Int. J. Appl. Math. Comput. Sci., 2022, Vol. 32, No. 1, 35–49
DOI: 10.34768/amcs-2022-0004

EXACT AND APPROXIMATION ALGORITHMS FOR SENSOR PLACEMENT AGAINST DDoS ATTACKS

Konstanty JUNOSZA-SZANIAWSKI [a,*], Dariusz NOGALSKI [b], Paweł RZĄŻEWSKI [a,c]

[a]Faculty of Mathematics and Information Science
Warsaw University of Technology
ul. Koszykowa 75, 00-662 Warsaw, Poland
e-mail: {konstanty.szaniawski,pawel.rzazewski}@pw.edu.pl

[b]Military Communication Institute
ul. Warszawska 22A, 05-130 Zegrze, Poland
e-mail: d.nogalski@wil.waw.pl

[c]Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland

In a distributed denial of service (DDoS) attack, the attacker gains control of many network users through a virus. Then the controlled users send many requests to a victim, leading to its resources being depleted. DDoS attacks are hard to defend because of their distributed nature, large scale and various attack techniques. One possible mode of defense is to place sensors in a network that can detect and stop an unwanted request. However, such sensors are expensive, as a result of which there is a natural question as to the minimum number of sensors and their optimal placement required to get the necessary level of safety. Presented below are two mixed integer models for optimal sensor placement against DDoS attacks. Both models lead to a trade-off between the number of deployed sensors and the volume of uncontrolled flow. Since the above placement problems are NP-hard, two efficient heuristics are designed, implemented and compared experimentally with exact mixed integer linear programming solvers.

Keywords: DDoS, sensor placement, network safety optimization, heuristics.

1. Introduction

1.1. Distributed denial of service. Denial-of-service (DoS) attacks are intended to stop legitimate users from accessing a specific network resource (Zargar *et al.*, 2013). A DoS attack is an attack on availability, which is one of the three dimensions from 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 (Criscuolo, 2000). In a DDoS attack, the attacker gains the 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 and out of resources, unable to provide services to legitimate users.

DDoS attacks are a problem not only on the Internet (Ramanathan *et al.*, 2018), but also in the context of a smart grid (Wang *et al.*, 2017; Cameron *et al.*, 2019; Huseinović *et al.*, 2020), cloud (Bonguet and Bellaïche, 2017) and control systems (Cetinkaya *et al.*, 2019). According to Cameron *et al.* (2019), 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

*Corresponding author

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, 2004; Ranjan *et al*., 2009). 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*., 2018). After a certain time, the botnet commander switches to another subset of nodes that conduct the attack.

As pointed out by Zargar *et al*. (2013), 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, 2009) and *Crossfire attacks* (Kang *et al*., 2013). Both of these attacks aim at intermediate network links located between attack sources and targets. Traditional target-based defenses do not work with these types of attacks (Liaskos and Ioannidis, 2018; Gkounis *et al*., 2016).

(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 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*., 2018)—an attacker sends a request with a fake address (of a victim) to the DNS server, and the server responds to the victim; *spoofed attacks* (Armbruster *et al*., 2007)—an attacker forges the true origin of packets. Detailed classifications of DDoS attacks are discussed by, e.g., Mirkovic and Reiher (2004), Douligeris and Mitrokotsa (2004), Peng *et al*. (2007), Zargar *et al*. (2013), Bonguet and Bellaïche (2017), and Huseinović *et al*. (2020).

A detection algorithm of DDoS attacks and the identification of an attack signature is out of the scope of this research. In the literature one can find various works in this field. Many works use machine learning or other artificial intelligence techniques, e.g., de Miranda Rios *et al*. (2021) use a multi-layer perceptron (MLP) neural network with backpropagation, K-nearest neighbors (K-NN), a support vector machine (SVM) and a multinomial naive Bayes classifier (MNB); Daya *et al*. (2020) incorporate graph-based features into machine learning. Other works focus on general methods of anomaly detection, including signature-based and profile-based methods, e.g., Huang *et al*. (2021) propose a multi-channel network traffic anomaly detection method combined with multi-scale decomposition; Hwang *et al*. (2020) present an anomaly traffic detection mechanism, which consists of a convolutional neural network (CNN) and an unsupervised deep learning model; Zang *et al*. (2019) use the ant colony optimization (ACO) to construct the baseline profile of the normal traffic behavior. Other related results are reported by, e.g., Liu *et al*. (2021), Gera and Battula (2018), Jiao *et al*. (2017), Zekri *et al*. (2017), de Assis *et al*. (2017), Kallitsis *et al*. (2016), and Afek *et al*. (2013). Comprehensive surveys of DDoS detection are also available: Jafarian *et al*. (2021) overview anomaly detection mechanisms in software defined networks; Khalaf *et al*. (2019) focus on the defense methods that adopt artificial intelligence and statistical approaches.

1.2. Sensor placement. One of the ways to defend against a DDoS attack is to place sensors 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 system)/firewall solutions that detect and eliminate DDoS attacks have a high acquisition price (Fayaz *et al*., 2015; Blazek *et al*., 2019). 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 (also known as distributed) defense mechanisms is discussed by Zargar *et al*. (2013).

El Defrawy *et al*. (2007) 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*. (2007) analyze 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 the forged origin) can reach its destination. They relate the problem to the vertex cover one 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 a specific version of DDoS—a *spoofed attack*.

Jeong *et al.* (2004) and Islam *et al.* (2008) 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.* (2015) propose a Bohatei system for DDoS defense within a single Internet service provider (ISP). 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 virtual machines (VMs). These VMs detect and block attack traffic. Having fixed VMs, the system routes the traffic through these VMs. The goal of the resource manager is to efficiently assign available network resources to the defense, (i) minimizing the latency experienced by legitimate traffic, and (ii) 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 a 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 does not 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.* (2018) assume an 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 are 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.* (2018) propose a collaboration system (SENSS) to protect against DDoS. The 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.* (2017) place control nodes (CNs) in a clustered wireless sensor network (WSN). The 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 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 (2010) discusses 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. Uciński (2012) focuses on the design of a monitoring sensor network to provide proper diagnostic information about the functioning of a distributed parameter system. Patan (2012) 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.* (2020) study the dependency between density of a sensor network and map quality in the radio environment map (REM) concept. There have been a large number of works on developing methods and technology of human 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.* (2019) 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 a well-known class of interdiction problems, which can be related to our DDoS problem. Altner *et al.* (2010) 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 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, 1993). 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*., 2014). A more general problem on graphs was proposed by Omer and Mucherino (2020), and it includes the interdiction problem. In our DDoS problem we delete vertices instead of arcs in the CMFNIP.

1.3. Discussion. Defense mechanisms against DDoS flooding attacks address specific attack types: *link-flooding* (Studer and Perrig, 2009; Kang *et al*., 2013) or *target-flooding* (Zargar *et al*., 2013). *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*., 2018), *spoofing* (Armbruster *et al*., 2007) or other techniques (Zargar *et al*., 2013). 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*., 2018) (out of the scope of this paper). Some defense approaches use attack load distribution (re-routing of traffic) to limit the effect on targets (Belabed *et al*., 2018).

In this paper, the focus is on the placing of detection sensors. There are several works in this field: Jeong *et al*. (2004) and Islam *et al*. (2008) minimize the number of sensors such that every path of a given length (r) contains a sensor; Armbruster *et al*. (2007) analyze the problem of packet filter placement to defend a network against spoofed denial of service attacks; Monnet *et al*. (2017) place control nodes in clustered WSNs to save the energy of nodes; Fayaz *et al*. (2015) 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.

1.4. Our proposal. A DDoS attack can be modeled as a flow from multiple sources to a single target (single commodity flow). Defined are a 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 $t \in T$ the max-flow algorithm (see, e.g., Goldberg and Tarjan, 2014) 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, the maximum uncontrolled flow as $\max_{t \in T} \mathsf{maxflow}_G(ss, t)$ is computed.

In Section 2.2 a proof is given of the decision problem as to whether d sensors suffice to reduce the uncontrolled flow to some defined amount $a \in \mathbb{R}$. When there is just one protected node, the proof is based on reduction from the *cardinality maximum flow network interdiction* Problem (CMFNIP) (Wood, 1993). When the number of pairs (S, t_i) is more than one, the reduction goes from multiway cut (cf. Garg *et al*., 1994).

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 of how much uncontrolled flow we can reduce with such a number of sensors is asked.

The main result of this paper, 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.

Preliminary work on sensor placement was published as a conference paper (Junosza-Szaniawski *et al*., 2020).

2. Problem definition

2.1. Problem of optimal sensor placement.

The 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 (also called 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 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 denoted by D. For the 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 Figs. 2 and 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

$$\forall_{u \in V \setminus (S \cup \{t\})} \sum_{(v,u) \in E} f_t(v,u) = \sum_{(u,w) \in E} f_t(u,w) \quad (1)$$

and

$$\forall_{e \in E}\ f_t(e) \le c(e). \quad (2)$$

The attack flow value is given by

$$f_t = \sum_{(v,t) \in E} f_t(v,t) - \sum_{(t,w) \in E} f_t(t,w). \quad (3)$$

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

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

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

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 a positive value.

For example, in Fig. 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, as a result of which they 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 $D \subseteq V \setminus (S \cup T)$ satisfying

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

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

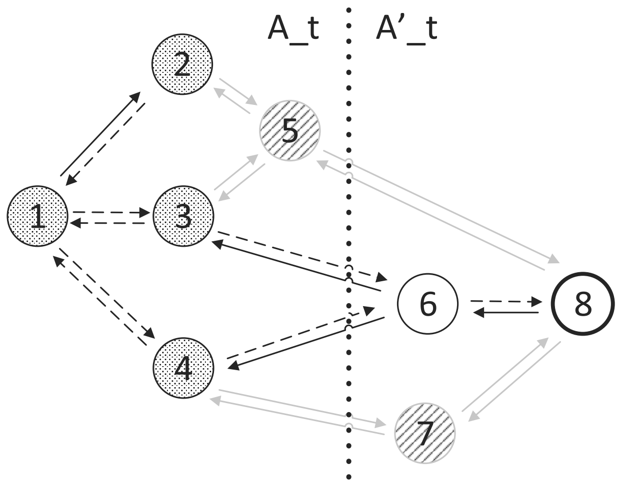


Fig. 1. Instance G with source (attack) nodes $S = \{1, 2, 3, 4\}$, protected nodes $T = \{8\}$ and sensors $D = \{5, 7\}$. The dotted vertical line denotes a possible *cut* for $t = 8 \in T$. The dashed lines denote the *uncontrolled flow* f_8.

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.

2.2. Complexity of 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 $t \in T$ we have $\mathsf{maxflow}_{G \setminus D}(S, t) \leq a$?

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

First, there are simple boundary cases. If $a = 0$, then the problem asks for an S-T-separator of a 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 when $|T| = 1$. This will yield a reduction from CMFNIP, which is known to be NP-hard (Wood, 1993). 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 $v_1, v_2, \ldots, v_{k+1}$. 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 when $|T| \geq 2$. This time we will reduce from Node Multiway Cut with 2 terminals, which is known to be NP-hard (Garg *et al.*, 1994). 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, we have*

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

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 $t \in T$ we have 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 a size at most k, such that for every $t \in T$ we have $\mathsf{maxflow}_{G \setminus D}(S, t) \leq a$?*

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 Cygan *et al.* (2015) 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 a size at most k, such that for every $t \in T$ we have $\mathsf{maxflow}_{G \setminus D}(S_t, t) \le a$. 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. An 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 (2018), 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.*, 2015).

3. Description of models

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 the well-known Ford–Fulkerson theorem (1956) 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 $v \notin A_t$ (Fig. 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 the *PQ* model, a function to minimize is $\sum_{v \in V} d[v]$ with respect to the restrictions

$$\forall_{t \in T} \ \forall_{s \in S} \ \ a[t, s] == 1, \tag{4}$$

$$\forall_{t \in T} \ \ a[t, t] == 0, \tag{5}$$

$$\forall_{t \in T} \ \forall_{(u,v) \in E} \quad cutT[t, u, v] \ge a[t, u] - a[t, v] - d[u] - d[v], \tag{6}$$

$$\forall t \in T \ \sum_{(u,v) \in E} cutT[t, u, v] \cdot c[u, v] \le a, \tag{7}$$

$$\forall s \in S \ \ d[s] = 0, \tag{8}$$

$$\forall t \in T \ \ d[t] = 0. \tag{9}$$

The meaning is as follows. For every target $t \in T$ each vertex $s \in S$ belongs to A_t, cf. (4). For every target $t \in T$ the vertex t does not belong to A_t, cf. (5). The restriction (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. Equation (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 (a parameter in the problem formulation), $q = 1$ signifies total control (100% traffic controlled), $q = 0$ signifies no control (zero sensors placed); furthermore, $\max_{t \in T} \mathsf{maxflow}_G(t)$ is equal to the value of max minimum cut M_t in G. The restrictions (8) and (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).

In the *PC* model, a function to minimize is just M with respect to the restrictions (4)–(6), (8) and (9),

$$\sum_{v \in V} d[v] = k, \tag{10}$$

$$\forall t \in T \quad \sum_{(u,v) \in E} cutT[t, u, v] \cdot c[u, v] \le M. \tag{11}$$

The restriction (10) makes sure that the number of sensors is fixed, and given as parameter k to the problem. Equation (11) bounds the value of the cut with M.

As shown in Section 5, the above models are very efficient in terms of the number of deployed sensors and the 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 4); they are reasonably efficient in terms of a goal value, but much faster than the models.

4. Algorithm description

Relaxed formulation of PQ and PC models. In this formulation we relax two types of variables to allow the fractional sensor placement (the first bullet point) and fractional traffic control (the second bullet point). 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. The relaxations are as follows:

- 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) 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), we have designed and implemented two algorithms:

1. *PQIterativeBestSensor* (see Algorithm 1)
2. *PCIterativeBestSensor* (see Algorithm 2).

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

4.1. 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, a 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, a 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.

4.2. PC iterative best sensor placement. The algorithm *PCIterativeBestSensor* consists of $k + 1$ iterations. In each $\{1, \ldots, k\}$ iteration, a 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, a 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.

Algorithm 1. PQIterativeBestSensor.

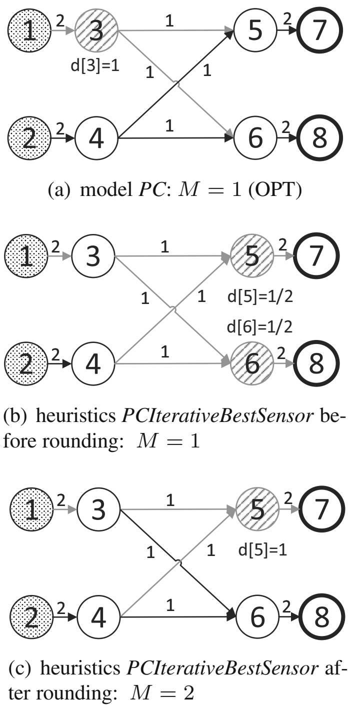
Require: G, c, T, S, q
1: Evaluate $a = (1 - q) \cdot \max_{t \in T} \mathsf{maxflow}_G(t)$
2: Form the relaxed *PQ* $problem$ (Section 4) with goal $minimize \sum_{v \in V} d[v]$. Add constraints {(4)–(9)} to the $problem$.
3: Initiate a set of vertices in which we place sensors $D = \emptyset$.
4: **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 $L = \{v$, s.t. $v \in V \setminus D$ and $d[v] \neq 0$ and $d[v] == \max\{d[j]\}_{j \in V \setminus D}\}$.
7: Choose randomly $v_{\max} \in L$, where the probability of selecting an element $v_{\max}$ equals $1/|L|$.
8: Add constraint $d[v_{\max}] == 1$ to the $problem$
9: $D = D \cup \{v_{\max}\}$.
10: **end while**
11: **return** D

Algorithm 2. PCIterativeBestSensor.

Require: G, c, T, S, k
1: From the relaxed *PC* $problem$ (Section 4) with goal $minimize\ M$. Add constraints {(4),(5),(6),(8),(9),(10),(11)} to the $problem$.
2: Initiate a set of vertices in which we place sensors $D = \emptyset$.
3: **for** $i = 1, \ldots, k$ **do**
4: Solve the $problem$.
5: Let $L = \{v$: $v \in V \setminus D$ and $d[v] \neq 0$ and $d[v] == \max\{d[j]\}_{j \in V \setminus D}\}$.
6: Choose randomly $v_{\max} \in L$, where the probability of selecting an element $v_{\max}$ equals $1/|L|$.
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)

We show that the algorithm *PCIterativeBestSensor* may give a result $2 \cdot OPT$. In Fig. 2 we compare the optimal solution *OPT* given by *PC* model (a) to the solution given by the *PCIterativeBestSensor* (b),(c). We assume two sources $S = \{1, 2\}$ and two targets $T = \{7, 8\}$, and we require to place $k = 1$ sensors. The optimal solution is $M = 1$ (a). Then one fractional solution given by the heuristics with its corresponding rounding is given. The results (b) and (c) 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 Fig. 3.

However, for practical scenarios the heuristics

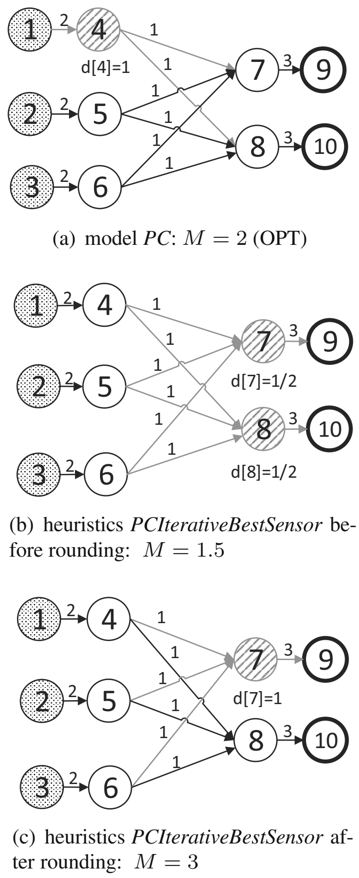


(a) model *PC*: $M = 1$ (OPT)

(b) heuristics *PCIterativeBestSensor* before rounding: $M = 1$

(c) heuristics *PCIterativeBestSensor* after rounding: $M = 2$

Fig. 2. Algorithm *PCIterativeBestSensor* yields $2 \cdot OPT$ (solutions (b) and (c)), where M is the value of uncontrolled flow, $S = \{1, 2\}$, $T = \{7, 8\}$, $k = 1$, and D is defined by gray striped circles.



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

(b) heuristics *PCIterativeBestSensor* before rounding: $M = 1.5$

(c) heuristics *PCIterativeBestSensor* after rounding: $M = 3$

Fig. 3. Algorithm *PCIterativeBestSensor* yields $\frac{3}{2} \cdot OPT$ (solutions (b) and (c)), where M is the value of uncontrolled flow, $S = \{1, 2, 3\}$, $T = \{9, 10\}$, $k = 1$, and D is defined by gray striped circles.

exposes a solid ratio (see Section 5).

5. Computational results

5.1. Experiment setup. The following experiments compare the efficiencies of the models with the algorithms. The *PQ* model is compared with the *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 processing 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: $Net|V|$, 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 Fig. 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 ($T_{i=1,\dots,4} \subseteq V$) were generated (all vertices V have equal probabilities). For each target instance T_i, four random instances of source locations were generated ($S_{j=1,\dots,4} \subseteq V \setminus T_i$) (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 the arithmetic mean computed from 64 measurements. Finally, we assumed the following number of targets and sources: Scenarios 1–4: $|T| = 10$,

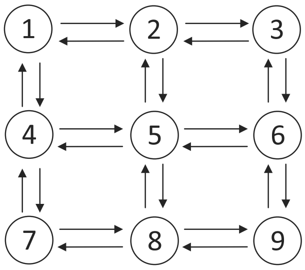


Fig. 4. Example of a small grid network, $|V| = 9$.

$|S| = 40$; Scenarios 1b and 2b: $|T| = 10$; Scenarios 3b and 4b: $|T| = 20$.

5.2. Scenario 1: A *PC* problem, *Net100*, an increasing number of sensors. The experiments were conducted for the grid network *Net100*. The number of sensors was increasing from $k = 0$ to $k = 10$.

The diagram of Fig. 5(a) demonstrates the average volume of uncontrolled traffic (the 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 the *PC* model and the *PCIterativeBestSensor* heuristics. The observed average objective values of *PCIterativeBestSensor* are higher than those of *PC* by up to 8%.

The diagram of Fig. 5(b) demonstrates the average time of execution (the y axis). The observed average values of execution time of *PC* are up to 10 times higher than those of *PCIterativeBestSensor*.

5.3. Scenario 2: A *PC* problem, $k = 5$, an increasing size of the grid for *Net64, Net81, ..., Net169*. The experiments were conducted for grid networks: *Net64, Net81, Net100, Net121, Net144, Net169*. The number of sensors was fixed at $k = 5$.

The diagram of Fig. 5(c) demonstrates the average time of execution (the 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 heuristics.

5.4. Scenario 3: A *PQ* problem, *Net196*, an increasing value of the quality factor. The experiments were conducted for the grid network *Net196*. The value of quality factor was increasing of $q \in \{0.1, 0.2, \ldots, 1.0\}$.

The diagram of Fig. 5(d) demonstrates the average number of sensors (the y-axis) required to control the q-factor of the network traffic (the x-axis). As the value of q-factor increases, the number of required sensors increases on average, for both model *PQ* and *PQIterativeBestSensor* heuristics. 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 of Fig. 5(e) demonstrates the average time of execution (the y-axis). The observed average values of execution time of *PQ* are up to 5 times higher than those of *PQIterativeBestSensor*.

5.5. Scenario 4: A *PQ* problem, $q = 0.5$, an increasing size of the grid for *Net121, Net144, ..., Net256*. The experiments were conducted for grid networks *Net121, Net144, Net169, Net196, Net225, Net256*. The quality factor was fixed at $q = 0.5$.

The diagram of Fig. 5(f) demonstrates the average time of execution (the 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 heuristics.

5.6. Scenarios 1b–4b.

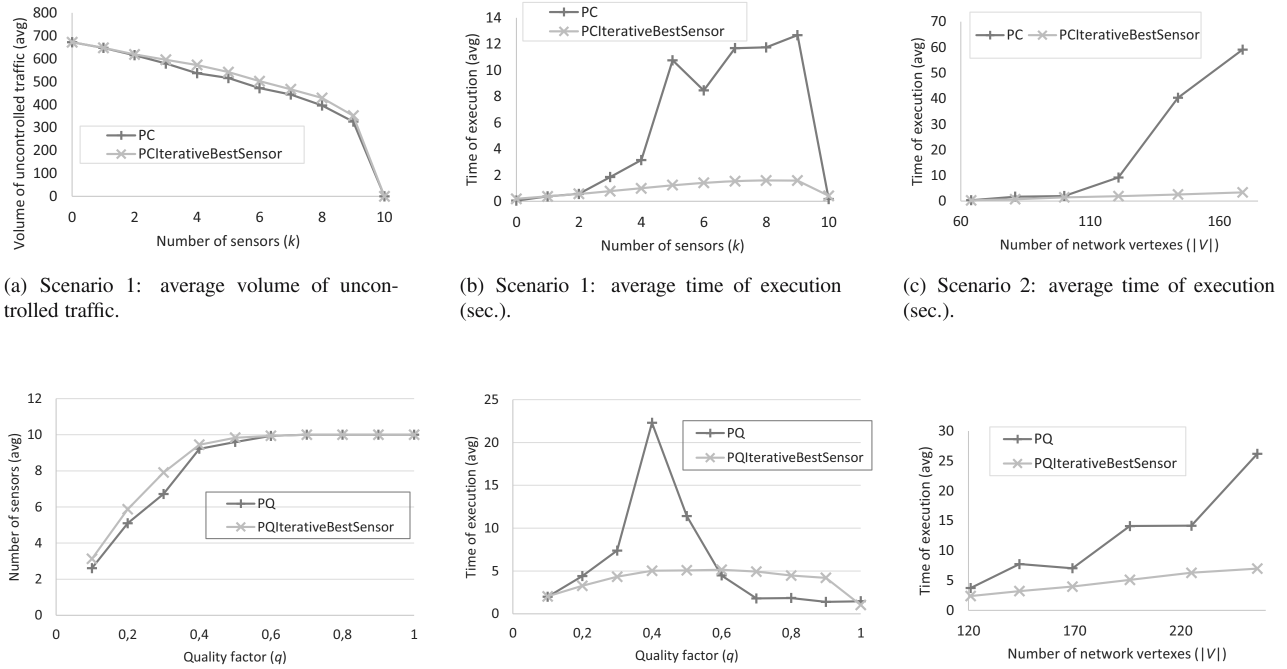
Super source formulation. In general, we would like to assume, that network flooding targeted at protected nodes $t \in T$ can start from any network node ($source$) $s \in V \setminus T$. In practical scenarios however, we may want to limit attention to a set of sources $S \subseteq V \setminus T$.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

First, explicit selection, as used in Experiments 1–4 (Sections 5.2, 5.3, 5.4 and 5.5). We selected subsets of vertices as sources $|S| = 40$. The sources were selected randomly with uniform distribution on 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 *single super source formulation* below for details). This method was applied within Scenarios $1b$–$4b$. Experiments $1b$–$4b$ were conducted with the following settings: Scenario1b: *Net100*, the number of sensors from $k = 0$ to $k = 10$; Scenario 2b: $k = 5$, the size of the grid *Net64, Net81,..., Net169*; Scenario 3b: *Net289*, the value of quality factor $q \in \{0.1, 0.2, \ldots, 1.0\}$; Scenario 4b: $q = 0.5$, the size of the grid *Net144, Net169, ..., Net256*.

The algorithms efficiency demonstrated in Scenarios $1b$–$4b$ (Fig. 6) is similar to that demonstrated in Scenarios 1–4 (Fig. 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



(a) Scenario 1: average volume of uncontrolled traffic.

(b) Scenario 1: average time of execution (sec.).

(c) Scenario 2: average time of execution (sec.).

(d) Scenario 3: average number of sensors.

(e) Scenario 3: average time of execution (sec.).

(f) Scenario 4: average time of execution (sec.).

Fig. 5. Results for Scenarios 1–4.

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

$$\forall_{v \in V \setminus T} \;\; c(ss, v) = R(v) \cdot \sum_{u:(v,u) \in E} c(v, u). \qquad (12)$$

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 the restriction

$$d[ss] = 0 \qquad (13)$$

to both *PQ* and *PC* models (models described in Section 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 (13) applies to both algorithms *PQIterativeBestSensor* and *PCIterativeBestSensor* (Section 4).

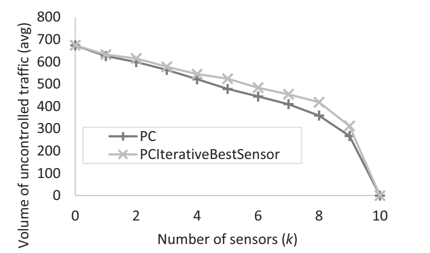
5.7. Summary of simulation results. The simulations for the *PC* algorithm led to a number of observations. Firstly, for all test networks, as the number of sensors increases, the volume of uncontrolled traffic decreases to zero, for both the *PC* model and the *PCIterativeBestSensor* heuristics. Secondly, the observed average objective values of the *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 heuristics.

The simulations of the *PQ* algorithm led to the following observations. Firstly, as the quality factor increases, the number of sensors increases on the average; however, at a certain point sensor usage becomes saturated, for both *PQ* model and *PQIterativeBestSensor* heuristics. Secondly, in the worst observed cases the *PQIterativeBestSensor* required approximately one more sensor 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 heuristics.

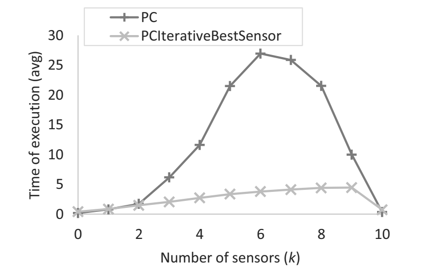
6. Conclusions

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, several natural questions arise: Is there a better exact algorithm than brute-force? Can the number of sensors be approximated with any constant?

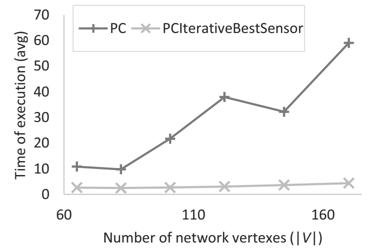
Although the problem is computationally hard, it can be efficiently solved with the use of a mixed integer programming solver for medium-sized networks. As



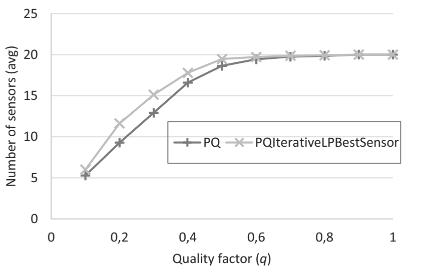
(a) Scenario 1 b: average volume of uncontrolled flow.



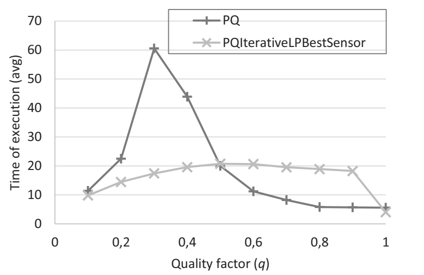
(b) Scenario 1 b: average time of execution (sec.).



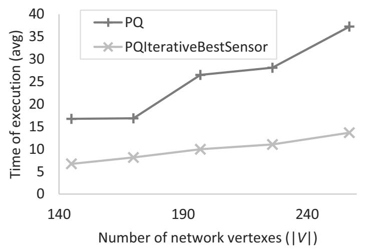
(c) Scenario 2 b: average time of execution (sec.).



(d) Scenario 3 b: average number of sensors.



(e) Scenario 3 b: average time of execution (sec.).



(f) Scenario 4 b: average time of execution (sec.).

Fig. 6. Results for Scenarios 1b–4b.

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., a risk factor). The models expose a highly desirable feature, that the deployment of a relatively small number of sensors (proportional to the number of protected nodes) can yield a significant quality. Both the 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.

Acknowledgment

The work of Dariusz Nogalski was partially supported within the statutory activity of the Military Communications Institute financed by the Ministry of Science and Higher Education (Poland). Paweł Rzążewski was supported through a project that received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement 714704).

References

Afek, Y., Bremler-Barr, A. and Landau Feibish, S. (2013). Automated signature extraction for high volume attacks, *Conference on Architectures for Networking and Communications Systems, San Jose, USA*, pp. 147–156.

Altner, D.S., Ergun, Ö. and Uhan, N.A. (2010). The maximum flow network interdiction problem: Valid inequalities, integrality gaps, and approximability, *Operations Research Letters* **38**(1): 33–38, DOI: 10.1016/j.orl.2009.09.013.

Armbruster, B., Smith, J.C. and Park, K. (2007). A packet filter placement problem with application to defense against denial of service attacks, *European Journal of Operational Research* **176**(2): 1283–1292.

de Assis, M.V.O., Hamamoto, A.H., Abrão, T. and Proença, M.L. (2017). A game theoretical based system using Holt-Winters and genetic algorithm with fuzzy logic for DoS/DDoS mitigation on SDN networks, *IEEE Access* **5**: 9485–9496, DOI: 10.1109/ACCESS.2017.2702341.

Belabed, D., Bouet, M. and Conan, V. (2018). Centralized defense using smart routing against link-flooding attacks, *2nd Cyber Security in Networking Conference, CSNet 2018, Paris, France*, pp. 1–8, DOI: 10.1109/CSNET.2018.8602966.

Blazek, P., Gerlich, T. and Martinasek, Z. (2019). Scalable DDoS mitigation system, *2019 42nd International Confer-*

ence on Telecommunications and Signal Processing (TSP), Budapest, Hungary, pp. 617–620.

Bonguet, A. and Bellaïche, M. (2017). A survey of denial-of-service and distributed denial of service attacks and defenses in cloud computing, *Future Internet* **9**(3), Article no. 43, DOI: 10.3390/fi9030043.

Cameron, C., Patsios, C., Taylor, P.C. and Pourmirza, Z. (2019). Using self-organizing architectures to mitigate the impacts of denial-of-service attacks on voltage control schemes, *IEEE Transactions on Smart Grid* **10**(3): 3010–3019.

Cetinkaya, A., Ishii, H. and Hayakawa, T. (2019). An overview on denial-of-service attacks in control systems: Attack models and security analyses, *Entropy* **21**(2): 210, DOI: 10.3390/e21020210.

Chou, J.-J., Shih, C.-S., Wang, W.-D. and Huang, K.-C. (2019). Iot sensing networks for gait velocity measurement, *International Journal of Applied Mathematics and Computer Science* **29**(2): 245–259, DOI: 10.2478/amcs-2019-0018.

Criscuolo, P.J. (2000). *Distributed Denial of Service: Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht*, Lawrence Livermore National Laboratory, Livermore.

Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M. and Saurabh, S. (2015). *Parameterized Algorithms*, Springer, Cham, DOI: 10.1007/978-3-319-21275-3.

Daya, A.A., Salahuddin, M.A., Limam, N. and Boutaba, R. (2020). BotChase: Graph-based bot detection using machine learning, *IEEE Transactions on Network and Servive Management* **17**(1): 15–29, DOI: 10.1109/TNSM.2020.2972405.

Douligeris, C. and Mitrokotsa, A. (2004). DDOS attacks and defense mechanisms: Classification and state-of-the-art, *Computer Networks* **44**(5): 643–666.

El Defrawy, K., Markopoulou, A. and Argyraki, K. (2007). Optimal allocation of filters against DDoS attacks, *2007 Information Theory and Applications Workshop, La Jolla, USA*, pp. 140–149.

Fayaz, S.K., Tobioka, Y., Sekar, V. and Bailey, M. (2015). Bohatei: Flexible and elastic DDOS defense, *24th USENIX Security Symposium, USENIX Security 15, Washington, USA*, pp. 817–832, `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/fayaz`.

Ford, L.R. and Fulkerson, D.R. (1956). Maximal flow through a network, *Canadian Journal of Mathematics* **8**: 399–404.

Garg, N., Vazirani, V.V. and Yannakakis, M. (1994). Multiway cuts in directed and node weighted graphs, *in* S. Abiteboul and E. Shamir (Eds), *Automata, Languages and Programming: 21st International Colloquium, ICALP94*, Springer, Berlin, pp. 487–498.

Gera, J. and Battula, B.P. (2018). Detection of spoofed and non-spoofed ddos attacks and discriminating them from flash crowds, *EURASIP Journal on Information Security* **2018**(1), Article no. 9, DOI: 10.1186/s13635-018-0079-6.

Gkounis, D., Kotronis, V., Liaskos, C. and Dimitropoulos, X.A. (2016). On the interplay of link-flooding attacks and traffic engineering, *Computer Communication Review* **46**(2): 5–11, DOI: 10.1145/2935634.2935636.

Goldberg, A.V. and Tarjan, R.E. (2014). Efficient maximum flow algorithms, *Communications of the ACM* **57**(8): 82–89, DOI: 10.1145/2628036.

Hemmati, M., Cole Smith, J. and Thai, M.T. (2014). A cutting-plane algorithm for solving a weighted influence interdiction problem, *Computational Optimization and Applications* **57**(1): 71–104, DOI: 10.1007/s10589-013-9589-9.

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

Huseinović, A., Mrdović, S., Bicakci, K. and Uludag, S. (2020). A survey of denial-of-service attacks and solutions in the smart grid, *IEEE Access* **8**: 177447–177470.

Hwang, R.-H., Peng, M.-C., Huang, C.-W., Lin, P.-C. and Nguyen, V.-L. (2020). An unsupervised deep learning model for early network traffic anomaly detection, *IEEE Access* **8**: 30387–30399.

Islam, M.H., Nadeem, K. and Khan, S.A. (2008). Efficient placement of sensors for detection against distributed denial of service attack, *2008 International Conference on Innovations in Information Technology, IIT 2008, Al Ain, UAE*, pp. 653–657.

Jafarian, T., Masdari, M., Ghaffari, A. and Majidzadeh, K. (2021). A survey and classification of the security anomaly detection mechanisms in software defined networks, *Cluster Computing* **24**(2): 1235–1253, DOI: 10.1007/s10586-020-03184-1.

Jeong, S.B., Choi, Y. and Kim, S. (2004). An effective placement of detection systems for distributed attack detection in large scale networks, *in* C.H. Lim and M. Yung (Eds), *Information Security Applications: 5th International Workshop, WISA 2004*, Springer, Berlin, pp. 204–210, DOI: 10.1007/978-3-540-31815-6_17.

Jiao, J., Ye, B., Zhao, Y., Stones, R.J., Wang, G., Liu, X., Wang, S. and Xie, G. (2017). Detecting TCP-based DDoS attacks in Baidu cloud computing data centers, *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, China*, pp. 256–258, DOI: 10.1109/SRDS.2017.37.

Junosza-Szaniawski, K., Nogalski, D. and Wójcik, A. (2020). 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*, pp. 295–301, DOI: 10.15439/2020F106.

Kallitsis, M.G., Stoev, S.A., Bhattacharya, S. and Michailidis, G. (2016). AMON: An open source architecture for online monitoring, statistical analysis, and forensics of multi-gigabit streams, *IEEE Journal on Selected Areas in Communications* **34**(6): 1834–1848, DOI: 10.1109/JSAC.2016.2558958.

Kang, M.S., Lee, S.B. and Gligor, V.D. (2013). The Crossfire attack, *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, USA*, pp. 127–141, DOI: 10.1109/SP.2013.19.

Khalaf, B.A., Mostafa, S.A., Mustapha, A., Mohammed, M.A. and Abduallah, W.M. (2019). Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods, *IEEE Access* **7** : 51691–51713.

Khapalov, A. (2010). Source localization and sensor placement in environmental monitoring, *International Journal of Applied Mathematics and Computer Science* **20**(3): 445–458, DOI: 10.2478/v10006-010-0033-3.

Liaskos, C. and Ioannidis, S. (2018). Network topology effects on the detectability of Crossfire attacks, *IEEE Transactions on Information Forensics and Security* **13**(7): 1682–1695.

Liu, X., Ren, J., He, H., Wang, Q. and Song, C. (2021). Low-rate ddos attacks detection method using data compression and behavior divergence measurement, *Computers & Security* **100**: 102–107, DOI: 10.1016/j.cose.2020.102107.

de Miranda Rios, V., Inácio, P.R.M., Magoni, D. and Freire, M.M. (2021). Detection of reduction-of-quality ddos attacks using fuzzy logic and machine learning algorithms, *Computer Networks* **186**: 107792, DOI: 10.1016/j.comnet.2020.107792.

Mirkovic, J. and Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms, *ACM SIGCOMM Computer Communication Review* **34**(2): 39–53, DOI: 10.1145/997150.997156.

Monnet, Q., Mokdad, L., Ballarini, P., Hammal, Y. and Ben-Othman, J. (2017). DoS detection in WSNs: Energy-efficient methods for selecting monitoring nodes, *Concurrency and Computation: Practice and Experience* **29**(23), Article ID: e44266, DOI: 10.1002/cpe.4266.

Mowla, N.I., Doh, I. and Chae, K. (2018). CSDSM: Cognitive switch-based DDoS sensing and mitigation in SDN-driven CDNI word, *Computer Science and Information Systems* **15**(1): 163–185, DOI: 10.2298/CSIS170328044M.

Omer, J. and Mucherino, A. (2020). Referenced vertex ordering problem: Theory, applications and solution methods, *Working paper/preprint*, `https://hal.archives-ouvertes.fr/hal-02509522`.

Patan, M. (2012). Distributed scheduling of sensor networks for identification of spatio-temporal processes, *International Journal of Applied Mathematics and Computer Science* **22**(2): 299–311, DOI: 10.2478/v10006-012-0022-9.

Peng, T., Leckie, C. and Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the DoS and DDoS problems, *ACM Computing Surveys* **39**(1): 3, DOI: 10.1145/1216370.1216373.

Pilipczuk, M. and Wahlström, M. (2018). Directed multicut is W[1]-hard, even for four terminal pairs, *ACM Transactions on Computation Theory* **10**(3): 13:1–13:18, DOI: 10.1145/3201775.

Ramanathan, S., Mirkovic, J., Yu, M. and Zhang, Y. (2018). SENSS against volumetric DDoS attacks, *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, USA*, pp. 266–277, DOI: 10.1145/3274694.3274717.

Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A. and Knightly, E. (2009). DDoS-shield: DDoS-resilient scheduling to counter application layer attacks, *IEEE/ACM Transactions on Networking* **17**(1): 26–39.

Studer, A. and Perrig, A. (2009). The Coremelt attack, *in* M. Backes and P. Ning (Eds), *Computer Security—ESORICS 2009: 14th European Symposium on Research in Computer Security*, Springer, Berlin, pp. 37–52, DOI: 10.1007/978-3-642-04444-1_3.

Suchanski, M., Kaniewski, P., Romanik, J., Golan, E. and Zubel, K. (2020). Radio environment maps for military cognitive networks: Density of small-scale sensor network vs. map quality, *EURASIP Journal on Wireless Communications and Networking* **2020**(1): 189, DOI: 10.1186/s13638-020-01803-4.

Uciński, D. (2012). Sensor network scheduling for identification of spatially distributed processes, *International Journal of Applied Mathematics and Computer Science* **22**(1): 25–40, DOI: 0.2478/v10006-012-0002-0.

Wang, K., Du, M., Maharjan, S. and Sun, Y. (2017). Strategic honeypot game model for distributed denial of service attacks in the smart grid, *IEEE Transactions on Smart Grid* **8**(5): 2474–2482.

Wood, R. (1993). Deterministic network interdiction, *Mathematical and Computer Modelling* **17**(2): 1–18.

Zang, X.-D., Gong, J. and Hu, X.-Y. (2019). An adaptive profile-based approach for detecting anomalous traffic in backbone, *IEEE Access* **7**: 56920–56934.

Zargar, S.T., Joshi, J. and Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks, *IEEE Communications Surveys and Tutorials* **15**(4): 2046–2069.

Zekri, M., Kafhali, S.E., Aboutabit, N. and Saadi, Y. (2017). Ddos attack detection using machine learning techniques in cloud computing environments, *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech), Rabat, Morocco*, pp. 1–7.

Konstanty Junosza-Szaniawski received his PhD in mathematics from the Warsaw University of Technology (Faculty of Mathematics and Information Science) in 2004. He obtained his habilitation in mathematics from the same university in 2020. He works as an assistant professor at the Faculty of Mathematics and Information Science of the Warsaw University of Technology. His research interests are related to cybersecurity, graph theory, especially graph coloring, geometrically defined graphs, networks, and graph algorithms.

Dariusz Nogalski received his MSc degree in applied computer science from the Warsaw University of Technology, Faculty of Mathematics and Information Science, Poland. He works as a researcher at the Military Communication Institute, Poland. He is currently pursuing his PhD degree. His present research interests are in networks, optimization, including graph-based optimization, computer and network security, and formal languages.

Paweł Rzążewski received his PhD degree in computer science at the University of Warsaw and is now an assistant professor at the Warsaw University of Technology. His research interests are related to graph theory, graph algorithms, and parameterized complexity.

Received: 7 June 2021
Revised: 2 August 2021
Accepted: 17 September 2021