{k,p}$ starts at the node i and thus the flow balance is b_k^p. If $z_i^{k,p} = 0$ and $z_i^{k,p+1} = 1$, then the flow $\phi^{k,p}$ terminates at the node i and thus the flow balance is $-b_k^p$.

$$\forall k \in D\ \forall i \in N\ \forall p \in \{0,..,T_k\}$$

$$\sum_{(i,j)\in A} \phi_{i,j}^{k,p} - \sum_{(j,i)\in A} \phi_{j,i}^{k,p}$$

$$= \begin{cases} b_k^0 & p = 0, i = o_k & (4.2a) \\ -b_k^0 \cdot z_i^{k,1} & p = 0, i \in N - \{o_k\} & (4.2b) \\ b_k^p \cdot z_i^{k,p} - b_k^p \cdot z_i^{k,p+1} & p \in \{1,..,T_k - 1\}, i \in N & (4.2c) \\ b_k^{T_k} \cdot z_i^{k,T_k} & p = T_k, i \in N - \{t_k\} & (4.2d) \\ -b_k^{T_k} & p = T_k, i = t_k & (4.2e) \end{cases}$$

- For non-VNF nodes $i \in N \setminus N_F$ variable z equals zero

$$\forall k \in D\ \forall p \in \{1,..,T_k\} \quad z_i^{k,p} = 0 \tag{4.3}$$

- For demand $k \in D$, the p-th service ($f^{k,p}$) is performed by at most one service available at some node $i \in N_{f^{k,p}}$

$$\forall k \in D\ \forall p \in \{1,..,T_k\} \sum_{i \in N_{f^{k,p}}} z_i^{k,p} = 1 \tag{4.4}$$

- There is flow only on used edges - the connection between ϕ and x

$$\forall k \in D\ \forall (i,j) \in A\ \forall p \in \{0,..,T_k\}\ \phi_{i,j}^{k,p} = b_k^p \cdot x_{i,j}^{k,p} \tag{4.5}$$

- Maximum latency has two components: delay on transport link A, and delay caused by VNF processing (the latter can be easily included)

$$\forall k \in D \sum_{(i,j)\in A} \sum_{p\in\{0,..,T_k\}} l_{i,j} \cdot x_{i,j}^{k,p} \leq L_k \tag{4.6}$$

- If the slice arc $(i,j) \in A_s$ is used by at least one demand $k \in D$, then the slice is enabled

$$\forall k \in D\ \forall p \in \{0,..,T_k\}\ \forall (i,j) \in A_s \quad x_{i,j}^{k,p} \leq e_{i,j} \tag{4.7}$$

- If the slice arc $(i,j) \in A_s$ is not used by any demand $k \in D$, then the slice is disabled

$$\forall (i,j) \in A_s \sum_{k \in D} \sum_{p \in \{0,..,T_k\}} x_{i,j}^{k,p} \geq e_{i,j} \tag{4.8}$$

- The sum of flows does not exceed the edge capacity. For online version of the algorithm (also for Section 4.4.1)

$$\forall (i,j) \in A,$$
$$1 - \frac{1}{\gamma_{i,j}^{max}} \cdot \left(\gamma_{i,j} - \sum_{k \in D} \sum_{p \in \{0,..,T_k\}} \phi_{i,j}^{k,p} \right) \leq U$$

We consider two objective functions:

- Traffic Engineering (TE) goal: minimize the maximum network link utilization (e.g., to increase future demand admissibility):

$$\min U \tag{4.9}$$

- Slice Deployment (SD) goal: minimize the number of used slices (e.g., to reduce slice setup time):

$$\min \sum_{(i,j) \in A_s} e_{i,j} \cdot \frac{1}{|A_S|} \tag{4.10}$$

We define S as equal to $\sum_{(i,j) \in A_s} e_{i,j} \cdot \frac{1}{|A_S|}$.

Thus, depending on the federated operator need, the objective is to balance between TE and SD goals by adjusting α, where $\alpha \in [0,1]$ is a parameter describing the importance of TE goal over SD goal

$$\min \alpha \; U + (1-\alpha) \; S \tag{4.11}$$

We define G as equal to $\alpha \; U + (1-\alpha) \; S$.

4.4 Algorithm description

4.4.1 The Greedy Heuristic

The heuristic is given as Algorithm 4. Firstly, the demand set is sorted by latency (could also be sorted by bandwidth if needed) (line 3). Secondly, in each iteration (line 5-10) the

subset of d demands $D_d \in D$ is selected (line 5), processed by model (line 6), embedded (line 7-8), and demands marked as processed (line 9).

Algorithm 4 SFC Greedy Heuristic

1: INPUT: $G(N, A)$; A_s; for each link $(i, j) \in A$ capacity $\gamma_{i,j}^{max}$, $\gamma_{i,j}$, latency $l_{i,j}$; VNF advertisements F, N_f; demands $k \in D$; d - number of demands processed in a single algo iteration
2: OUTPUT: result ϕ (list of embedded slices (path definitions)), in other words $|D|$ E2E slices (paths) one for each requested demand $k \in D$, defined via $\phi_{i,j}^{k,p}$ for each $k \in D$, $(i, j) \in A$ and $p \in \{0, .., T_k\}$
3: Sort demands D by latency (from low latency to high latency (latency insensitive))
4: **while** $D \neq \emptyset$ **do**
5: Select d demands $D_d \subseteq D$ (if $d > |D|$, select remaining)
6: Run SFC ILP Model (Section 4.3.1) with the input $(N, A, A_s, \gamma_{i,j}^{max}, \gamma_{i,j}, l_{i,j}, F, N_f, D_d)$ to get $\phi_{i,j}^{k,p}$ for each $k \in D_d$, $(i, j) \in A$ and $p \in \{0, .., T_k\}$
7: Add $\phi_{i,j}^{k,p}$ assignments to the result ϕ (list of embedded paths)
8: Update available capacity $\forall (i, j) \in A,\ \gamma_{i,j} = \gamma_{i,j} - \sum_{k \in D_d} \sum_{p \in \{0,..,T_k\}} \phi_{i,j}^{k,p}$
9: $D = D \setminus D_d$
10: return ϕ
11: **end while**

4.5 Computational results

The model (Section 4.3.1) was implemented in Optimization Programming Language (IBM CPLEX v22.1), and the heuristic (Section 4.4.1) was implemented in Python v3.10.

4.5.1 Evaluation1 simulation on "*cost266*" network

For this experiment, we defined a multi-domain topology based on the "*cost266*" topology (Orlowski et al. [90]), by assigning nodes (cities) to domains (countries) (Figure 4.2).

Results of ILP model

We conducted the tests of the model with the following configurations (4x2):

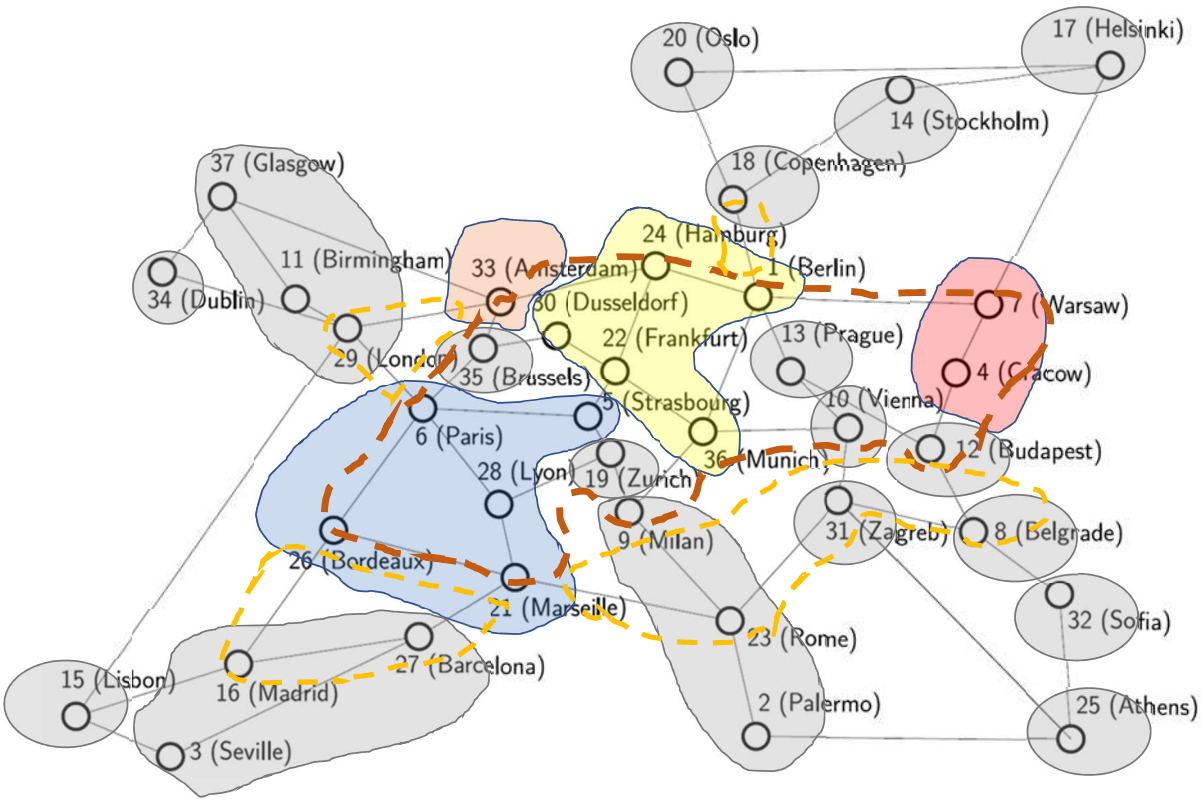


Figure 4.2: Evaluation1 Multi-domain topology based on "*cost266*"

- 4 topology configurations. Each topology configuration was generated with a randomly assigned maximum link capacity and latency. Links in the red dotted area (Figure 4.2) received a capacity in the range of 5-10 Gbit/s. Links in the orange dotted area 2-5 Gbit/s, and peripheral links 1-2 Gbit/s. All links received latency in the range of 3-10 ms.

- 2 data center (DC) configurations. Each DC configuration contained seven randomly selected DCs - two located in France, three in Germany, one in Poland, and one in the Netherlands. For each DC, we randomly assigned two VNFs out of four from the VNF catalog = {FW, IDS, Deep Packet Inspection (DPI), Traffic Storage}).

We conducted three series of measurements, each corresponding to a fixed number of user demands {4, 6, 8}. Demand sets were generated incrementally. To the demand set of four demands (4), we added two demands (6) and again two demands (8). A single user demand was defined in the following way:

- a random start and end (in a different domain)

- a random bitrate in the range of 100-400 Mbit/s
- a random SFC (two ordered VNFs selected from the VNF catalog)
- a random maximum latency. In total, 25% of all demands in a demand set (D) were low latency (100 ms), and others latency insensitive.

For each series, we assumed the following values $\alpha = \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ (Equation 4.11). For fixed α, we conducted eight measurements (4x2) by changing topology and DC configuration and computed the average value of the goal (G), link utilization (U), slice deployment (S), and execution time (T). These average values were taken to analyze the experiment results (Figure 4.3).

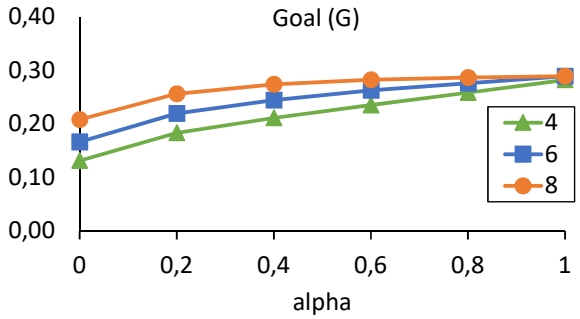
As one can observe, when the number of demands goes up, the combined goal value (G) (Equation 4.11) goes up (or stays at the same level) for any value of α (Figure 4.3a). This shows the expected behavior of the model since, as we embed more demands, either more links are used (e.g. $\alpha = \{0, .., 0.8\}$ (Figure 4.3c)), or the maximum link utility might be increased (but will never decrease) (e.g. $\alpha = \{0.2\}$ (Figure 4.3b)).

Additionally, the average U and S (Equation 4.9 and 4.10) values behave as expected for any demand series (Table 4.3, Figure 4.3b-c). For any number of demands: first, when α increases, the U decreases (as we increase the importance of the link utility minimization factor) (Figure 4.3b); second, when α decreases, the S decreases (as we increase the importance of the slice deployment cost minimization factor) (Figure 4.3c).

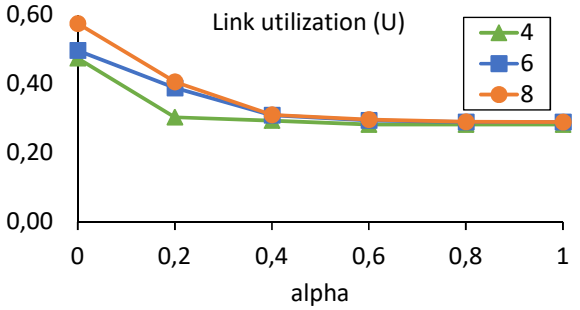
Since the problem is NP-Hard, we show the model significant computation times (Figure 4.3d, up to 1730 s for only 8 demands). This is the reason why we designed the heuristic.

Table 4.3: Evaluation1 “cost266”: The results of the model for demand set $\{4, 6, 8\}$ - U and S

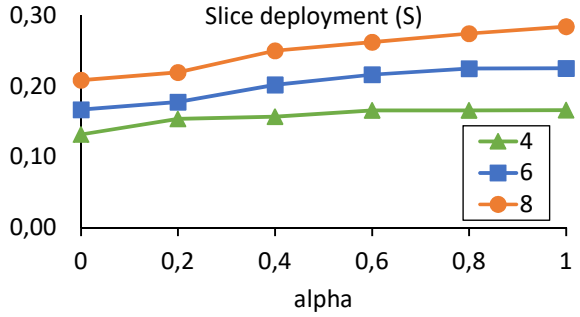
$\lvert D\rvert \setminus \alpha$	0	0,2	0,4	0,6	0,8	1	U or S
4	-	0,303	0,293	0,282	0,282	0,282	U
6	-	0,389	0,309	0,294	0,290	0,290	U
8	-	0,406	0,310	0,297	0,290	0,290	U
4	0,132	0,154	0,157	0,166	0,166	-	S
6	0,167	0,178	0,202	0,216	0,225	-	S
8	0,208	0,219	0,250	0,262	0,274	-	S



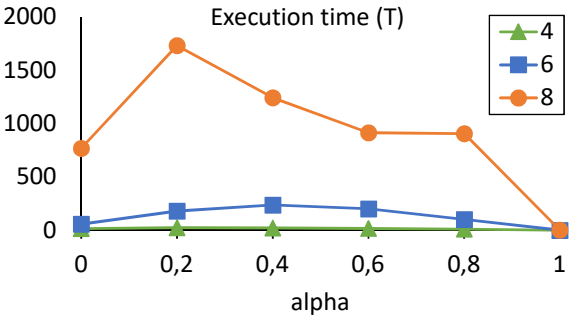
(a) The average goal value



(b) The average link utilization



(c) The average slice deployment



(d) The average execution time (s)

Figure 4.3: Evaluation1 “cost266”: The results of the model for demand set $\{4, 6, 8\}$ - G, U, S and T

Results of heuristic

We conducted tests of the heuristic with the same 4 topology configurations and 2 DC configurations as in the previous subsection (same 4x2 configurations as for the testing of the ILP model). This time we conducted four series of measurements, each corresponding to a fixed number of user demands {10, 25, 45, 60}. Demand sets were generated incrementally. The heuristic was run with step $d = 3$.

As one can observe (Figure 4.4a), the average G values behave as expected; for any α, they increase as we increase the number of demands. The execution time is much lower compared to the one measured for the ILP model. A single observed average max computation time is up to 35 s (for 60 demands, Figure 4.4d).

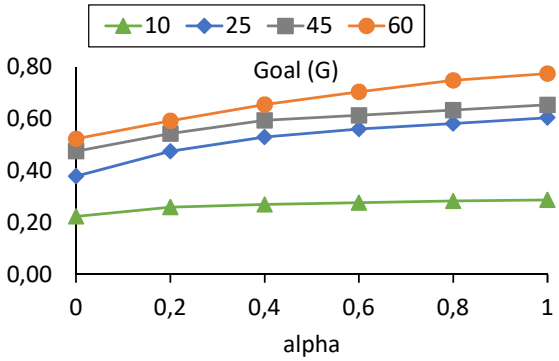
4.5.2 Evaluation2 simulation on "*nsfnet*"network

We proved experimentally that the ILP model (Section 4.3.1) is more efficient than the heuristic (Section 4.4.1). First, we transformed the "*nsfnet*" topology (Zhu et al. [136]) to multi-domain by assigning nodes to domains (Figure 4.5). Both the ILP model and the heuristic were run with the following configurations (5x2):

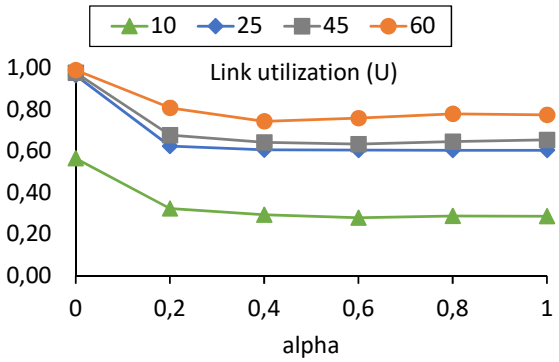
- 5 topology configurations. Each topology configuration was generated with a randomly assigned maximum link capacity and latency. Each link (Figure 4.5) received a capacity from the range of 5-10 Gbit/s. All links received latency in the range of 3-10 ms.
- 2 data center (DC) configurations. Each DC configuration contained seven randomly selected DCs - two located in the East, one in the Center, one in the South, and three in the West. For each DC, we randomly assigned two VNFs out of four from the VNF catalog.

We generated eight random demands. A single user demand was defined in the following way:

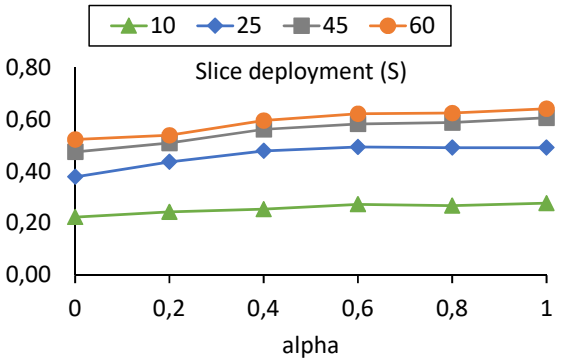
- a random start and end (in a different domain)
- a random bitrate in the range of 100-400 Mbit/s
- a random SFC (two ordered VNFs selected from the VNF catalog)



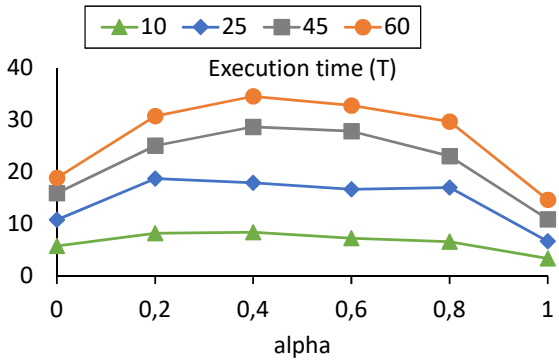
(a) The average goal value



(b) The average link utilization



(c) The average slice deployment



(d) The average execution time (s)

Figure 4.4: Evaluation1 "cost266": The results of the heuristic for demand set $\{10, 25, 45, 60\}$ - G, U, S and T

- a random maximum latency. In total, 25% of all demands in a demand set (D) were low latency (100 ms), and others were latency insensitive.

The heuristic was run with step $d = 2$. As observed in this scenario, the ILP model (M) is more efficient (for any α, it uses less network resources to embed the demand set) than the heuristic (H) by up to 27% (Figure 4.6a). On the other hand, the heuristic is much faster (Figure 4.6d). A single average max computation is approximately 1 s. The model works fast for α=0 and 1. For these cases, the model only minimizes S or U. In such cases, it is easier to find an optimal solution. When $\alpha * U$ and $(1 - \alpha) * S$ have similar values, there are more feasible solutions with similar values to be analyzed by the model.

Additionally, we note that the G function is decreasing (Figure 4.6a), which is different compared to Figure 4.3a and Figure 4.4a, which are both increasing. The behavior of G depends on: the proportion of the sum of all demanded bitrates per average link capacity; the proportion of the number of demands per cardinality of the set A_s; and the structure of the network. The average link capacities in the two experiments were different. The networks also have different size and structure. Furthermore, the monotonicity of the goal function G, in general, is not guaranteed.

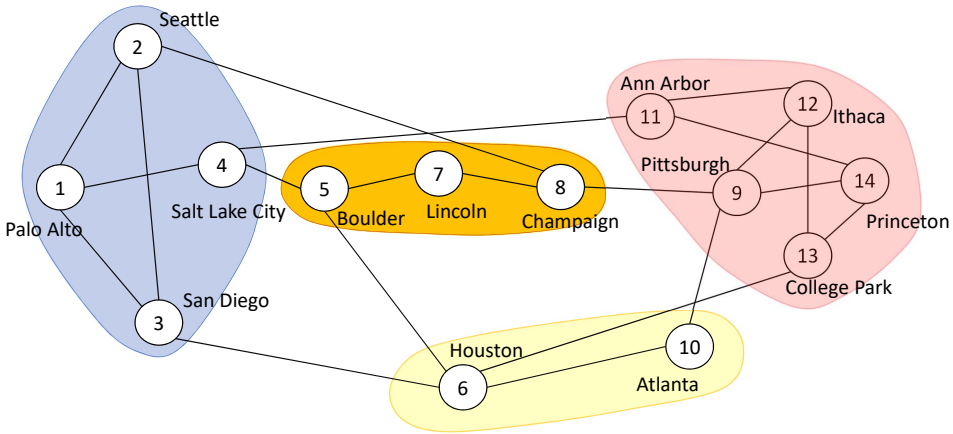
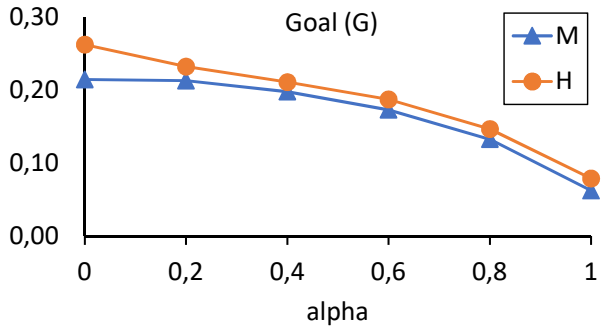


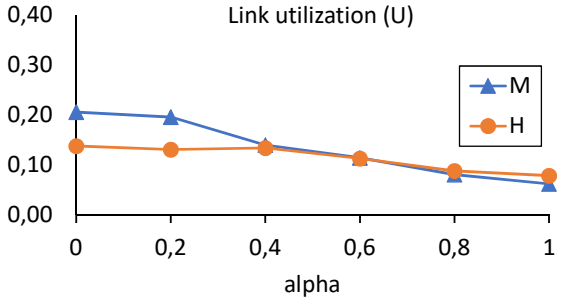
Figure 4.5: Evaluation2 Multi-domain topology based on "*nsfnet*"

4.5.3 Multi-domain SFC emulation - proof of concept

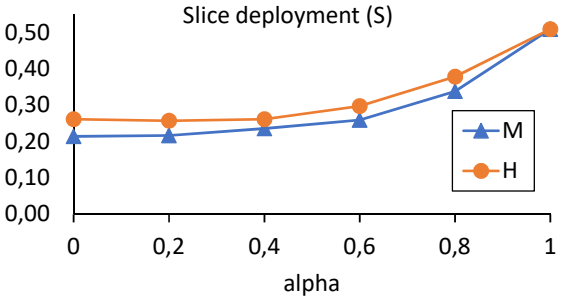
The SFC proof of concept is based on Containernet, Docker (VNFs), Open vSwitch (OVS), and Linux Router. We set up a network of three domains (Domain 1-3, Figure 4.7), with several OpenFlow-enabled OVS switches (performing packet forwarding within a domain)



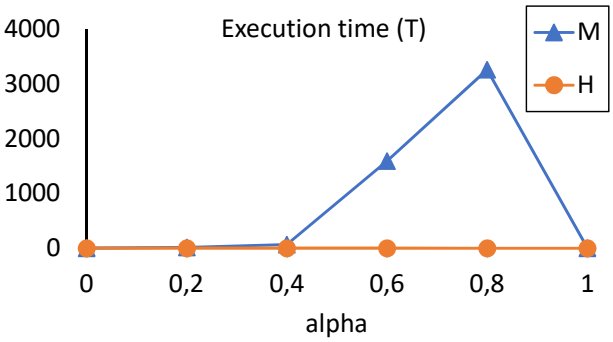
(a) The average goal value



(b) The average link utilization



(c) The average slice deployment



(d) The average execution time (s)

Figure 4.6: Evaluation2 “nsfnet”: comparison of the model (M) and heuristic (H) for demand set {8}

and Linux IP Routers (that connect different domains). Two user networks (UN) are connected to each domain. To emulate the security scenario (IDS resource share), we use Snort IDS (VNF). Snort VNFs are located in Data Centers 1, 2, and 3 (DC for short). DC4 contains a Storage VNF. For simplicity, each link capacity (inter-domain and intra-domain) has the same capacity, equal to 2 units.

We assume that slice-able links are switch-to-switch links (s_i, s_j). In our topology, there are 15 slice-able links (6 in Domain 1, 5 in Domain 2, 1 in Domain 3, and 3 inter-domain slice links).

If, according to the slice definition, the network traffic is to be handled by a VNF service (e.g. IDS - Snort), e.g. transfer of packets from switch S6 to the data center DC1 (Figure 4.8a, orange slice), this is done by copying the traffic to the appropriate network interface of a DC. Additionally, the network interface in the DC (e.g. Snort machine) must be configured in promiscuous mode to enable the capture and reading of every network packet that reaches its interface.

For emulation of our two scenarios, we define a set of two E2E demands (E2E slices), each requires a single VNF in the chain:

- D1 (orange)={origin: UN1, target: UN3, bitrate=1 unit, VNF=(IDS)}
- D2 (green)={origin: UN2, target: UN4, bitrate=1 unit, VNF=(IDS)}

For both scenarios, we compute E2E slices using the ILP model and provision them in our Containernet environment: first by installing OpenFlow rules on OVS switches (intra-domain), and then by installing IP routing entries (inter-domain).

Emulation scenario1 - maximum link utility minimized

In this scenario (Figure 4.8a), we minimize maximum link utility. In our goal function $\min \alpha\ U + (1 - \alpha)\ S$, by setting $\alpha = 1$ we receive $\min\ U$. The computation resulted in $U = 0.5$ (any link is used in a maximum of 50%). As one can observe (Figure 4.8a), slices tend to be disjoint (spread). As explained above, the computed data paths (slices) are provisioned by generating and installing the corresponding OpenFlow rules in all traversed domains.

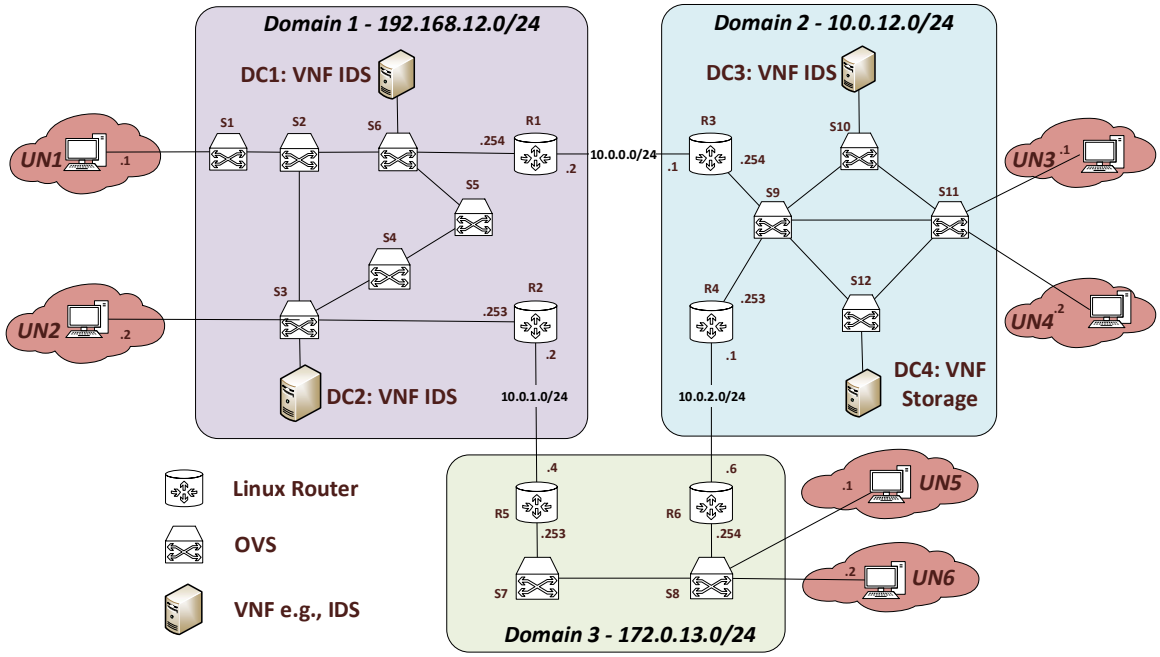


Figure 4.7: Containernet emulation - environment

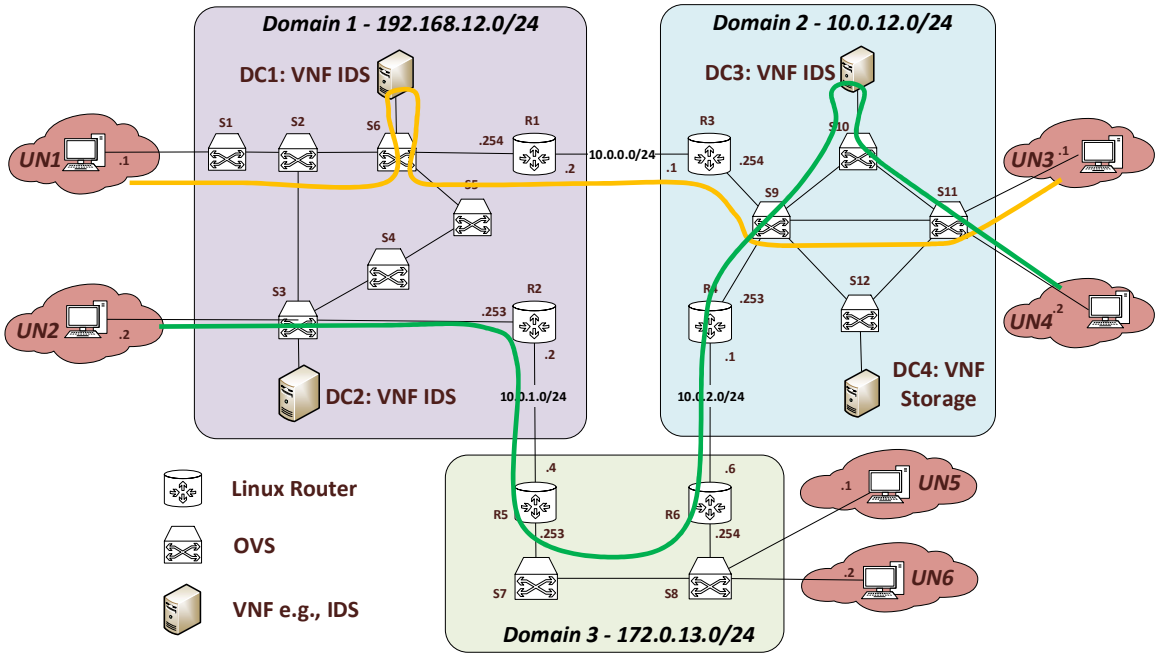
Emulation scenario2 - national slice usage minimized

We minimize the number of used national slices in this scenario (Figure 4.8b). In our goal function $\min \alpha\ U + (1 - \alpha)\ S$, by setting $\alpha = 0$ we receive $\min\ S$. The computation resulted in $S = 0.33(3)$ (used 5 slices out of 15; however, link utility is sacrificed $U = 1$). As one can observe (Figure 4.8b), slices tend to overlap (orange, green). As explained above, the computed data paths (slices) are provisioned by generating and installing the corresponding OpenFlow rules in all traversed domains.

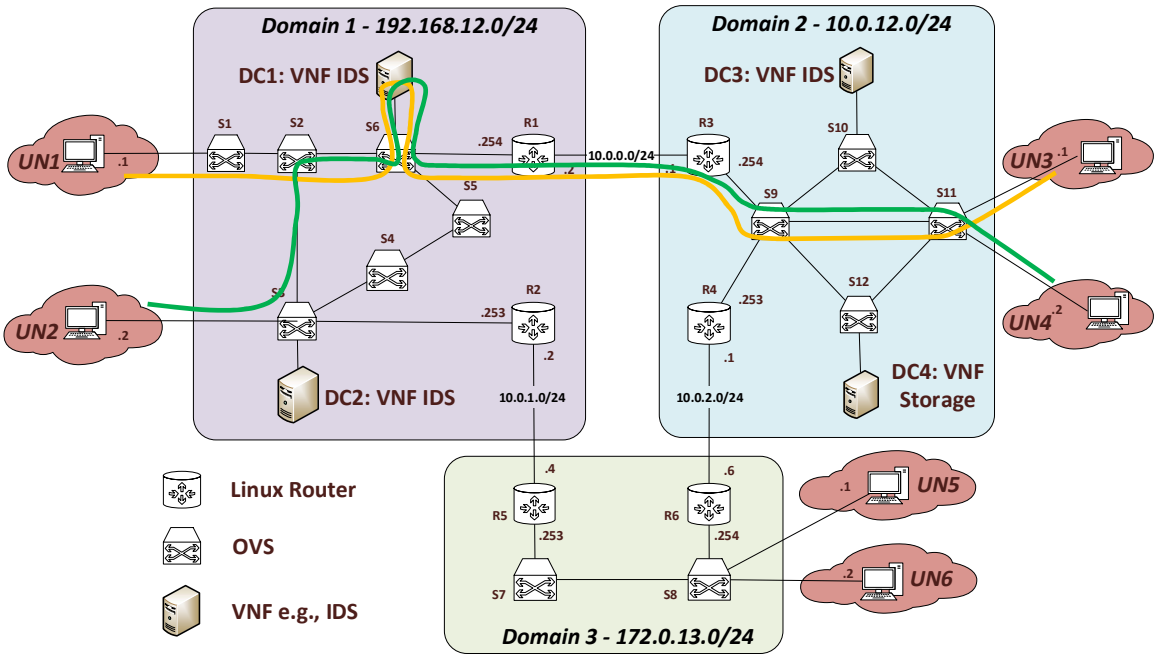
4.6 Speedup methods

4.6.1 Introduction

Below, we formulate a simplified problem to show the direction of our future works on SFC embedding. We formulate the Network Utility Maximization problem (NUM) (Kelly et al. [66].) in a multi-commodity single path setting. The problem is joint optimization over rates and paths (single-path). Our NUM problem belongs to a class of more general flow problems, namely the multicommodity flow problem (MFP) (Leighton et al. [73]). However, our formulation is single-path, unlike the multi-path formulation in MFP (Leighton et al. [73]). In the NUM problem, we consider only network resources



(a) Scenario1: maximum link utility minimized



(b) Scenario2: national slice usage minimized

Figure 4.8: Containernet emulation - evaluation scenarios

(link capacity) and do not consider computational resources (at nodes) as in the SFC embedding problem.

Such a problem is computationally hard. The decision version of the m-commodity flow problem with fixed rates and the single-path setting is NP-complete (Drwal [34]). This is proved by the reduction from the decision version of the bin-packing problem.

Even et al. [37] show that even the decision version of the two-commodity integral flow problem is NP-complete by reduction from the SAT problem.

Since such a basic problem is computationally hard, we propose relaxation (polynomial-time solvable) and rounding-based techniques and provide several heuristics (in this work, we present two of them; more can be found in our previous work by Szaniawski and Nogalski [61]). We designed and implemented several heuristics and compared their performance with the exact linear programming solver. Because our utility function is nonlinear, we apply linear approximation within the optimization procedure. The results of the experiments demonstrate a trade-off between resource utilization efficiency and computation time.

4.6.2 Problem definition

Let's assume:

- $G = (V, E)$ an undirected graph representing a network, where V is a set of nodes and E a set of edges (this model could also be defined for a directed graph)
- $c : E \to [0, \infty)$ capacity function
- S a set of users
- $D = \{(s_k, t_k) \in V \times V : k \in S, \text{ and } s_k, t_k \in V\}$ a set of (source,target) pairs - demands for transmission
- $u : S \times [0, \infty) \to [0, \infty)$ utility function
- $u(k, x) = u_k(x)$ utility value of transmission rate x for the user $k \in S$.

The problem is to find a set of pairs $\{(x_k, P_k) : k \in S\}$, where x_k is a non-negative transmission rate assigned to the user (4.13), P_k is a transmission path in the graph G from s_k to t_k, assignments do not exceed (altogether) the edge capacity (4.14) and rates are maximized (4.12). Such an extended NUM problem is a generalization of the basic NUM formulated by Kelly et al. [66].

$$\max \sum_{k \in S} u_k(x_k) \tag{4.12}$$

We assume $\forall_{i,j,\in S} \;\; u_i = u_j$.

$$\forall_{k \in S} \;\; x_k \geq 0 \tag{4.13}$$

$$\forall_{\{u,v\} \in E(G)} \sum_{k, \text{ s.t. } \{u,v\} \in E(P_k)} x_k \leq c(u,v) \tag{4.14}$$

4.6.3 Model description

MILP NUM

In *MilpNum* each flow f_k takes a single path P_k from the source s_k to the target t_k. We introduce a real variable f_{kuv}, which indicates the amount of flow f_k passing via edge (u, v). The f_{kuv} indicates $u \to v$ flow direction and f_{kvu} reverse. We rewrite the constraint (4.14) as (4.15). We set a requirement that the flow is balanced (4.16), except source and terminal nodes (4.17)(4.18).

$$\forall_{\{u,v\} \in E(G)} \sum_{k, \text{ s.t. } \{u,v\} \in E(P_k)} f_{kuv} + f_{kvu} \leq c(\{u,v\}) \tag{4.15}$$

$$\forall_{k \in S} \;\; \forall_{u \in V \setminus \{s_k, t_k\}} \sum_{v:\{v,u\} \in E(G)} f_{kvu} = \sum_{w:\{u,w\} \in E(G)} f_{kuw} \tag{4.16}$$

$$\forall_{k \in S} \sum_{w:\{s_k,w\} \in E(G)} f_{ks_kw} = x_k \tag{4.17}$$

$$\forall_{k \in S} \sum_{v:\{v,t_k\} \in E(G)} f_{kvt_k} = x_k \tag{4.18}$$

We introduce a binary variable y_{kuv}, which indicates whether flow f_k is passing via edge (u, v) (4.19). The variable y_{kuv} indicatesWe have chosen $\alpha = 1/2$ for the experiment purpose $u \to v$ flow direction and y_{kvu} reverse.

$$\forall_{k \in S} \;\; \forall_{\{u,v\} \in E(G)} \;\; y_{kuv}, y_{kvu} \in \{0, 1\} \tag{4.19}$$

We bind the variable y_{kuv} with the variable f_{kuv} using two additional constraints (4.20)(4.21).

$$\forall_{kuv: k \in S, \{u,v\} \in E(G)} \;\; f_{kuv} \leq y_{kuv} \cdot c(\{u,v\}) \tag{4.20}$$

$$\forall_{kuv:k\in S,\{u,v\}\in E(G)} \; y_{kuv} \leq f_{kuv} \cdot L \tag{4.21}$$

where L is a large number.

In practice, *fair* allocations of capacity are needed. This can be obtained by choosing the utility function properly. In literature (Chiang et al. [24], Kelly et al. [66], Drwal [34]) most common is a family of functions:

$$u_k(x_k) = \begin{cases} w_k \frac{1}{1-\alpha} x_k^{1-\alpha}, & \alpha > 0, \alpha \neq 1 \\ w_k log(x_k), & \alpha = 1 \end{cases} \tag{4.22}$$

For the experiment purpose, we have chosen $\alpha = 1/2$, which gives the function (4.23). We assume the flow value to be non-negative (4.13).

$$\forall k \in S \;\; u_k(x) = u(x) = 2\sqrt{x} \tag{4.23}$$

We approximate the function (4.23) with three linear functions over range $x \in [0, 1]$ (see Figure 4.9). Such a choice stems from our assumption $\forall \{u, v\} \in E(G) \;\; c(u, v) = 1$. If the links differed in capacity, the approximation function could take values from 0 to the maximum edge capacity. For the approximation purpose, an additional equation is introduced (4.25). The goal function (4.12) is rewritten as (4.24).

$$\max \sum_{k\in S} (a_1 \cdot t_{1,k} + a_2 \cdot t_{2,k} + a_3 \cdot t_{3,k}) \tag{4.24}$$

where $a_1 = 12.00, a_2 = 3.00, a_3 = 1.3(3)$

$$\forall_{k\in S} \;\; x_k = t_{1,k} + t_{2,k} + t_{3,k} \tag{4.25}$$

where $0 \leq t_{1,k} \leq 1/36$, $0 \leq t_{2,k} \leq 8/36$, and $0 \leq t_{3,k}$

MILP NUM Relaxed

In *MilpNum Relaxed* we relax the y_{kuv} variable to allow the flow to split and flow via multiple paths (4.26). Such relaxation formulates the problem known in the literature as Multicommodity Flow Problem (MFP). Since this is a linear program, MFP can be solved in polynomial time (Pfetsch and Liebchen [97]).

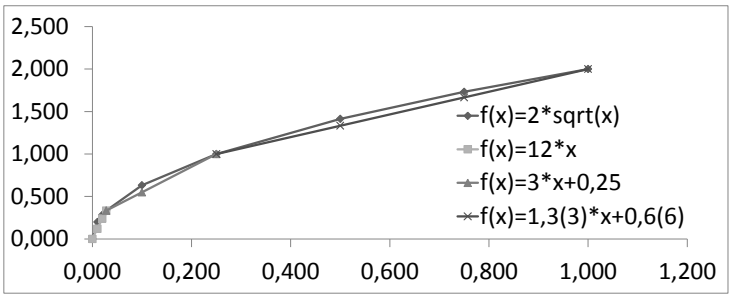


Figure 4.9: Approximation of utility function $u_k(x) = 2\sqrt{x}$

$$\forall_{k \in S} \forall_{\{u,v\} \in E(G)} \;\; y_{kuv}, y_{kvu} \in [0,1] \tag{4.26}$$

In this NUM formulation the following constraints hold { (4.13), (4.15), (4.16), (4.17), (4.18), (4.25) }, and constraints {(4.19),(4.20),(4.21)} are replaced with (4.26).

4.6.4 Algorithm description

To solve the NUM problem we have designed and implemented the below algorithms:

- *LPBestPathNum*(G, c, D) (see Algorithm 5)
- *IterativeLPBestPathNum*(G, c, D) (see Algorithm 6).

The algorithms assume the following *input* parameters:

- G graph representing a network
- c capacity function
- D a set of demands for transmission

and *output* parameters:

- A set of pairs $\{(x_k, P_k) : k \in S\}$ for which the constraints (4.13)(4.14) are satisfied
- $U = \sum_{k \in S} u(x_k)$ aggregated utility.

LP Best Path NUM

The algorithm *LPBestPathNum* (Algorithm 5) begins with solving linear program relaxation (line 2). Within the relaxed LP solution, for each $k \in S$ the single max path (max

min y_{kuv} on the path) is identified as P_k (line 5). The procedure is using the BFS (Breadth First Search) for that purpose. Finally, the LP problem is solved again taking fixed single paths identified (line 8) as a constraint. The algorithm is similar to the vertex projection method given in Wang et al. [125].

Algorithm 5 *LPBestPathNum*(G, c, D)

1: Assume a relaxed *problem* with goal (4.24) and constraints {(4.13),(4.15),(4.16),(4.17),(4.18),(4.25),(4.26)}
2: Solve the *problem*
3: Retrieve $\{y_{(k,u,v)\in S\times V\times V}\}$ and U from the *problem* solution
4: **for** $k \in S$ **do**
5: Using the BFS based procedure choose the max path P_k with value y_{P_k} (max min y_{kuv} on the path)
6: **end for**
7: Add constraints $\{P_k : k \in S\}$ to the *problem*
8: Solve the *problem*
9: **return** $(\{(x_k, P_k) : k \in S\}, U)$

Iterative LP Best Path NUM

The algorithm *IterativeLPBestPathNum* (Algorithm 6) constitutes $|S| + 1$ iterations. In each $\{1, .., |S|\}$ iteration, linear program relaxation is solved (line 6). Within the relaxed LP solution, for each $i \in S \setminus S'$ the single max path P_i (max min y_{iuv} on the path) is selected (line 9). Among the $|S \setminus S'|$-paths, the best (max) P_i is selected and added to the model (line 13) as a constraint (random selection, since there may be more than one best path (line 12)). The constraint forces i-flow to use only P_i edges in the next iterations. In the last $|S| + 1$ iteration the LP relaxation is solved assuming single fixed paths for all the $k \in S$ flows (line 16).

4.6.5 Computational results

Experiment Setup

The algorithms were programmed in Python 2.7 with the use of CPLEX Python library 12.8.0. The algorithm *MilpNum* uses CPLEX executable. All the simulations were run

Algorithm 6 *IterativeLPBestPathNum*(G, c, D)

Assume a relaxed *problem* with goal (4.24) and constraints {(4.13),(4.15),(4.16),(4.17),(4.18),(4.25),(4.26)}
Let's initiate a set of users with fixed single path $S' = \emptyset$
$stop = false$
while $(stop == false)$ **do**
  if $(S \setminus S'$ is not $\emptyset)$ **then**
    Solve the *problem*
    Retrieve $\{y_{(k,u,v)\in S\times V\times V}\}$ and U from the *problem* solution
    for $i \in S \setminus S'$ **do**
      Using the BFS based procedure choose the max path P_i with value y_{P_i} (max min y_{iuv} on the path)
    end for
    Let's define a set M as following $M = \{i \in S \setminus S' \colon y_{P_i} \cdot x_i == max\{y_{P_j} \cdot x_j\}_{j\in S\setminus S'}\}$
    Choose randomly $i \in M$, where probability of selecting an element i equals $\frac{1}{|M|}$
    Add constraint P_i to the *problem*
    $S' = S' \cup \{i\}$
  else
    Solve the *problem*
    $stop = true$
  end if
end while
return $(\{(x_k, P_k) : k \in S\}, U)$

on a PC with 2-core 2.7GHz and 8GB RAM.

Networks and capacity

The experiments were conducted on five types of networks:

- *Net7* with 7 nodes and 10 links (Figure 4.10, Jaskóła et al. [57]).
- *Net22* with 22 nodes and 40 links (Figure 4.11, Bianzino et al. [15]). Here, nodes 1-8 and T (which is a network exit node) were source-destination nodes (for definition of demands).

- *Net42* with 40 disjoint paths spanned between node 1 and 42 (Figure 4.12).
- *Net64* grid with 8x8 nodes (simple 3x3 grid to demonstrate a node numbering scheme is shown in Figure 4.13).
- *Net400* grid with 20x20 nodes.

In all the above networks each link had a capacity equal to one unit ($\forall_{\{u,v\}\in E(G)}\ c(\{u,v\}) = 1$). In total 24 experiments were conducted. The experiment parameters (network, demand) are given below.

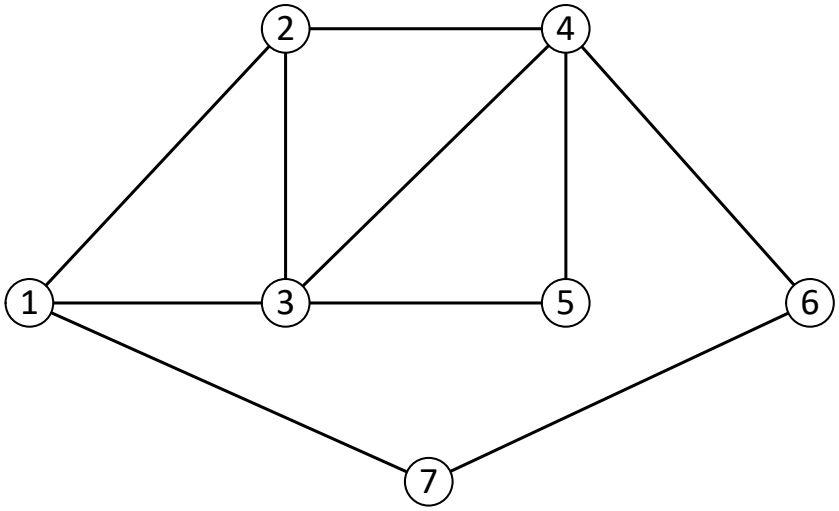


Figure 4.10: Network *Net7*

Experiments *Exp1-11* - Verification of algorithm performance for different network structure, size and number of demands

Assumptions

The following 11 experiments were conducted with various settings:

- *Exp1* - *Net7*, 10 demands [1];
- *Exp2* - *Net22*, 10 demands randomly selected [2];
- *Exp3/4* - *Net22*, 20 demands randomly selected [2];

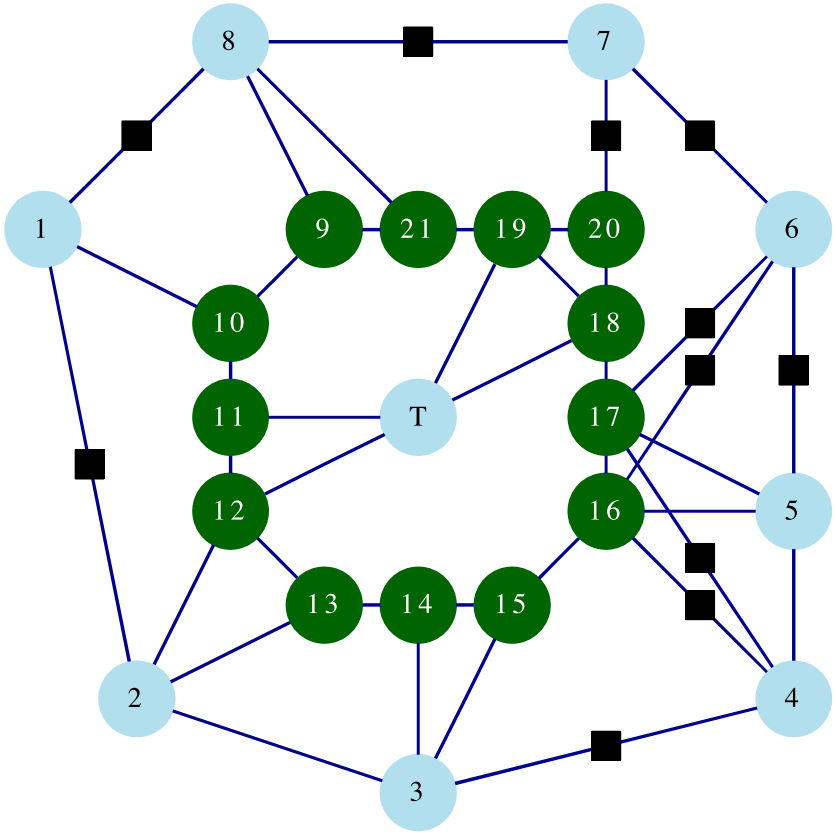


Figure 4.11: Network *Net22*

- *Exp5 - Net22*, 50 demands randomly selected [2];
- *Exp6 - Net64*, 20 demands [3];
- *Exp7/8 - Net64*, 20 demands randomly selected;
- *Exp9/10 - Net64*, 20 demands randomly selected [4];
- *Exp11 - Net42*, 20 demands each with ($s_k = 1$, $t_k = 42$).

Detailed assumptions:

[1] set of (s_k, t_k) pairs is the following {(2,6), (2,3), (2,6), (2,5), (3,6), (3,6), (3,5), (6,3), (2,6), (3,4)} (as given in [57])

[2] 50% of the (s_k, t_k) pairs constitute internal traffic (between nodes 1-8), and 50% of the pairs constitute external traffic (between nodes 1-8 and T)

[3] 10 pairs with $(s_k = 1, t_k = 64)$ and 10 pairs with $(s_k = 8, t_k = 57)$

[4] Traffic sources and targets are located on the grid edge, and are randomly selected.

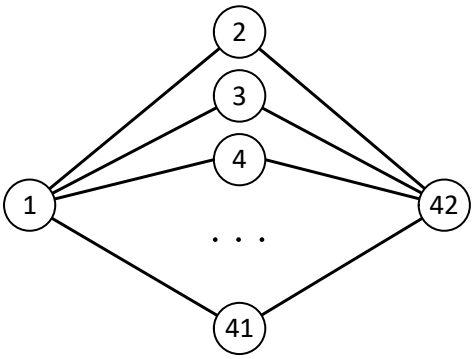


Figure 4.12: Network *Net42*

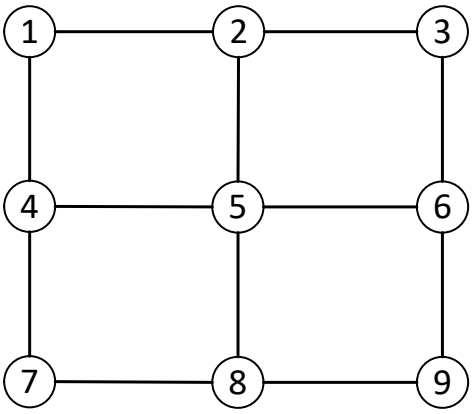


Figure 4.13: Grid network *Net9*

Additionally, in each experiment, the algorithms: *LPBestPathNum* and *IterativeLPBestPathNum* were run 20 times and the average values of objective U_{avg} and execution time T_{avg} are presented on the figures.

Results

- ***LPBestPathNum* vs *IterativeLPBestPathNum.*** In *Exp1-11* the average observed objective values of *IterativeLPBestPathNum* were higher than values of *LPBestPathNum* by up to 14.5%, except for the parallel edge network (*Net42*) by 31% (Figure 4.14, *Exp11*). However, *IterativeLPBestPathNum* exposed higher execution time due to additional $|S| - 1$ iterations (Figure 4.15, Table 4.4).

- ***IterativeLPBestPathNum* vs *MilpNum.*** In *Exp1-6,11* the optimal objective values (*MilpNum*) were higher than *IterativeLPBestPathNum* solution by not more than 1.5%. In *Exp7,9-10* - by not more than 4.8%. In *Exp8* - by 7.9%. However,

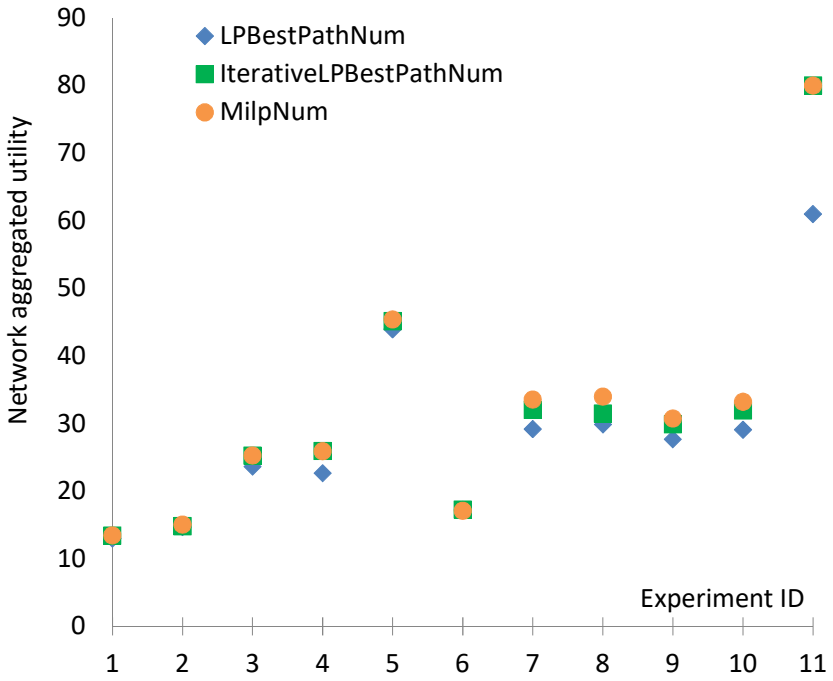


Figure 4.14: Network aggregated utility U (experiments *Exp1-11*)

the observed computation time of *MilpNum* for *Net64* (*Exp7-10*) is very significant (in hours) (Table 4.4). A possible, way around for *MilpNum* would be to configure the automatic termination when reaching a defined solution gap. As an example, in the experiment *Exp9* *MilpNum* gave the value 26,22 (gap 16,95%) in 12s and 29 (5,75%) in 90s. *IterativeLPBestPathNum* gave 29,928 in 8,4s. In *Exp10* *MilpNum* gave 30 (11,11%) in 12s and 31,66 (5%) in 268s. *IterativeLPBestPathNum* gave 31,962 in 8,2s.

Experiments *Exp12-21* - Verification of algorithm performance for different number of demands

Assumptions

Additional 10 experiments on *Net64* were conducted to demonstrate an increase in utility and time upon an increase in the number of demands:

- *Exp12* - *Net64*, 10 demands [6];

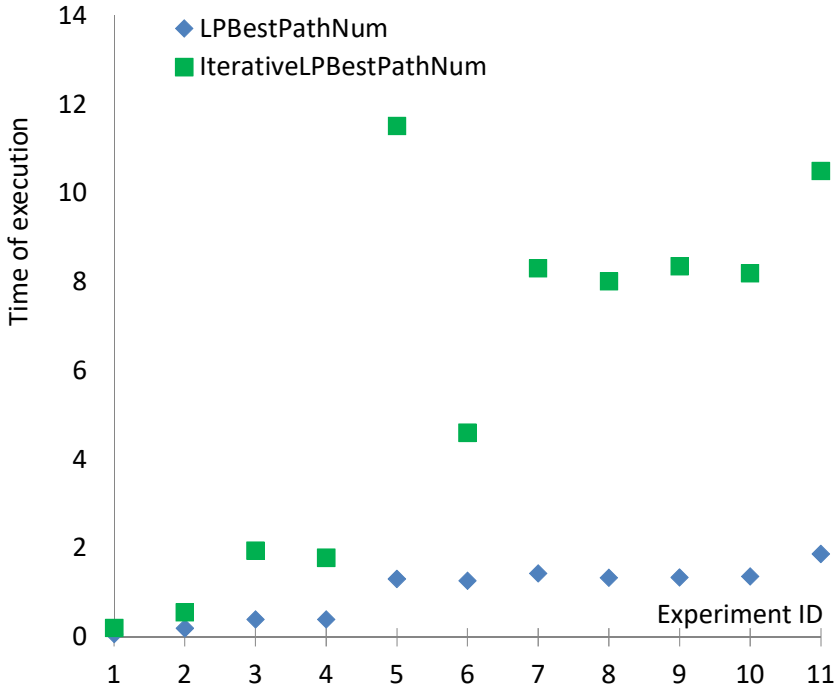


Figure 4.15: Time of execution T [s] (experiments *Exp1-11*)

Table 4.4: Execution time [s] (experiments *Exp1-11*)

Experiment ID	*LPBestPathNum*	*IterativeLPBestPathNum*	*MilpNum*
	$T_{avg}[s]$	$T_{avg}[s]$	$T[s]$
1	0,067	0,204	0,08
2	0,199	0,558	0,33
3	0,398	1,949	4,09
4	0,397	1,786	1,14
5	1,309	11,517	8,05
6	1,268	4,604	15,3
7	1,435	8,31	1030,62
8	1,337	8,018	68073
9	1,344	8,359	4254,55
10	1,366	8,201	9168,24
11	1,873	10,502	0,25

- *Exp13* - *Net64*, 20 demands [6];
- ...
- *Exp21* - *Net64*, 100 demands [6];

Detailed assumptions:

[6] Traffic sources and targets were located on the grid edge, and were randomly selected.

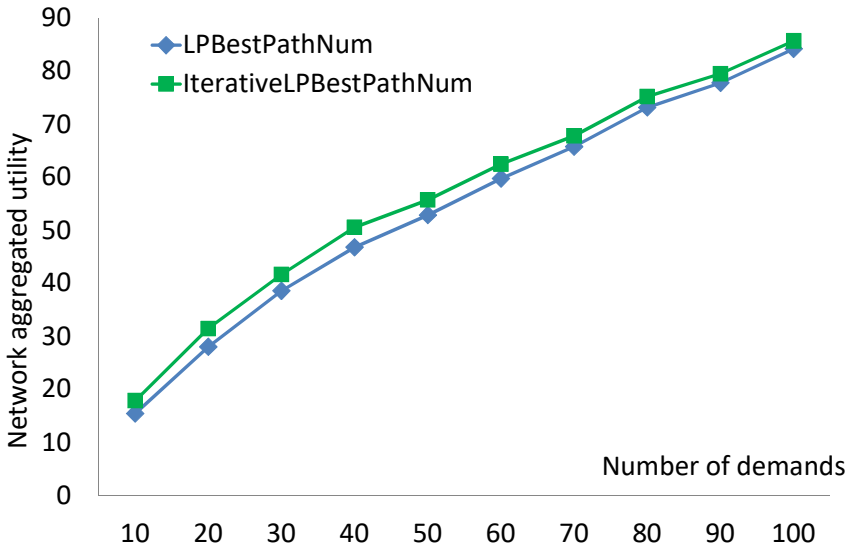


Figure 4.16: Network aggregated utility (experiments *Exp12-21*)

Each smaller demand set is a subset of a larger set.

Results

- ***IterativeLPBestPathNum* vs *LPBestPathNum*.** In *Exp12-21* the average objective values of *IterativeLPBestPathNum* and *LPBestPathNum* increased when the number of demands increased. The utility values of *IterativeLPBestPathNum* were higher than those of *LPBestPathNum* (Figure 4.16). The observed difference became smaller when the number of demands increased. As observed, from 16% for 10 demands to 2% for 100 demands.

- However, *IterativeLPBestPathNum* requires much longer computation times than *LPBestPathNum*, with an increasing number of demands (Figure 4.17).

Experiments *Exp22-24* - Verification of algorithm performance for a very large grid network

Assumptions

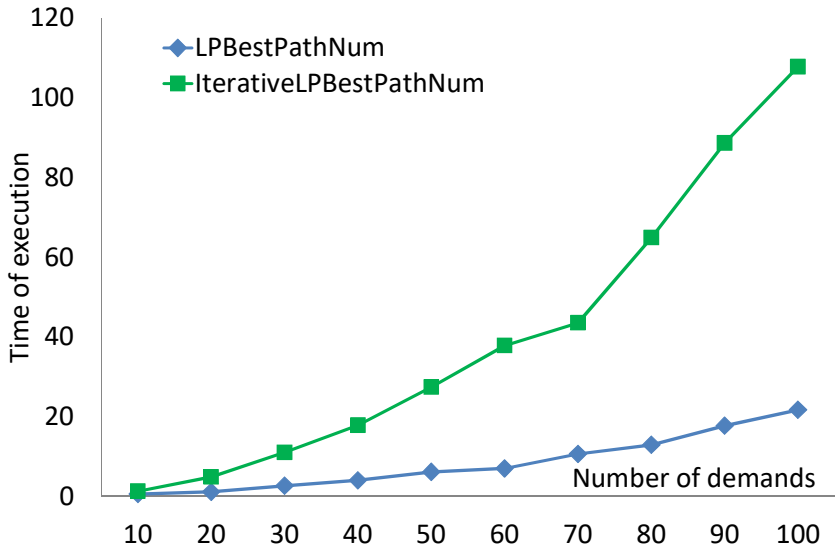


Figure 4.17: Time of execution [s] (experiments *Exp12-21*)

Finally, additional 3 experiments were conducted to capture utility and time for the network *Net400*:

- *Exp22* - *Net400*, 20 demands [7];
- *Exp23* - *Net400*, 50 demands [7];
- *Exp24* - *Net400*, 100 demands [7];

Detailed assumptions:

[7] Traffic sources and targets were located on the grid edge, and were randomly selected. Each smaller demand set is a subset of a larger set.

Results

- ***IterativeLPBestPathNum* vs *LPBestPathNum*.** The utility difference between *IterativeLPBestPathNum* and *LPBestPathNum* was more visible for a very large grid network (20x20 nodes), as observed 14.7% (20 demands) - 21.8% (50 demands) in favor of the former (Table 4.5).
- However, the computation time of *IterativeLPBestPathNum* for a very large network as *Net400* and large number of demands may not be practically attainable (Table 4.5).

Table 4.5: Utility and Time for a very large grid network (experiments *Exp22-24*)

Experiment ID	Number of demands	*LPBestPathNum*		*IterativeLPBestPathNum*	
		U_{avg}	$T_{avg}[s]$	U_{avg}	$T_{avg}[s]$
22	20	31,54	27,09	36,18	139,78
23	50	63,59	341,51	77,46	6152,44
24	100	109,40	5129,93	-	-

4.6.6 Conclusions on Speedup methods

Since the problem is NP-hard, search for optimum with the ILP model (*MilpNum*) may not be practically attainable even for medium-sized grid networks (8x8 nodes) and a small number of demands (~ 20).

The experiments show that the algorithm *LPBestPathNum* demonstrates a reasonably good trade-off between computing time and precision of the utility value.

As observed, the LP-based heuristic *IterativeLPBestPathNum* demonstrates up to $\sim 15\%$ higher utility, on networks with up to 64 nodes and 50 demands, but it is slower, in comparison to *LPBestPathNum*. The difference in utility may be a few percent bigger for large grid networks (20x20 nodes) and a high number of demands (~ 50), and even more for a special type of network - parallel edges. The bigger difference is caused by the edge allocation conflict. The *IterativeLPBestPathNum* reduces the probability of such conflicts by iterative path fixing (a single path fixed for each iteration).
The utility values of *IterativeLPBestPathNum*, on network with up to 64 nodes and 50 demands, are lower than *MilpNum* by only $\sim 2\%$ on average.
The successful *IterativeLPBestPathNum* computations were conducted for grid networks with up to 20x20 nodes and 50 demands. For comparison, the successful and not terminated *MilpNum* computations were conducted for grid networks with up to 8x8 nodes and 20 demands.

4.7 Conclusions

Since modern coalition networks are expected to simultaneously provide connectivity (i.e., QoS-enabled paths) and computational capabilities (i.e., network functions), it is neces-

sary to use efficient methods for embedding E2E slices. Such efficient methods are crucial in security scenarios to properly handle network and compute resources and protect the network against cyber-attacks.

The computational results reveal that:

- Extensive simulations of the ILP model and heuristic, on realistic topologies show the correct trend of the goal values (TE & SD) as the α parameter (trade-off) and the number of demands change.
- In terms of goal value, the ILP model is more efficient than the heuristic (it uses less network resources to embed the demand set). The observed average gap is up to 27% (8 demands) on a network of practical size (based on "*nsfnet*").
- On the other hand, the heuristic is much faster. On a large-scale network (based on "*cost266*") the observed average execution time is below 35 s (60 demands), while the ILP model's average run time is up to 1730 s (for only 8 demands).
- While the ILP model efficiently solves SFC problems of a practical size, the heuristic can solve large-scale problems very fast with decent efficiency, leading to a trade-off between the two.

As for future work, the following extensions to our methodology are worth considering:

- In the evaluation, we assume the simplification that the intra-domain topology fully contributes to the federated topology. In future work, we plan to address the intra-domain SFC mapping problem algorithmically.
- Investigate how different ways of defining a federated topology from a local topology (taking into account local slices and compute nodes) affect the performance and efficiency of the entire two-level system in terms of resource utilization and admission ratios.

Bibliography

[1] R. A. Addad, M. Bagaa, T. Taleb, D. L. C. Dutra, and H. Flinck. "Optimization Model for Cross-Domain Network Slices in 5G Networks". *IEEE Trans. Mob. Comput.* 19.5 (2020). doi: 10.1109/TMC.2019.2905599, pp. 1156–1169.

[2] B. Addis, D. Belabed, M. Bouet, and S. Secci. "Virtual network functions placement and routing optimization". *4th IEEE International Conference on Cloud Networking, CloudNet 2015, Niagara Falls, ON, Canada, October 5-7, 2015*. doi: 10.1109/CloudNet.2015.7335301. IEEE, 2015, pp. 171–177.

[3] Y. Afek, A. Bremler-Barr, and S. Landau Feibish. "Automated signature extraction for high volume attacks". *Architectures for Networking and Communications Systems*. doi: 10.1109/ANCS.2013.6665197. 2013, pp. 147–156.

[4] Y. Al-Dunainawi, B. R. Al-Kaseem, and H. S. Al-Raweshidy. "Optimized Artificial Intelligence Model for DDoS Detection in SDN Environment". *IEEE Access* 11 (2023). doi: 10.1109/ACCESS.2023.3319214, pp. 106733–106748.

[5] M. Aljebreen, H. A. Mengash, M. A. Arasi, S. S. Aljameel, A. S. Salama, and M. A. Hamza. "Enhancing DDoS Attack Detection Using Snake Optimizer With Ensemble Learning on Internet of Things Environment". *IEEE Access* 11 (2023). doi: 10.1109/ACCESS.2023.3318316, pp. 104745–104753.

[6] A. Almadhor, A. Altalbe, I. Bouazzi, A. A. Hejaili, and N. Kryvinska. "Strengthening network DDOS attack detection in heterogeneous IoT environment with federated XAI learning approach". *Scientific Reports* 14.1 (2024). doi: 10.1038/s41598-024-76016-6, p. 24322.

[7] D. S. Altner, Ö. Ergun, and N. A. Uhan. "The Maximum Flow Network Interdiction Problem: Valid inequalities, integrality gaps, and approximability". *Oper. Res. Lett.* 38.1 (2010). doi: 10.1016/j.orl.2009.09.013, pp. 33–38.

[8] K. S. Amorim and G. S. Pavani. "Ant Colony Optimization-based distributed multilayer routing and restoration in IP/MPLS over optical networks". *Computer Networks* 185 (2021). doi: 10.1016/j.comnet.2020.107747, p. 107747.

[9] B. Armbruster, J. C. Smith, and K. Park. "A packet filter placement problem with application to defense against denial of service attacks". *European Journal of Operational Research* 176.2 (2007). doi: 10.1016/j.ejor.2005.09.031, pp. 1283–1292.

[10] M. V. O. de Assis, A. H. Hamamoto, T. Abrão, and M. L. P. Jr. "A Game Theoretical Based System Using Holt-Winters and Genetic Algorithm With Fuzzy Logic for DoS/DDoS Mitigation on SDN Networks". *IEEE Access* 5 (2017). doi: 10.1109/ACCESS.2017.2702341, pp. 9485–9496.

[11] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. "Dynamic Controller Provisioning in Software Defined Networks". *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*. doi: 10.1109/CNSM.2013.6727805. 2013, pp. 18–25.

[12] M. T. Beck and J. F. Botero. "Scalable and coordinated allocation of service function chains". *Comput. Commun.* 102 (2017). doi: 10.1016/j.comcom.2016.09.010, pp. 78–88.

[13] D. Belabed, M. Bouet, and V. Conan. "Centralized Defense Using Smart Routing Against Link-Flooding Attacks". *2nd Cyber Security in Networking Conference, CSNet 2018, Paris, France, October 24-26, 2018*. Ed. by P. Hebrard and S. Ghernaouti. doi: 10.1109/CSNET.2018.8602966. IEEE, 2018, pp. 1–8.

[14] A. Belbekkouche, M. M. Hasan, and A. Karmouch. "Resource Discovery and Allocation in Network Virtualization". *IEEE Commun. Surv. Tutorials* 14.4 (2012). doi: 10.1109/SURV.2011.122811.00060, pp. 1114–1128.

[15] A. Bianzino, L. Chiaraviglio, M. Mellia, and J. Rougier. "GRiDA: GReen Distributed Algorithm for energy-efficient IP backbone networks". *Computer Networks* 56.14 (2012). doi: 10.1016/j.comnet.2012.06.011, pp. 3219–3232.

[16] P. Blazek, T. Gerlich, and Z. Martinasek. "Scalable DDoS Mitigation System". *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*. doi: 10.1109/TSP.2019.8768869. 2019, pp. 617–620.

[17] A. Bonguet and M. Bellaiche. "A Survey of Denial-of-Service and Distributed Denial of Service Attacks and Defenses in Cloud Computing". *Future Internet* 9.3 (2017). doi: 10.3390/fi9030043, p. 43.

[18] J. F. Botero, M. Molina, X. Hesselbach-Serra, and J. R. Amazonas. "A novel paths algebra-based strategy to flexibly solve the link mapping stage of VNE problems". *J. Netw. Comput. Appl.* 36.6 (2013). doi: 10.1016/j.jnca.2013.02.029, pp. 1735–1752.

[19] E. Calle, S. G. Cosgaya, D. Martínez, and M. Pióro. "Solving The Backup Controller Placement Problem In SDN Under Simultaneous Targeted Attacks". *11th International Workshop on Resilient Networks Design and Modeling, RNDM 2019, Nicosia, Cyprus, October 14-16, 2019*. doi: 10.1109/RNDM48015.2019.8949096. IEEE, 2019, pp. 1–7.

[20] E. Calle, D. Martínez, M. Mycek, and M. Pióro. "Resilient backup controller placement in distributed SDN under critical targeted attacks". *Int. J. Crit. Infrastructure Prot.* 33 (2021). doi: 10.1016/j.ijcip.2021.100422, p. 100422.

[21] C. Cameron, C. Patsios, P. C. Taylor, and Z. Pourmirza. "Using Self-Organizing Architectures to Mitigate the Impacts of Denial-of-Service Attacks on Voltage Control Schemes". *IEEE Transactions on Smart Grid* 10.3 (2019). doi: 10.1109/TSG.2018.2817046, pp. 3010–3019.

[22] A. Cetinkaya, H. Ishii, and T. Hayakawa. "An Overview on Denial-of-Service Attacks in Control Systems: Attack Models and Security Analyses". *Entropy* 21.2 (2019). doi: 10.3390/e21020210, p. 210.

[23] R. Chaudhary and N. Kumar. "PARC: Placement Availability Resilient Controller Scheme for Software-Defined Datacenters". *IEEE Transactions on Vehicular Technology* 69.8 (2020). doi: 10.1109/TVT.2020.2999072, pp. 8985–9001.

[24] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. "Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures". *Proceedings of the IEEE* 95.1 (2007). doi: 10.1109/JPROC.2006.887322, pp. 255–312.

[25] J. Chou, C. Shih, W. Wang, and K. Huang. "IoT Sensing Networks for Gait Velocity Measurement". *Int. J. Appl. Math. Comput. Sci.* 29.2 (2019). doi: 10.2478/amcs-2019-0018, pp. 245–259.

[26] G. Clapp, R. A. Skoog, A. C. Von Lehmen, and B. Wilson. "Management of switched systems at 100 Tbps: The DARPA CORONET program". *2009 International Conference on Photonics in Switching*. doi: 10.1109/PS.2009.5307846. 2009, pp. 1–4.

[27] P. J. Criscuolo. "Distributed Denial of Service Trin00, Tribe Flood Network, Tribe Flood Network 2000, And Stacheldraht, CIAC-2319". *Department of Energy Computer Incident Advisory Capability (CIAC), Lawrence Livermore National Laboratory* (2000).

[28] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. doi: 10.1007/978-3-319-21275-3. Springer, 2015.

[29] A. A. Daya, M. A. Salahuddin, N. Limam, and R. Boutaba. "BotChase: Graph-Based Bot Detection Using Machine Learning". *IEEE Trans. Netw. Serv. Manag.* 17.1 (2020). doi: 10.1109/TNSM.2020.2972405, pp. 15–29.

[30] M. DeVos and D. A. Kent. *Game theory: a playful introduction*. Vol. 80. Bibliografija: str. 335-338 Kazali. American Mathematical Society, 2016.

[31] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella. "Towards an elastic distributed SDN controller". *Comput. Commun. Rev.* 43.4 (2013). doi: 10.1145/2534169.2491193, pp. 7–12.

[32] S. Dou, L. Qi, C. Yao, and Z. Guo. "Exploring the Impact of Critical Programmability on Controller Placement for Software-Defined Wide Area Networks". *IEEE/ACM Trans. Netw.* 31.6 (2023). doi: 10.1109/TNET.2023.3252639, pp. 2575–2588.

[33] C. Douligeris and A. Mitrokotsa. "DDoS attacks and defense mechanisms: classification and state-of-the-art". *Computer Networks* 44.5 (2004). doi: 10.1016/j.comnet.2003.10.003, pp. 643–666.

[34] M. Drwal. "Approximation algorithms for utility-maximizing network design problem". *Progress in Systems Engineering: Proceedings of the Twenty-Third International Conference on Systems Engineering*. Ed. by H. Selvaraj et al. Vol. 366. Advances in Intelligent Systems and Computing. doi: 10.1007/978-3-319-08422-0-60. Springer International Publishing, Cham, 2015.

[35] K. El Defrawy, A. Markopoulou, and K. Argyraki. "Optimal Allocation of Filters against DDoS Attacks". *2007 Information Theory and Applications Workshop*. doi: 10.1109/ITA.2007.4357573. 2007, pp. 140–149.

[36] A. El-Amine, O. Brun, S. Abdellatif, and P. Berthou. "Shortening the Deployment Time of SFCs by Adaptively Querying Resource Providers". *IEEE Global Communications Conference, GLOBECOM 2021, Madrid, Spain, December 7-11, 2021*. doi: 10.1109/GLOBECOM46510.2021.9685356. IEEE, 2021, pp. 1–6.

[37] S. Even, A. Itai, and A. Shamir. "On the complexity of time table and multi-commodity flow problems". *16th Annual Symposium on Foundations of Computer Science 13-15 October 1975*. Ed. by IEEE. doi: 10.1109/SFCS.1975.21. 1975.

[38] L. Faramondi, R. Setola, S. Panzieri, F. Pascucci, and G. Oliva. "Finding critical nodes in infrastructure networks". *Int. J. Crit. Infrastructure Prot.* 20 (2018). doi: 10.1016/j.ijcip.2017.11.004, pp. 3–15.

[39] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. D. Bailey. "Bohatei: Flexible and Elastic DDoS Defense". *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. Ed. by J. Jung and T. Holz. USENIX Association, 2015, pp. 817–832.

[40] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. "Virtual Network Embedding: A Survey". *IEEE Commun. Surv. Tutorials* 15.4 (2013). doi: 10.1109/SURV.2013.013013.00155, pp. 1888–1906.

[41] L. R. Ford and D. R. Fulkerson. "Maximal Flow Through a Network". *Canadian Journal of Mathematics* 8 (1956), pp. 399–404.

[42] J. B. García, J. Vergara, S. Ríos-Guiral, C. Garzón, S. A. Gutierrez, J. F. Botero, J. L. Q. Arroyo, and J. A. P. Díaz. "Exploring Traffic Patterns Through Network Programmability: Introducing SDNFLow, a Comprehensive OpenFlow-Based Statistics Dataset for Attack Detection". *IEEE Access* 12 (2024). doi: 10.1109/ACCESS.2024.3378271, pp. 42163–42180.

[43] N. Garg, V. V. Vazirani, and M. Yannakakis. "Multiway Cuts in Directed and Node Weighted Graphs". *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*. Ed. by

S. Abiteboul and E. Shamir. Vol. 820. Lecture Notes in Computer Science. doi: 10.1007/3-540-58201-0_92. Springer, 1994, pp. 487–498.

[44] J. Gera and B. P. Battula. "Detection of spoofed and non-spoofed DDoS attacks and discriminating them from flash crowds". *EURASIP Journal on Information Security* 2018.1 (2018). doi: 10.1186/s13635-018-0079-6, p. 9.

[45] J. Gil Herrera and J. F. Botero. "Resource Allocation in NFV: A Comprehensive Survey". *IEEE Transactions on Network and Service Management* 13.3 (2016). doi: 10.1109/TNSM.2016.2598420, pp. 518–532.

[46] D. Gkounis, V. Kotronis, C. Liaskos, and X. A. Dimitropoulos. "On the Interplay of Link-Flooding Attacks and Traffic Engineering". *Comput. Commun. Rev.* 46.2 (2016). doi: 10.1145/2935634.2935636, pp. 5–11.

[47] A. V. Goldberg and R. E. Tarjan. "Efficient maximum flow algorithms". *Commun. ACM* 57.8 (2014). doi: 10.1145/2628036, pp. 82–89.

[48] M. He, A. Varasteh, and W. Kellerer. "Toward a Flexible Design of SDN Dynamic Control Plane: An Online Optimization Approach". *IEEE Transactions on Network and Service Management* 16.4 (2019). doi: 10.1109/TNSM.2019.2935160, pp. 1694–1708.

[49] B. Heller, R. Sherwood, and N. McKeown. "The controller placement problem". *Comput. Commun. Rev.* 42.4 (2012). doi: 10.1145/2377677.2377767, pp. 473–478.

[50] M. Hemmati, J. Cole Smith, and M. T. Thai. "A cutting-plane algorithm for solving a weighted influence interdiction problem". *Computational Optimization and Applications* 57.1 (2014). doi: 10.1007/s10589-013-9589-9, pp. 71–104.

[51] L. Huang, J. Ran, W. Wang, T. Yang, and Y. Xiang. "A multi-channel anomaly detection method with feature selection and multi-scale analysis". *Comput. Networks* 185 (2021). doi: 10.1016/j.comnet.2020.107645, p. 107645.

[52] A. Huseinović, S. Mrdović, K. Bicakci, and S. Uludag. "A Survey of Denial-of-Service Attacks and Solutions in the Smart Grid". *IEEE Access* 8 (2020). doi: 10.1109/ACCESS.2020.3026923, pp. 177447–177470.

[53] R.-H. Hwang, M.-C. Peng, C.-W. Huang, P.-C. Lin, and V.-L. Nguyen. "An Unsupervised Deep Learning Model for Early Network Traffic Anomaly Detection". *IEEE Access* 8 (2020). doi: 10.1109/ACCESS.2020.2973023, pp. 30387–30399.

[54] A. A. Z. Ibrahim, F. Hashim, N. K. Noordin, A. Sali, K. Navaie, and S. M. E. Fadul. "Heuristic Resource Allocation Algorithm for Controller Placement in Multi-Control 5G Based on SDN/NFV Architecture". *IEEE Access* 9 (2021). doi: 10.1109/ACCESS.2020.3047210, pp. 2602–2617.

[55] M. H. Islam, K. Nadeem, and S. A. Khan. "Efficient placement of sensors for detection against distributed denial of service attack". *2008 International Conference on Innovations in Information Technology, IIT 2008* (2008). doi: 10.1109/INNOVATIONS.2008.4781681, pp. 653–657.

[56] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh. "A survey and classification of the security anomaly detection mechanisms in software defined networks". *Clust. Comput.* 24.2 (2021). doi: 10.1007/s10586-020-03184-1, pp. 1235–1253.

[57] P. Jaskóła, P. Arabas, and A. Karbowski. "Simultaneous routing and flow rate optimization in energy-aware computer networks". *International Journal of Applied Mathematics and Computer Science* 26.1 (2016). doi: 10.1515/amcs-2016-0016, pp. 231–243.

[58] S. B. Jeong, Y. Choi, and S. Kim. "An Effective Placement of Detection Systems for Distributed Attack Detection in Large Scale Networks". *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*. Ed. by C. H. Lim and M. Yung. Vol. 3325. Lecture Notes in Computer Science. doi: 10.1007/978-3-540-31815-6_17. Springer, 2004, pp. 204–210.

[59] J. Jiao, B. Ye, Y. Zhao, R. J. Stones, G. Wang, X. Liu, S. Wang, and G. Xie. "Detecting TCP-Based DDoS Attacks in Baidu Cloud Computing Data Centers". *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017*. doi: 10.1109/SRDS.2017.37. IEEE Computer Society, 2017, pp. 256–258.

[60] K. Junosza-Szaniawski, D. Nogalski, and A. Wójcik. "Exact and approximation algorithms for sensor placement against DDoS attacks". *2020 15th Conference on Computer Science and Information Systems (FedCSIS), 13th International Workshop on Computational Optimization, Sofia, Bulgaria*. doi: 10.15439/2020F106. 2020, pp. 295–301.

[61] K. Junosza-Szaniawski and D. Nogalski. "Exact and approximation algorithms for joint routing and flow rate optimization". *Preprints of Communication Papers of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019, Leipzig, Germany, September 1-4, 2019*. Ed. by M. Ganzha, L. A. Maciaszek, and M. Paprzycki. Vol. 20. Annals of Computer Science and Information Systems. doi: 10.15439/2019F85. 2019, pp. 29–36.

[62] K. Junosza-Szaniawski and D. Nogalski. "Game-Theoretic Approach to Attack Planning and Controller Placement in Software Defined Networks". *International Conference on Military Communications and Information Systems, ICMCIS 2023, Skopje, North Macedonia, May 16-17, 2023*. 10.1109/ICMCIS59922.2023.10253594. IEEE, 2023, pp. 1–8.

[63] K. Junosza-Szaniawski, D. Nogalski, and P. Rzazewski. "Exact and Approximation Algorithms for Sensor Placement Against DDoS Attacks". *International Journal of Applied Mathematics and Computer Science* 32.1 (2022). doi: 10.34768/amcs-2022-0004, pp. 35–49.

[64] M. G. Kallitsis, S. A. Stoev, S. Bhattacharya, and G. Michailidis. "AMON: An Open Source Architecture for Online Monitoring, Statistical Analysis, and Forensics of Multi-Gigabit Streams". *IEEE J. Sel. Areas Commun.* 34.6 (2016). doi: 10.1109/JSAC.2016.2558958, pp. 1834–1848.

[65] M. S. Kang, S. B. Lee, and V. D. Gligor. "The Crossfire Attack". *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. doi: 10.1109/SP.2013.19. 2013, pp. 127–141.

[66] F. Kelly, A. Maulloo, and D. Tan. "Rate control for communication networks: shadow prices, proportional fairness and stability". *Journal of the Operational Research Society* 49.3 (1998). doi: 10.1057/palgrave.jors.2600523, pp. 237–252.

[67] B. A. Khalaf, S. A. Mostafa, A. Mustapha, M. A. Mohammed, and W. M. Abduallah. "Comprehensive Review of Artificial Intelligence and Statistical Approaches in Distributed Denial of Service Attack and Defense Methods". *IEEE Access* 7 (2019). doi: 10.1109/ACCESS.2019.2908998, pp. 51691–51713.

[68] A. Khapalov. "Source localization and sensor placement in environmental monitoring". *Int. J. Appl. Math. Comput. Sci.* 20.3 (2010). doi: 10.2478/v10006-010-0033-3, pp. 445–458.

[69] B. P. R. Killi and S. V. Rao. "Capacitated Next Controller Placement in Software Defined Networks". *IEEE Trans. Netw. Serv. Manag.* 14.3 (2017). doi: 10.1109/TNSM.2017.2720699, pp. 514–527.

[70] H. Ko, D. Suh, H. Baek, S. Pack, and J. Kwak. "Optimal placement of service function in service function chaining". *Eighth International Conference on Ubiquitous and Future Networks, ICUFN 2016, Vienna, Austria, July 5-8, 2016*. doi: 10.1109/ICUFN.2016.7536993. IEEE, 2016, pp. 102–105.

[71] G. Kumar Saini and G. Somani. "Is There a DDoS?: System+Application Variable Monitoring to Ascertain the Attack Presence". *IEEE Transactions on Network and Service Management* 21.6 (2024). doi: 10.1109/TNSM.2024.3451613, pp. 6899–6908.

[72] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks". *IEEE Trans. Netw. Serv. Manag.* 12.1 (2015). doi: 10.1109/TNSM.2015.2402432, pp. 4–17.

[73] T. Leighton and S. Rao. "Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms". *Journal of the ACM* 46.6 (1999), pp. 787–832.

[74] H. Li, R. E. D. Grande, and A. Boukerche. "An Efficient CPP Solution for Resilience-Oriented SDN Controller Deployment". *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, Orlando / Buena Vista, FL, USA, May 29 - June 2, 2017*. doi: 10.1109/IPDPSW.2017.161. IEEE Computer Society, 2017, pp. 540–549.

[75] H. Li and L. Wang. "Online orchestration of cooperative defense against DDoS attacks for 5G MEC". *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. doi: 10.1109/WCNC.2018.8377309. 2018, pp. 1–6.

[76] C. Liaskos and S. Ioannidis. "Network Topology Effects on the Detectability of Crossfire Attacks". *IEEE Transactions on Information Forensics and Security* 13.7 (2018). doi: 10.1109/TIFS.2018.2799425, pp. 1682–1695.

[77] X. Liu, J. Ren, H. He, Q. Wang, and C. Song. "Low-rate DDoS attacks detection method using data compression and behavior divergence measurement". *Comput. Secur.* 100 (2021). doi: 10.1016/j.cose.2020.102107, p. 102107.

[78] A. N. Maksimovic, V. Nikolic, D. Vidojevic, D. M. Randelovic, S. M. Djukanovic, and D. M. Randjelovic. "Using Triple Modular Redundancy for Threshold Determination in DDOS Intrusion Detection Systems". *IEEE Access* 12 (2024). doi: 10.1109/ACCESS.2024.3384380, pp. 53785–53804.

[79] V. de Miranda Rios, P. R. M. Inácio, D. Magoni, and M. M. Freire. "Detection of reduction-of-quality DDoS attacks using Fuzzy Logic and machine learning algorithms". *Comput. Networks* 186 (2021). doi: 10.1016/j.comnet.2020.107792, p. 107792.

[80] J. Mirkovic and P. Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms". *ACM SIGCOMM Computer Communication Review* 34.2 (2004). doi: 10.1145/997150.997156, p. 39.

[81] Q. Monnet, L. Mokdad, P. Ballarini, Y. Hammal, and J. Ben-Othman. "DoS detection in WSNs: Energy-efficient methods for selecting monitoring nodes". *Concurr. Comput. Pract. Exp.* 29.23 (2017). doi: 10.1002/cpe.4266.

[82] N. I. Mowla, I. Doh, and K. Chae. "CSDSM: Cognitive switch-based DDoS sensing and mitigation in SDN-driven CDNi word". *Comput. Sci. Inf. Syst.* 15.1 (2018). doi: 10.2298/CSIS170328044M, pp. 163–185.

[83] A. Murray, A. Arulselvan, M. Roper, M. Cashmore, S. K. Mohalik, I. Burdick, and S. David. "The Cost of Quality of Service: SLA Aware VNF Placement and Routing Using Column Generation". *2023 13th International Workshop on Resilient Networks Design and Modeling (RNDM)*. doi: 10.1109/RNDM59149.2023.10293056. 2023, pp. 1–8.

[84] M. Mycek, M. Pióro, A. Tomaszewski, and A. de Sousa. "Optimizing primary and backup SDN controllers' placement resilient to node-targeted attacks". *17th International Conference on Network and Service Management, CNSM 2021, Izmir, Turkey, October 25-29, 2021*. Ed. by P. Chemouil, M. Ulema, S. Clayman, M. Sayit, C. Çetinkaya, and S. Secci. doi: 10.23919/CNSM52442.2021.9615578. IEEE, 2021, pp. 397–401.

[85] D. Nogalski, D. Belabed, A. Triollet, K. Junosza-Szaniawski, S. Abdellatif, P. Berthou, S. Pedebearn, and A. Dudko. "Critical Function Placement based on Service Chains in Multi-administrative Federated Networks". *Journal of Communications Software and Systems* 21.1 (2025). doi: 10.24138/jcomss-2024-0117.

[86] D. Nogalski, D. Belabed, A. Triollet, K. Junosza-Szaniawski, S. Abdellatif, P. Berthou, S. Pedebearn, and A. Dudko. "Federated SFC Placement in Sliced Collaborative Multi-Administrative Multi-Domain Networks". *2024 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. doi: 10.23919/SoftCOM62040.2024.10721844. 2024, pp. 1–6.

[87] D. Nogalski, J. Śliwa, and D. Duda. "Standard STANAG 5066 jako element warstwy transportowej dynamicznej architektury systemów taktycznych - TACTICS". *Przegląd Telekomunikacyjny Wiadomości Telekomunikacyjne* 8-9 (2017). doi: 10.15199/59.2017.8-9.19.

[88] M. Obadia, J. Rougier, L. Iannone, V. Conan, and M. Bouet. "Revisiting NFV orchestration with routing games". *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, November 7-10, 2016*. doi: 10.1109/NFV-SDN.2016.7919484. IEEE, 2016, pp. 107–113.

[89] J. Omer and A. Mucherino. "Referenced Vertex Ordering Problem: Theory, Applications and Solution Methods". working paper or preprint. 2020.

[90] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. "SNDlib 1.0 - Survivable Network Design Library". *Networks* 55.3 (2010). doi: 10.1002/net.20371, pp. 276–286.

[91] M. Patan. "Distributed scheduling of sensor networks for identification of spatio-temporal processes". *Int. J. Appl. Math. Comput. Sci.* 22.2 (2012). doi: 10.2478/v10006-012-0022-9, pp. 299–311.

[92] S. Pedebearn, S. Abdellatif, P. Berthou, D. Nogalski, and D. Belabed. "Controllable virtual network service over a multi-administrative multi-domain network". *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*. doi: 10.1109/ISORC61049.2024.10551360. 2024, pp. 1–10.

[93] S. Pedebearn, S. Abdellatif, P. Berthou, D. Nogalski, and D. Belabed. "Virtual Link Embedding in Collaborative Sliced Multi-Administrative Multi-Domain Networks". *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*. doi: 10.1145/3605098.3636067. Avila, Spain: Association for Computing Machinery, 2024, pp. 1088–1095.

[94] T. Peng, C. Leckie, and K. Ramamohanarao. "Survey of network-based defense mechanisms countering the DoS and DDoS problems". *ACM Comput. Surv.* 39.1 (2007). doi: 10.1145/1216370.1216373, p. 3.

[95] Y. Peng and B. Di. "Joint VNF Deployment and Resource Allocation in Integrated Terrestrial-Aerial Access Networks Enabled by Network Slicing". *20th IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2022, Wuhan, China, December 9-11, 2022*. doi: 10.1109/EUC57774.2022.00021. IEEE, 2022, pp. 74–80.

[96] N. Perrot and T. Reynaud. "Optimal placement of controllers in a resilient SDN architecture". *12th International Conference on the Design of Reliable Communication Networks, DRCN 2016, Paris, France, March 15-17, 2016*. Ed. by P. Chemouil, A. Pattavina, É. Gourdin, and S. Secci. doi: 10.1109/DRCN.2016.7470849. IEEE, 2016, pp. 145–151.

[97] M. E. Pfetsch and C. Liebchen. "Multicommodity Flows and Column Generation". *Lecture Notes*. Zuse Institute Berlin. 2006.

[98] M. Pilipczuk and M. Wahlström. "Directed Multicut is W[1]-hard, Even for Four Terminal Pairs". *ACM Trans. Comput. Theory* 10.3 (2018). doi: 10.1145/3201775, 13:1–13:18.

[99] M. Pióro, M. Mycek, and A. Tomaszewski. "Network Protection Against Node Attacks Based on Probabilistic Availability Measures". *IEEE Trans. Netw. Serv. Manag.* 18.3 (2021). doi: 10.1109/TNSM.2021.3067775, pp. 2742–2763.

[100] M. Pióro, M. Mycek, and A. Tomaszewski. "Using Probabilistic Availability Measures for Predicting Targeted Attacks on Network Nodes". *4th Cyber Security in Networking Conference, CSNet 2020, Lausanne, Switzerland, October 21-23, 2020*. Ed. by R. Laborde, N. Aitsaadi, S. Ghernaouti, A. Benzekri, and J. García-Alfaro. doi: 10.1109/CSNet50428.2020.9265459. IEEE, 2020, pp. 1–8.

[101] M. Pióro, M. Mycek, A. Tomaszewski, K. Junosza-Szaniawski, and D. Nogalski. "Finding Optimal Mixed Strategies in a Matrix Game Between the Attacker and the Network Operator". *13th International Workshop on Resilient Networks Design and Modeling, RNDM 2023, Hamburg, Germany, September 20-22, 2023*. doi: 10.1109/RNDM59149.2023.10293051. IEEE, 2023, pp. 1–8.

[102] M. Pióro, M. Mycek, A. Tomaszewski, and A. de Sousa. "Maximizing SDN resilience to node-targeted attacks through joint optimization of the primary and backup controllers placements". *Networks* 83.2 (2024). doi: 10.1002/NET.22201, pp. 428–467.

[103] L. Popokh, J. Su, S. Nair, and E. V. Olinick. "IllumiCore: Optimization Modeling and Implementation for Efficient VNF Placement". *International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2021, Split, Hvar, Croatia, September 23-25, 2021*. IEEE, 2021, pp. 1–7.

[104] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang. "SENSS Against Volumetric DDoS Attacks". *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. doi: 10.1145/3274694.3274717. 2018, pp. 266–277.

[105] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly. "DDoS-shield: DDoS-resilient scheduling to counter application layer attacks". *IEEE/ACM Transactions on Networking* 17.1 (2009). doi: 10.1109/TNET.2008.926503, pp. 26–39.

[106] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha. "Optimal controller placement in Software Defined Networks (SDN) using a non-zero-sum game". *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia*

Networks, WoWMoM 2014, Sydney, Australia, June 19, 2014. doi: 10.1109/WoWMoM.2014.6918987. IEEE Computer Society, 2014, pp. 1–6.

[107] D. F. Rueda, E. Calle, and J. L. Marzo. "Improving the Robustness to Targeted Attacks in Software Defined Networks (SDN)". *DRCN 2017 - Design of Reliable Communication Networks; 13th International Conference*. 2017, pp. 1–8.

[108] D. Santos, T. Gomes, and D. Tipper. "SDN Controller Placement With Availability Upgrade Under Delay and Geodiversity Constraints". *IEEE Transactions on Network and Service Management* 18.1 (2021). doi: 10.1109/TNSM.2020.3049013, pp. 301–314.

[109] D. Santos, A. de Sousa, and C. M. Machuca. "Robust SDN controller placement to malicious node attacks". *21st Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN 2018, Paris, France, February 19-22, 2018*. doi: 10.1109/ICIN.2018.8401617. IEEE, 2018, pp. 1–8.

[110] D. Santos, A. de Sousa, and P. P. Monteiro. "Compact Models for Critical Node Detection in Telecommunication Networks". *Electron. Notes Discret. Math.* 64 (2018). doi: 10.1016/j.endm.2018.02.007, pp. 325–334.

[111] S. Scott-Hayward. "Design and deployment of secure, robust, and resilient SDN controllers". *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. doi: 10.1109/NETSOFT.2015.7258233. 2015, pp. 1–5.

[112] A. Studer and A. Perrig. "The Coremelt Attack". *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*. Ed. by M. Backes and P. Ning. Vol. 5789. Lecture Notes in Computer Science. doi: 10.1007/978-3-642-04444-1_3. Springer, 2009, pp. 37–52.

[113] M. Suchanski, P. Kaniewski, J. Romanik, E. Golan, and K. Zubel. "Radio environment maps for military cognitive networks: density of small-scale sensor network vs. map quality". *EURASIP J. Wirel. Commun. Netw.* 2020.1 (2020). doi: 10.1186/s13638-020-01803-4, p. 189.

[114] J. Śliwa, D. Nogalski, K. Gleba, D. Duda, A. Flizikowski, and K. Samp. "TACTICS - architektura usługowa dla systemów taktycznych – wyniki projektu". *Przegląd Telekomunikacyjny Wiadomości Telekomunikacyjne* 8-9 (2017). 10.15199/59.2017 .8-9.19.

[115] N. Tastevin, M. Obadia, and M. Bouet. "A graph approach to placement of Service Functions Chains". *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, May 8-12, 2017*. doi: 10.23919/INM.2017 .7987273. IEEE, 2017, pp. 134–141.

[116] E. Tohidi, S. Parsaeefard, M. A. Maddah-Ali, B. H. Khalaj, and A. Leon-Garcia. "Near-Optimal Robust Virtual Controller Placement in 5G Software Defined Networks". *IEEE Transactions on Network Science and Engineering* 8.2 (2021). doi: 10.1109/TNSE.2021.3068975, pp. 1687–1697.

[117] A. Tomaszewski, M. Pióro, and M. Mycek. "Max-Min Optimization of Controller Placements vs. Min-Max Optimization of Attacks on Nodes in Service Networks". *Proceedings of the 10th International Network Optimization Conference, INOC 2022, Aachen, Germany, June 7-10, 2022*. Ed. by C. Büsing and A. M. C. A. Koster. doi: 10.48786/inoc.2022.13. OpenProceedings.org, 2022, pp. 1–6.

[118] S. Toufga, S. Abdellatif, H. T. Assouane, P. Owezarski, and T. Villemur. "Towards Dynamic Controller Placement in Software Defined Vehicular Networks". *Sensors* 20.6 (2020). doi: 10.3390/s20061701, p. 1701.

[119] N. Toumi, O. Bernier, D. Meddour, and A. Ksentini. "On Using Physical Programming for Multi-Domain SFC Placement With Limited Visibility". *IEEE Trans. Cloud Comput.* 10.4 (2022). doi: 10.1109/TCC.2020.3046997, pp. 2787–2803.

[120] D. Ucinski. "Sensor network scheduling for identification of spatially distributed processes". *Int. J. Appl. Math. Comput. Sci.* 22.1 (2012). doi: 10.2478/v10006-012-0002-0, pp. 25–40.

[121] G. Vasileios, S. D. Wolthusen, A. Flizikowski, A. Stachowicz, D. Nogalski, K. Gleba, and J. Sliwa. "Interoperability of security and quality of Service Policies Over Tactical SOA". *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. doi: 10.1109/SSCI.2016.7850077. 2016, pp. 1–7.

[122] P. Vizarreta, C. M. Machuca, and W. Kellerer. "Controller placement strategies for a resilient SDN control plane". *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), Halmstad, Sweden, September 13-15, 2016*. doi: 10.1109/RNDM.2016.7608295. IEEE, 2016, pp. 253–259.

[123] G. Wang, Y. Zhao, J. Huang, and Y. Wu. "An Effective Approach to Controller Placement in Software Defined Wide Area Networks". *IEEE Transactions on Network and Service Management* 15.1 (2018). 10.1109/TNSM.2017.2785660, pp. 344–355.

[124] K. Wang, M. Du, S. Maharjan, and Y. Sun. "Strategic Honeypot Game Model for Distributed Denial of Service Attacks in the Smart Grid". *IEEE Transactions on Smart Grid* 8.5 (2017). doi: 10.1109/TSG.2017.2670144, pp. 2474–2482.

[125] M. Wang, C. W. Tan, W. Xu, and A. Tang. "Cost of Not Splitting in Routing: Characterization and Estimation". *IEEE/ACM Transactions on Networking* 19.6 (2011). doi: 10.1109/TNET.2011.2150761, pp. 1849–1859.

[126] T. Wang and H. Chen. "Optimal Model for Link Failure Foresight Controller Placement in SDN". *2021 IEEE 4th International Conference on Electronics Technology (ICET)*. doi: 10.1109/ICET51757.2021.9450905. 2021, pp. 727–730.

[127] A. Wion, M. Bouet, L. Iannone, and V. Conan. "Change in Continuity: Chaining Services With an Augmented IGP". *IEEE Trans. Netw. Serv. Manag.* 16.4 (2019). doi: 10.1109/TNSM.2019.2944011, pp. 1332–1344.

[128] R. Wood. "Deterministic network interdiction". *Mathematical and Computer Modelling* 17.2 (1993). doi: 10.1016/0895-7177(93)90236-R, pp. 1–18.

[129] Y. Wu and J. Zhou. "Dynamic Service Function Chaining Orchestration in a Multi-Domain: A Heuristic Approach Based on SRv6". *Sensors* 21.19 (2021). doi: 10.3390/s21196563, p. 6563.

[130] Y. Yang, B. Wang, J. Tian, and P. Luo. "Efficient SFC Protection Method against Network Attack Risks in Air Traffic Information Networks". *Electronics* 13.13 (2024). doi: 10.3390/electronics13132664.

[131] G. Yao, J. Bi, Y. Li, and L. Guo. "On the Capacitated Controller Placement Problem in Software Defined Networks". *IEEE Commun. Lett.* 18.8 (2014). doi: 10.1109/LCOMM.2014.2332341, pp. 1339–1342.

[132] X.-D. Zang, J. Gong, and X.-Y. Hu. "An Adaptive Profile-Based Approach for Detecting Anomalous Traffic in Backbone". *IEEE Access* 7 (2019). doi: 10.1109/ACCESS.2019.2914303, pp. 56920–56934.

[133] S. T. Zargar, J. Joshi, and D. Tipper. "A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks". *IEEE Communications Surveys and Tutorials* 15.4 (2013). doi: 10.1109/SURV.2013.031413.00127, pp. 2046–2069.

[134] M. Zekri, S. E. Kafhali, N. Aboutabit, and Y. Saadi. "DDoS attack detection using machine learning techniques in cloud computing environments". *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech).* doi: 10.1109/CloudTech.2017.8284731. 2017, pp. 1–7.

[135] Y. Zhang, N. Beheshti, and M. Tatipamula. "On Resilience of Split-Architecture Networks". *Proceedings of the Global Communications Conference, GLOBECOM 2011, 5-9 December 2011, Houston, Texas, USA*. 10.1109/GLOCOM.2011.6134496. IEEE, 2011, pp. 1–6.

[136] Q. Zhu, X. Yu, Y. Zhao, A. Nag, and J. Zhang. "Auxiliary-Graph-Based Energy-Efficient Traffic Grooming in IP-Over-Fixed/Flex-Grid Optical Networks". *Journal of Lightwave Technology* 39.10 (2021). doi: 10.1109/JLT.2021.3057389, pp. 3011–3024.

List of Abbreviations

AI	Artificial Intelligence
BFS	Breadth First Search
CDN	Content Delivery Networks
CIA	Confidentiality Integrity Availability
CIAC	Computer Incident Advisory Capability
CMFNIP	Cardinality Maximum Flow Network Interdiction Problem
CND	Critical Node Detection
CN	Control Node
CPP	Control Placement Problem
CPU	Central Processing Unit
DC	Data Center
DDoS	Distributed Denial of Service
DL	Deep Learning
DNS	Domain Name System
DoS	Denial of Service
DPI	Deep Packet Inspection
E2E	End-to-End
FW	Firewall
FPT	Fixed Parameter Tractable

IDS	Intrusion Detection System
ILP	Integer Linear Program
IPS	Intrusion Prevention System
IP	Internet Protocol
ISP	Internet Service Provider
L2	Layer 2
L3	Layer 3
LP	Linear Programming
LSTM	Long Short-Term Memory
MEC	Multi-Access Edge Computing
MFNIP	Maximum Flow Network Interdiction Problem
MFP	Multicommodity Flow Problem
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
ML	Machine Learning
MM	Min Max
NFV	Network Function Virtualization
NP	Nondeterministic Polynomial Time
NUM	Network Utility Maximization
OPEX	Operational Expenses
OPT	Optimum
OVS	Open vSwitch
P	Polynomial Time
P2P	Point-to-Point
PC	Placement with required Cardinality

PoP	Point of Presence
PQ	Placement with required Quality
RAN	Radio Access Network
SAT	Boolean Satisfiability Problem
SD	Slice Deployment
SD-WAN	Software-Defined Wide Area Networks
SDN	Software Defined Network
SFC	Service Function Chaining
SN	Substrate Network
SVM	Support Vector Machine
TE	Traffic Engineering
QoS	Quality of Service
UN	User Network
VM	Virtual Machine
VNE	Virtual Network Embedding
VNF	Virtual Network Function
VNF-FG	Virtual Network Function Forwarding Graph
VoIP	Voice over Internet Protocol
VPRN	Virtual Private Routed Networks
WSN	Wireless Sensor Network
XP	Slicewise Polynomial

List of Figures

List of Tables