DOI: 10.15439/2020F106
ISSN 2300-5963 ACSIS, Vol. 21

Exact and approximation algorithms for sensor placement against DDoS attacks

Konstanty Junosza-Szaniawski*, Dariusz Nogalski†, Agnieszka Wójcik*
* *Warsaw University of Technology, Faculty of Mathematics and Information Science*
ul. Koszykowa 75, 00-662 Warszawa, Poland
email: {k.szaniawski, a.wojcik}@mini.pw.edu.pl
† *Military Communication Institute, C4I Systems Department*
ul. Warszawska 22A, 05-130 Zegrze, Poland
email: d.nogalski@wil.waw.pl

***Abstract*—In DDoS attack (Distributed Denial of Service), an attacker gains control of many network users by a virus. Then the controlled users send many requests to a victim, leading to lack of its resources. DDoS attacks are hard to defend because of distributed nature, large scale and various attack techniques.**

One of possible ways of defense is to place sensors in the network that can detect and stop an unwanted request. However, such sensors are expensive so there is a natural question about a minimum number of sensors and their optimal placement to get the required level of safety.

We present 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 above placement problems are NP-hard, two efficient heuristics are designed, implemented and compared experimentally with exact linear programming solvers.

***Index Terms*—DDoS, sensor placement, network safety optimization, minimum multicut, heuristics.**

I. Introduction

DENIAL of Service (DoS) attacks are intended attempts to stop legitimate users from accessing a specific network resource (Zargar et al. [1]). 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 [2]). The attacker gets the control of a large number of users by a virus and then simultaneously performs a large number of requests to a victim server via infected machines. As a result of a large number of tasks, the victim server is out of resources and it cannot provide its service to legitimate users. DDoS attacks are also a problem in the context of Smart Grid environments (SG) (Wang et al. [3], Provos and Holz [4], Cameron et al. [5]). According to [5] availability is more critical than integrity and confidentiality for SG environments.

DDoS attacks are difficult to defend because a large number of machines may be controlled by botnets and participate in an attack, and in consequence, an attack may be launched from a large number of directions. A single bot (compromised machine) sends a small amount of traffic which looks legitimate, but the total traffic at target from the whole botnet is very high. This leads to exhaustion of resources and disruption of legal users (Mirkovic and Reiher [6], Ranjan et al. [7]). Another difficulty is that the attack pattern may be changed frequently. Typically only a subset of botnet nodes conduct the attack at the same time. After certain time, the botnet commander switches to another subset of nodes that conduct the attack.

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. Hence, a natural question which arises is what should be the number of sensors and where to place such sensors. The detection precision may be higher closer to attack sources since its easier to detect spoofed addresses and other anomalies. On the other hand the traffic closer to targets is big enough to accurately recognize 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 in [1].

Jeong et al. [8] and Islam et al. [9] 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 low, especially for low r. In this paper we consider the problem of sensor placement under a different assumption.

In literature, one can find a well-known class of interdiction problems, which can be related to our DDoS problem. Altner et al. [10] study the Maximum Flow Network Interdiction Problem (MFNIP). In 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 MFNIP when an 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 [11]). One of the recent works on 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. [12]). A more general problem on graphs was proposed by Omer and Mucherino [13], which includes, among the others, the interdiction problem. In our DDoS problem we delete vertices, instead of arcs in CMFNIP. Additionally, we consider multiflow instead of a single flow.

From an attacker point of view, a DDoS attack can be modeled as a flow from multiple sources to single target (single commodity flow). We define a directed graph with a capacity function on edges, a set of sources (S) and a set of targets (T). We receive a set of possible attacks $P = \{(S, t_i)$, where $t_i \in T$ is a target of an attack$\}$. An attacker can conduct a single or multiple attack $p \in P$. The strength of an attack is given by a value of a maxflow for $p \in P$. The single attack $maxflow(p)$ can be computed efficiently by Ford-Fulkerson algorithm [14].

Within our DDoS defense approach we want to place sensors in network nodes, which 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. We call $D \subset V$ a set of sensors. The goal of our defense is to limit maximum uncontrolled flow towards each $t \in T$. To achieve that we minimize multi-cut. The case of a single target $|T| = 1$ can be reduced to single pair min-cut/max-flow problem and solved efficiently by a well-known Ford-Fulkerson algorithm [14]. Additionally, in such case the maximum flow is equal to the minimum cut. When the number of pairs is two the problem can be reduced to the single pair case in an undirected graph (Hu's two-commodity flow theorem [15]). When the number of pairs is more than two the problem of multi-cut becomes NP-hard. The reduction goes from the *multiterminal cuts* problem (Dahlhaus et al. [16]), also known as *multiway cuts* (Garg et al. [17]). In the multiterminal cuts problem we are given an edge-weighted graph and a subset of the vertices called terminals, and asked for a minimum weight set of edges that separates each terminal from all the others. When the number of terminals is more than two the multiterminal cuts is NP-hard (proved by Dahlhaus et al. [16]). Garg et al. [17] proved that the undirected 3-way edge cut problem can be reduced to the directed 2-way cut problem.

The main result of this paper are two mixed integer models for optimal sensor placement against DDoS attacks. One model generalises the edge multiterminal cut problem, and the other generalizes node multiterminal cut problem. Hence, both problems described by our models are NP-hard. Moreover we present two efficient heuristics (one for each problem). Finally, we present experimental comparison of solutions given by the heuristics and the mixed-integer programming solvers.

II. Problem definition

A. Problem of optimal sensor placement

Network Model: We assume that the network is modeled as a directed graph $G = (V, E)$ without multiple edges. Every directed edge has assigned a capacity by a function $c : E \to [0, \infty)$. Each node in the network can be interpreted as a router or an autonomous system.

Protected nodes: We use $T \subset V$ to denote a set of $protected$ nodes (a.k.a target nodes) in a 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 can start from any network node ($source$). In practical scenario however we want to limit our attention to a set of sources $S \subset V$.

Attacks: We define a set of possible attacks $P = \{(S, t_i)$, where $t_i \in T$ is a target of an attack$\}$. We don't assume which traffic from a source $s_j \in S$ is legitimate and which one is hostile. Every potential attack $p \in P$ starts from S, is targeted at $t(p)$ and is modeled as a single-commodity flow. Routing policies allow multi-path transmissions from $s_j \in S$ to $t(p)$.

Sensors: A detection sensor can be placed in each network node. When a sensor is placed in a node $v \in V$, then all the incoming and outgoing nodes' edges are assumed controlled. A set of nodes where sensors are placed is called D.

Definition 1: **Attack flow** A function $f : P \times E \to [0, \infty)$ is called $attack\ flow$ if both conditions are satisfied: conservation of flow (1) and capacity constraints (2).

$$\forall_{p \in P} \forall_{u \in V \setminus \{S, t(p)\}} \sum_{v:(v,u)\in E} f_p(v,u) = \sum_{w:(u,w)\in E} f_p(u,w), \quad (1)$$

where $f_p(u,v) = f(p,(u,v))$.

$$\forall_{e \in E} f_p(e) \leq c(e). \quad (2)$$

The attack flow value is given by

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

Definition 2: **Maxflow** By $maxflow_G(p)$, where $t(p) \in T$, we denote the maximum value of f_p (3).

Definition 3: $\mathbf{G \setminus D}$ Having a graph $G = (V, E)$ and a set of sensors D. A graph $G \setminus D$ is a graph $(V \setminus D, E \setminus E_{incident(D)})$, where $E_{incident(D)}$ is a set of edges incident to $d \in D$.

Definition 4: **Uncontrolled flow** An uncontrolled flow for $t \in T$ is a flow for which $f_t > 0$ in a graph $G \setminus D$.

For example, in Fig. 2 all the incoming and outgoing edges of node 5 and 7 are controlled. An uncontrolled flow exists in a graph $G \setminus \{5, 7\}$.

In order to defend against DDoS attack we want to place sensors in a network 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, we search for a placement such that uncontrolled flows are "distributed" among all $t_j \in T$. We want to avoid the situation in which some targets are fully protected (all traffic from S is controlled) and the other targets receive a high portion of an uncontrolled traffic, and in consequence are vulnerable to DDoS attack.

We consider two models *PC* (Placement with required Cardinality) and *PQ* (Placement with required Quality).

In the ***PC*** model we assume that the number of sensors k is given and the task is to find a k-element set $D \subset V$ such that $\max_{t \in T} maxflow_{G-D}(t)$ is minimal. Such 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.

In the second model, denoted by ***PQ***, 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,

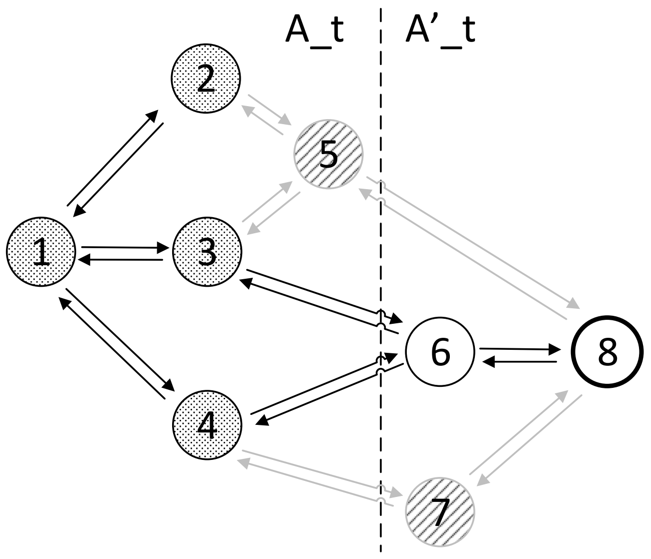

Fig. 1. Cut for $t = 8 \in T$ in G, source nodes $S = \{1, 2, 3, 4\}$, protected nodes $T = \{8\}$ and sensors $D = \{5, 7\}$.

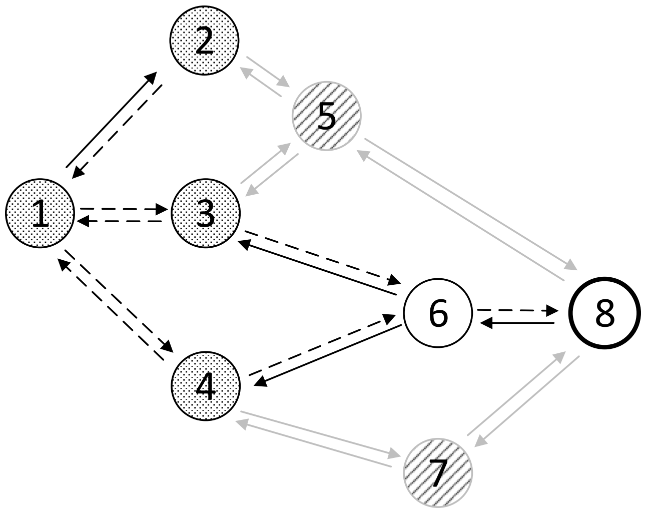

Fig. 2. Uncontrolled flow in G (dashed lines), source nodes $S = \{1, 2, 3, 4\}$, protected nodes $T = \{8\}$ and sensors $D = \{5, 7\}$.

for a given number $q \in [0, 1]$, we ask what a minimal number $k \in \mathbb{N}$ is such that there exists a k-element set $D \subset V$ such that

$$\max_{t \in T} maxflow(t)_{G-D} \leq (1 - q) \cdot \max_{t \in T} maxflow_G(t).$$

For $q = 1$ we get the question: what is the minimal number of sensors that guarantee the total control in the network.

B. Complexity of the optimal sensor placement

In the multiway cuts problem we are given a directed graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}^+$, and a set of t terminals $S = \{s_1, s_2 ... s_t\}$. A (edge) multiway cut in G is a set of nodes (edges) whose deletion separates every terminal from every other terminal (i.e. the remaining graph does not contain any path from s_i to s_j for $i \neq j$). The problem of computing the minimum (edge) multiway cut in directed graphs is NP-hard in case $t \geq 2$ (Garg et al. [17], Theorem 3.1).

Notice that the minimum node multiway cut can be reduced to the problem described by PQ model for $q = 1$. Moreover the minimum edge multiway cut problem can be reduced to the problem described by PC model for $k = 0$. Hence both our problems are NP-hard.

III. Models description

Basic formulation of PC and PQ models: To solve the problem of optimal sensor placement in the sense of models *PC* and *PQ* we use mix-integer programming. Our solution is based on a well-known Ford-Fulkerson Theorem [14] stating that the maximum flow cannot exceed the minimum cut and actually, in our solution we minimize the min-cuts. 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 we place sensors. We start with the *PC* model.

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 $e \in$ belongs to a cut in $G - D$ for t.
- A real variable $M \in \mathbb{R}$, that denotes the value of the minimum cut in $G - D$.

A function to minimize is just M with respect to the below restrictions (4)-(8). For every target $t \in T$ each vertex $s \in S$ belongs to A_t (4). For every target $t \in T$ the vertex t does not belong to A_t (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. The restriction (7) makes sure that the number of sensors is fixed. Finally, the equation (8) bounds the value of the cut with M.

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

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

$$\begin{gathered}\forall_{t\in T}\ \forall_{(u,v)\in E}\\ cutT[t,u,v] \geq a[t,u] - a[t,v] - d[u] - d[v]\end{gathered} \tag{6}$$

$$\sum_{v\in V} d[v] = k \tag{7}$$

$$\forall t\in T \quad \sum_{(u,v)\in E} cutT[t,u,v]\cdot c[u,v] \leq M \tag{8}$$

To obtain the *PQ* model it is enough to replace the target function to minimize by $\sum_{v\in V} d[v]$, omitting the restrictions (7) and (8), and adding the restriction

$$\begin{gathered}\forall t\in T \sum_{(u,v)\in E} cutT[t,u,v]\cdot c[u,v] \leq\\ (1-q)\cdot \max_{t\in T} maxflow_G(t)\end{gathered} \tag{9}$$

where $\max_{t\in T} maxflow_G(t)$ is equal to the value of max minimum cut M_t in G (result of *PC* model for $k=0$).

IV. Algorithms description

Relaxed formulation of PC and PQ models: In this formulation we relax two types of variables to allow the fractional sensor placement (first) and fractional traffic control (second):

- 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 III) 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 edge as fully controlled. When no sensor is placed in neither u nor v such edge is uncontrolled. Whereas in the relaxed formulation we allow fractional sensor placement (d variables) and fractional control of edges in a cut ($cutT$ variables).

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

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

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, *PCIterativeBestSensor* heuristics takes k (number of sensors) as input and *PQIterativeBestSensor* heuristics q (quality factor).

A. PC Iterative Best Sensor Placement

The algorithm *PCIterativeBestSensor* constitutes $k+1$ iterations. In each $\{1,..,k\}$ iteration, 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.
In the last iteration, the LP relaxation is solved assuming fixed sensor placements for all $v\in D$ (line 11).

Algorithm 1 PCIterativeBestSensor

```
1:  Require $G, c, T, S, k$
2:  Create the relaxed PC problem (section IV) with goal
    $minimize\ M$. Add constraints {(4),(5),(6),(7),(8)} to the
    $problem$.
3:  Let's initiate a set of vertices in which we place sensors
    $D = \emptyset$
4:  for $i = 1,..,k$ 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 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 for
11: Solve the $problem$
12: Retrieve $M$ from the $problem$ solution
13: Return $(D, M)$
```

B. PQ Iterative Best Sensor Placement

The preparatory step of the algorithm *PQIterativeBestSensor* is a computation of the value of $\max_{t\in T} maxflow_G(t)$ (line 2). In each while loop, linear program relaxation is solved (line 6). 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 7). Among the $|L|$ best sensor locations, the single best (max) one v_{max} is selected and added to the model as a constraint (line 9). The constraint fixes a sensor in the location v_{max} in the next iterations.

V. Computational results

A. Experiment Setup

The two models *PC* and *PQ* and two algorithms *PCIterativeBestSensor* and *PQIterativeBestSensor* 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 9 types of grid networks: $Net|V|$, where $|V| = \{64, 81, 100, 121, 144, 169, 196, 225, 256\}$ indicates the

Algorithm 2 PQIterativeBestSensor

```
1: Require $G, c, T, S, q$
2: Compute a value of $\max_{t\in T} maxflow_G(t)$
3: Create the relaxed $PQ\ problem$ (section IV) with goal $minimize \sum_{v\in V} d[v]$. Add constraints {(4),(5),(6),(9)} to the $problem$
4: Let's initiate a set of vertices in which we place sensors $D = \emptyset$
5: while $(\exists t \in T \sum_{(u,v)\in E} cutT[t,u,v] \cdot c[u,v] > (1-q) \cdot \max_{t\in T} maxflow_G(t))$ do
6:     Solve the $problem$
7:     Let $L = \{v$, s.t. $v \in V \setminus D$ and $d[v] \neq 0$ and $d[v] == max\{d[j]\}_{j\in V\setminus D}\}$
8:     Choose randomly $v_{max} \in L$, where probability of selecting an element $v_{max}$ equals $\frac{1}{|L|}$
9:     Add constraint $d[v_{max}] == 1$ to the $problem$
10:    $D = D \cup \{v_{max}\}$
11: end while
12: Return $D$
```

number of vertices in a network. All of 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. 3. Each vertex in a graph may correspond to a router or an autonomous system in telecommunication network.

For the purpose of 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 $T_{i=1..4} \subset V$) were generated (all vertices V have equal probabilities). For each target instance T_i, four random instances of source locations were generated ($S_{j=1..4} \subset V \backslash T_i$)(all vertices $V \backslash T_i$ 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, for all scenarios we assumed $|T| = 10$ and $|S| = 40$.

B. Scenario1: PC problem, Net100, 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 Fig. 4 demonstrates the average volume of uncontrolled traffic (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* heuristics. The observed average objective values of *PCIterativeBestSensor* are higher than those of *PC* by up to 8%.

The diagram Fig. 5 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*.

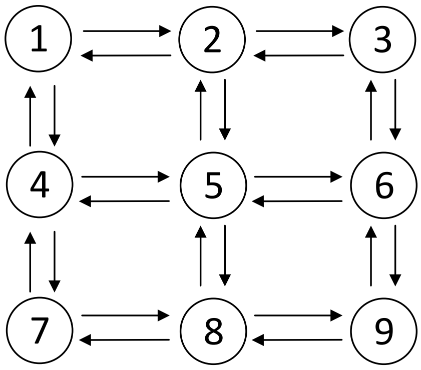

Fig. 3. An example of a small grid network $|V| = 9$

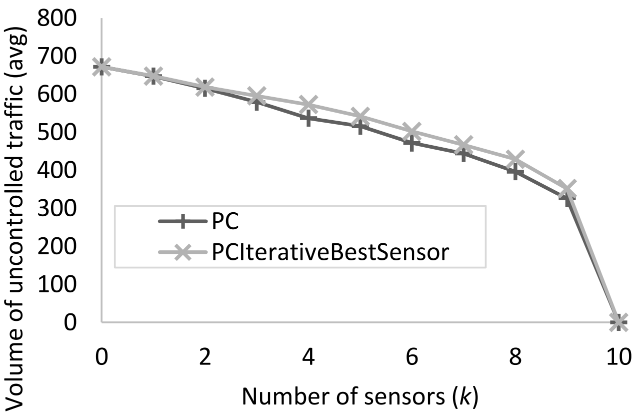

Fig. 4. Scenario1, volume of uncontrolled traffic (avg), *PC* vs. *PCIterativeBestSensor*

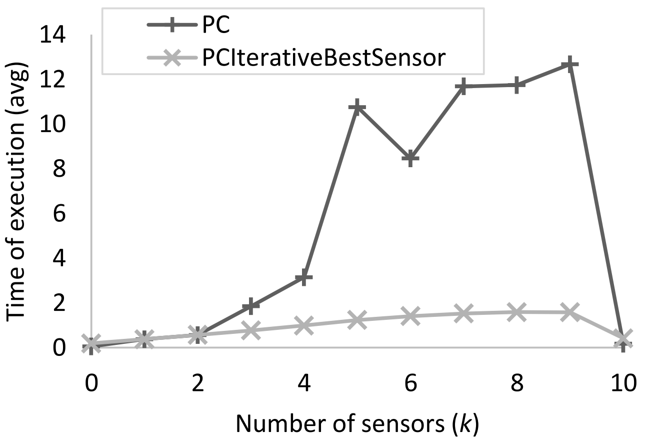

Fig. 5. Scenario1, time of execution (avg) [s], *PC* vs. *PCIterativeBestSensor*

C. Scenario2: PC problem, k=5, increasing size of the grid Net64, Net81, ... , Net169

The experiments were conducted for the grid networks: *Net64, Net81, Net100, Net121, Net144, Net169*. The number of sensors was fixed $k = 5$.

The diagram Fig. 6 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 favour of the heuristics.

D. Scenario3: PQ problem, Net196, increasing value of quality factor

The experiments were conducted for the grid network *Net196*. The value of quality factor was increasing $q \in \{0.1, 0.2, ..., 1.0\}$.

The diagram Fig. 7 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* 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 Fig. 8 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*.

E. Scenario4: PQ problem, q=0.5, increasing size of the grid Net121, Net144, ... , Net256

The experiments were conducted for the grid networks: *Net121, Net144, Net169, Net196, Net225, Net256*. The quality factor was fixed $q = 0.5$.

The diagram Fig. 9 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 favour of the heuristics.

F. Summary of simulation results

The *PC* algorithms simulations lead to a number of observations. Firstly, for all test networks, as the number of sensors

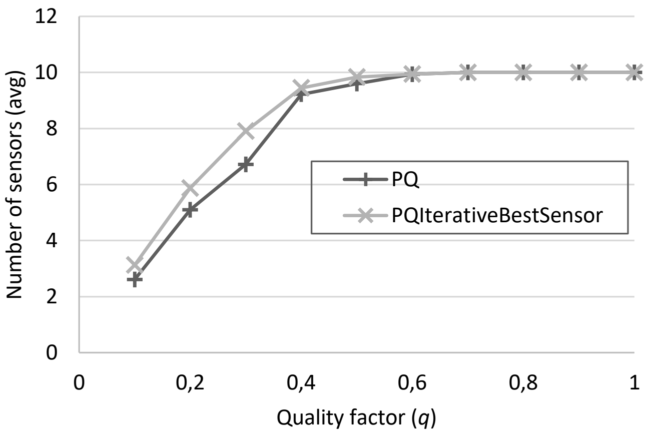

Fig. 7. Scenario3, number of sensors (avg), *PQ* vs. *PQIterativeBestSensor*

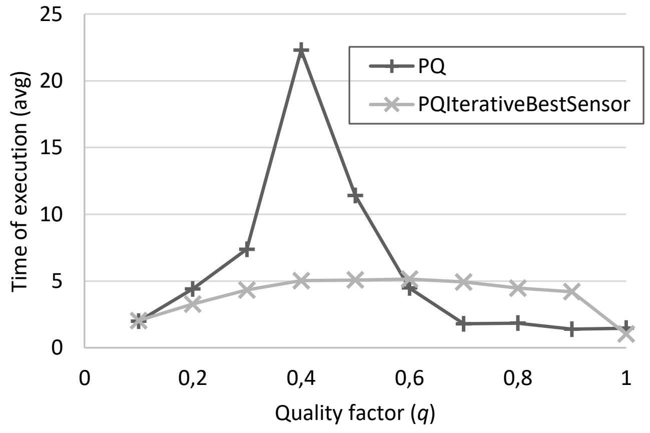

Fig. 8. Scenario3, time of execution (avg) [s], *PQ* vs. *PQIterativeBestSensor*

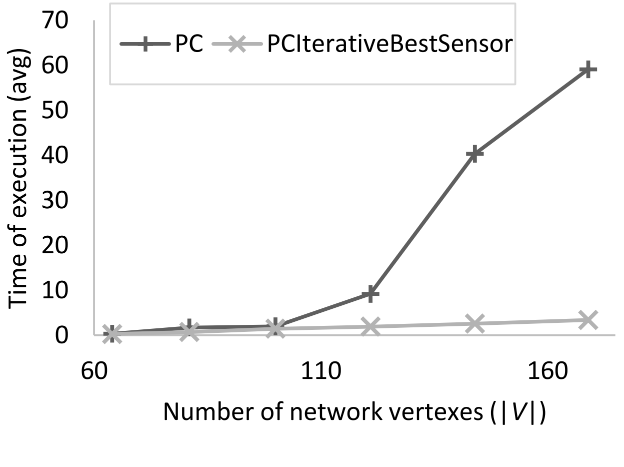

Fig. 6. Scenario2, time of execution (avg) [s], *PC* vs. *PCIterativeBestSensor*

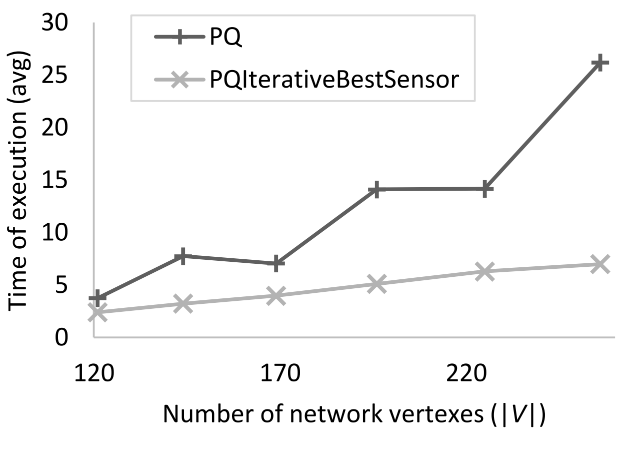

Fig. 9. Scenario4, time of execution (avg) [s], *PQ* vs. *PQIterativeBestSensor*

increases, the volume of uncontrolled traffic decreases to zero, for both *PC* model and *PCIterativeBestSensor* heuristics. 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 favour of the heuristics.

The *PQ* algorithms simulations lead 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* heuristics. Secondly, in the worst observed cases *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 favour of the heuristics.

VI. Conclusions

As demonstrated for some medium-sized 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. 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.

Finally, for large networks, the execution time gap between the two models and their corresponding heuristics increases significantly in favour of the heuristics.

References

[1] 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*, vol. 15, no. 4, pp. 2046–2069, 2013. doi: 10.1109/SURV.2013.031413.00127

[2] 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.

[3] 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*, vol. 8, no. 5, pp. 2474–2482, Sep. 2017. doi: 10.1109/TSG.2017.2670144

[4] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 2007. ISBN 978-0321336323

[5] 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*, vol. 10, no. 3, pp. 3010–3019, 2019. doi: 10.1109/TSG.2018.2817046

[6] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, p. 39, 2004. doi: 10.1145/997150.997156. [Online]. Available: http://portal.acm.org/citation.cfm?doid=997150.997156

[7] 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*, vol. 17, no. 1, pp. 26–39, 2009. doi: 10.1109/TNET.2008.926503

[8] S. B. Jeong, Y. Choi, and S. Kim, "An effective placement of detection systems for distributed attack detection in large scale networks," in *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. H. Lim and M. Yung, Eds., vol. 3325. Springer, 2004. doi: 10.1007/978-3-540-31815-6_17 pp. 204–210. [Online]. Available: https://doi.org/10.1007/978-3-540-31815-6_17

[9] 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*, pp. 653–657, 2008. doi: 10.1109/INNOVATIONS.2008.4781681

[10] D. S. Altner, Ö. Ergun, and N. A. Uhan, "The maximum flow network interdiction problem: Valid inequalities, integrality gaps, and approximability," *Oper. Res. Lett.*, vol. 38, no. 1, pp. 33–38, 2010. doi: 10.1016/j.orl.2009.09.013. [Online]. Available: https://doi.org/10.1016/j.orl.2009.09.013

[11] R. Wood, "Deterministic network interdiction," *Mathematical and Computer Modelling*, vol. 17, no. 2, pp. 1 – 18, 1993. doi: 10.1016/0895-7177(93)90236-R. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089571779390236R

[12] M. Hemmati, J. Cole Smith, and M. T. Thai, "A cutting-plane algorithm for solving a weighted influence interdiction problem," *Computational Optimization and Applications*, vol. 57, no. 1, pp. 71–104, Jan. 2014. doi: 10.1007/s10589-013-9589-9. [Online]. Available: https://doi.org/10.1007/s10589-013-9589-9

[13] J. Omer and A. Mucherino, "Referenced vertex ordering problem: Theory, applications and solution methods," Mar. 2020, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02509522

[14] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, p. 399–404, 1956. doi: 10.4153/CJM-1956-045-5

[15] T. Hu, "Multi-commodity network flows," *Operations Research*, vol. 11, no. 3, p. 344–360, 1963. doi: 10.1287/opre.11.3.344

[16] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis, "The complexity of multiterminal cuts," *SIAM Journal on Computing*, vol. 23, no. 4, pp. 864–894, 1994. doi: 10.1137/S0097539792225297

[17] N. Garg, V. V. Vazirani, and M. Yannakakis, "Multiway cuts in directed and node weighted graphs," in *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, ser. Lecture Notes in Computer Science, S. Abiteboul and E. Shamir, Eds., vol. 820. Springer, 1994. doi: 10.1007/3-540-58201-0_92 pp. 487–498.