2022

"MystifY": A Proactive Moving-Target Defense for a Resilient SDN Controller in Software Defined CPS

Mohamed Azab

Mohamed Samir

Effat Samir
Old Dominion University, efath002@odu.edu

Original Publication Citation

Azab, M., Samir, M., & Samir, E. (2022). "MystifY": A proactive moving-target defense for a resilient SDN controller in software defined CPS. *Computer Communications*, 189, 205-220. https://doi.org/10.1016/j.comcom.2022.03.019

Computer Communications 189 (2022) 205–220

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

"MystifY": A proactive Moving-Target Defense for a resilient SDN controller in Software Defined CPS

Mohamed Azab [a,*], Mohamed Samir [b], Effat Samir [c]

[a] The Department of Computer and Information Sciences, Virginia Military institute, Lexington, VA, USA
[b] The Arab Academy for Science, Technology and Maritime Transport (AAST-MT), Alexandria, Egypt
[c] The Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA, USA

ARTICLE INFO

Keywords:
CPC
SDN
CPP
MTD
Smart grid
Network security

ABSTRACT

The recent devastating mission Cyber–Physical System (CPS) attacks, failures, and the desperate need to scale and to dynamically adapt to changes, revolutionized traditional CPS to what we name as Software Defined CPS (SD-CPS). SD-CPS embraces the concept of Software Defined (SD) everything where CPS infrastructure is more elastic, dynamically adaptable and online-programmable. However, in SD-CPS, the threat became more immanent, as the long-been physically-protected assets are now programmatically accessible to cyber attackers. In SD-CPSs, a network failure hinders the entire functionality of the system. In this paper, we present MystifY, a spatiotemporal runtime diversification for Moving-Target Defense (MTD) to secure the SD-CPS infrastructure. In this paper, we relied on Smart Grid networks as crucial SD-CPS application to evaluate our presented solution. MystifY's MTD relies on a set of pillars to ensure the SDN controller resiliency against failures and attacks. The 1st pillar is a grid-aware algorithm that optimally allocates the most suitable controller–deployment location in large-scale grids. The 2nd pillar is a special diversifier that dynamically relocates the controller between heterogeneously configured hosts to avoid host-based attacks. The 3rd pillar is a temporal diversifier that dynamically detours controller–workload between multiple controllers to enhance their reliability and to detect and avoid controller intrusions. Our experimental results showed the efficiency and effectiveness of the presented approach.

1. Introduction

The ubiquitous computing and smart-everything revolution are trends aiming to embed computational capabilities into physical components mainly to make our lives easier, safer, and more reliable. However, the incremental development of such innovations are relying on conventional technologies, which were not designed to support such cyber–physical interaction.

The recent overwhelming attack-waves targeting smart mission critical infrastructure assets [1–3] across the globe showed how vulnerable they can be against highly motivated adversaries. With the desperate need to scale, and to dynamically adapt Cyber–Physical Systems (CPSs), designers started to embrace the Software Defined (SD) everything concept to present a more evolved version of CPS.

We coined the term Software Defined CPS (SD-CPS) to describe such an evolution of the new systems that offer more elasticity and adaptability by enabling online re-programmability of its components. Such adaptation ability, and the enhanced awareness, gave SD-CPS the advantage over conventional CPS systems when it comes to reliable operation.

However, relying on programmable software components to control the infrastructure of the CPS opened the door for new classes of threats and attacks [4–6]. Such attacks were never considered in the static old fashion infrastructure equipment. As for any smart system, the network infrastructure is the main backbone supporting the system's operation. Therefore, in this paper, we present a novel solution to improve the Software Defined Network (SDN) security in SDN-enabled Smart Grids as one of the mission critical SD-CPS applications.

The SDN controller is a centralized component that maintains a full control of the entire network. Being fully aware of the network details, enabled such a controller to dynamically adjust the attached SDN programmable switches, on the packet level, towards the best interest of the entire network. Such design structure gave the SDNs a high level of adaptability, reliable operation, and much better Quality of Service (QoS) assurances. However, such centralized control is a major single point-of-failure. Given the critical value of SDN controllers, its failure became a devastating event that cripples the entire network operation and the application relying on it. When it comes to mission critical

* Corresponding author.
E-mail addresses: azabmm@vmi.edu, mazab@vt.edu (M. Azab), mohamed.samir@aast.edu (M. Samir), effat_samir@mena.vt.edu, efath002@odu.edu (E. Samir).

https://doi.org/10.1016/j.comcom.2022.03.019
Received 21 August 2021; Received in revised form 25 March 2022; Accepted 26 March 2022
Available online 2 April 2022

CPS applications like Smart Grids, an SDN controller failure can lead to catastrophic situation.

Researchers addressed this problem by replicating and optimizing the location-selection of the SDN controller(s). Such attempts lead to what is known as the Controller Placement Problem (CPP), which was first introduced by Heller et al. in 2012, as a k-median, or NP-hard problem [7]. The main goal for Heller's study was to determine the number of needed controller(s), and the optimal placements for these controller(s) in a large-scale network.

The controller security problem that we focus on is the static initial placeman and its impact on controller security to enable runtime assurances that such controller(s) will keep operational as expected in presence of attacks. In this paper, we present MystifY, a novel grid-aware proactive Moving-Target Defense (MTD) mechanism that applies spatiotemporal runtime diversification to secure the SDN controller(s) in SD-CPS network.

In space, we rely on a lightweight virtualization technology to enable SDN controller(s) live-migration between some preconfigured and optimally selected placements to avoid host-based attacks and failures. In time, we present a novel distributed SDN proxy module to enable dynamic workload detouring between diversified replicas to avoid unexpected controller intrusions. Further, with multi-controller feedback on each incoming switch message, we implemented a lightweight Intrusion Detection System (IDS) module, to detect misbehaving controllers. This module will automatically remove misbehaving controllers (until replaced) from the active controller pool.

The multidimensional changes enabled by MystifY, massively complicate controller traceability. Further, enabling multi-controller management of same switch traffic increases the network resilience not only against attacks but also against intentional/unintentional failures. Throughout this paper, SDN-enabled Smart Grid is used as a demonstrative example for a crucial SD-CPS application. However, the presented approach can be easily applied to other mission critical large-scale SD-CPS applications.

The proposed framework of MystifY relies on three main pillars, which are; (1) The AllocatoR Module, (2) The Migrator Module, and (3) The Proxy Module. Each one plays an essential role in protecting the SDN controller(s) in Smart Grid networks. To ensure appropriate time stamping for some of the supporting ideas, a skeleton architecture of a basic idea of two of the early designed pillars were published for time stamping [8,9]. In this paper, we present a comprehensive novel defense framework with detailed designs and in-depth evaluations for the proposed SDN controller resilience mechanism in the context of a mission critical application like smart grids.

With real-time monitoring and analysis of the active Smart Grid traffic and the insights provided by MystifY's Smart grid-aware IDS module, MystifY builds the migration strategy for the SDN controller(s) to satisfy the targeted objectives, best serve the grid needs, and ensures safe and reliable network operations. MystifY's analysis of the grid distribution and workload, will guide the process towards selecting the best targets for migration, and the best relocation points. The migration module will customize the migration process to best serve the network and the smart grid needs.

The contributions of this paper can be summarized as follows:

1. Present a grid-operation and performance aware, multi-objective CPP solution for mobile controller placement in SDN-enabled Smart Grids.
2. Enable a dynamic controller/workload migration inducing multidimensional spatiotemporal MTD for the SDN controller(s).
3. Enable a multi-controller management of same switch traffic to enhance the network resiliency against attacks and failures.
4. Integrate MystifY with a novel SDN-enabled Grid simulator for grid-aware evaluations.

The rest of this paper is organized as follows: Section 2, briefly describes the key concept beyond CPS network security and other MTD mechanisms. Section 3, presents MystifY threat model. Section 4, explains MystifY framework architecture in details. Section 5, presents our simulation studies and the analysis results. Finally, Section 6, concludes by summarizing our contributions and future work.

2. Literature review

2.1. MTD technique

The main goal of MTD technique in CPS systems works by enabling frequent dynamic movement between multiple configuration points in the cyber domain. It is emerged as a solution that breaks the static nature of CPS systems introducing programmed heterogeneous changes in its operation. Therefore, MTD solutions were presented discussing various techniques that can be classified based on what to change and how to change it. Fig. 1 gives a quick overview of these points.

To summarize, within the context of Smart Grids, or in general CPS systems, the attack surface includes:

[1] Network platform, such as the forwarding links and routing nodes [10,11].
[2] Endpoint data, such as the IP address, MAC address, port, and protocol [12–14].
[3] Platform environment, such as the operating system, software application, and some runtime environment parameters [15–17].

The MTD application strategies include:

[1] Game-theoretic strategies, to model the interaction between the system defender and the attacker [18–20].
[2] System state randomizations strategies, the basic idea is to randomly and continuously change the system configuration or parameters to make the systems inaccessible for attackers, such as IP address randomizations and IP hopping techniques [21–23].
[3] Strategy formulation based on analytical model building, this method uses machine learning, deep learning, and neural networks algorithms to accurately percept the network status [24–26].
[4] Software diversification strategies, this technique dynamically changes the software instruction codes itself to create multiple execution tasks, while ensuring the software functional equivalence [27–29].
[5] Virtualization-based strategies, it has presented as a technology to virtualize the hardware-based systems and shuffle the endpoint control and application layer [30,31].

The next subsection highlights the literature review discussing different MTD solutions presented to secure CPSs.

2.2. Related work

Coupling SDN and CPS noted as an area of great interest to develop advanced communication architectures. Recently, Software Defined Networking inspired systems designers to develop more economical, reliable, secure CPS systems and applications especially in the Smart Grid [32–34].

Aswin. C. Pappa et al. [35] introduced the use of MTD strategy in power grid SCADA environments to mitigate attacks, leverage the network with end-to-end IP hopping techniques. Aswin designed his MTD system architecture on Iowa's state PowerCyber testbed to study the delay and throughput characteristic in a realistic environment. He analyzed the proposed solution on two types of attacks; the address range exhaustion attack, and the traffic analysis attack.

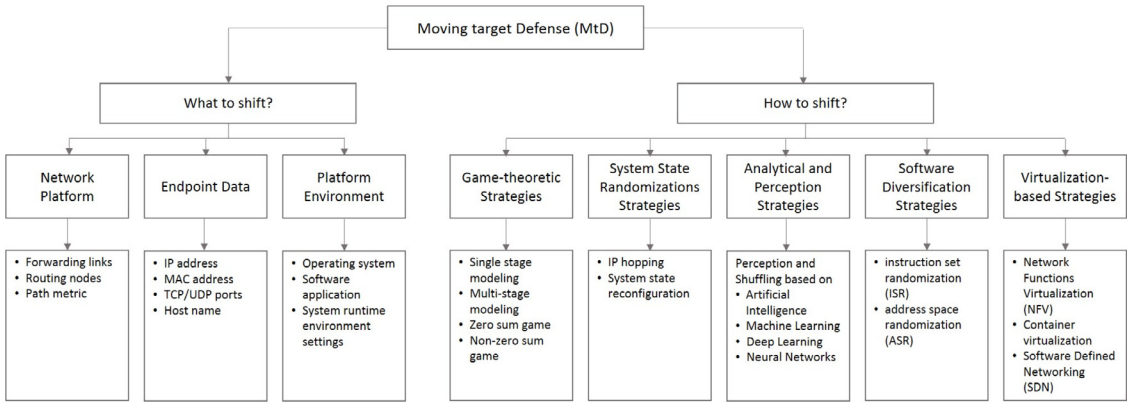

Fig. 1. MTD technique classification.

Jue Tian et al. [36] discussed the use of a hidden MTD approach to mitigate False Data Injection (FDI) attacks in Smart Grid networks. He developed a novel algorithm to compute the needed parameter perturbations to the transmission lines of a Distributed Flexible AC Transmission System (D-FACTS) under the AC and DC power flow scenarios. Further, Jue analyzed a class of False Data Injection attacks in traditional MTD systems and proposed an enhanced approach that strategically deploy D-FACTS systems and preclude the attacks. Moreover, he relied on IEEE 14-Bus system to conduct the simulation analysis and compare the completeness and stealthiness of the proposed approach.

Abdullah Aydeger et al. [37] introduced an SDN-based MTD mechanism to protect the networks against DDoS crossfire attacks. The proposed MTD mechanism relied on routing the traffic on congested links during the attack time. This approach consists of two main defense mechanisms. First, the "proactive stage" which obfuscate the links during the potential link-map creation of the attackers making it harder to launch attacks. Second, the "active stage" which detect and mitigate during the attacks. For his experimental work, such mitigation and detection techniques are implemented using Mininet [38] emulator and Floodlight SDN controller [39].

J. B. Hong et al. [40] deployed a new MTD mechanism to frequently change the attack surface in SDN-enabled networks. Jin realized his proposed to optimally reconfigure the network topology system by exploiting the SDN functionality. He introduced a new Shuffle Assignment Problem (SAP) to enhance network security. In addition, he proposed the Shuffle-based online MTD mechanism to select an optimal countermeasure against detected attacks using the topological distance metric.

J. Steinbergeret al. [41] investigated the effectiveness of MTD in the high-speed SDN-enabled networks to limit the effects caused by large scale cyber-attacks. Jessica presented a DDoS defense solution that combines the use of Software Defined Everything (SDx) and MTD. The MTD system is implemented using ONOS, a carrier-grade SDN network operating system [42]. However, it relied on different Border Gateway Protocol (BGP) routes to reshape the network and some IP-hopping strategies to setup a honeypot during the attack time.

Tao Hu et al. [43] proposed a Reliable and Load balance-aware Multi-Controller Deployment (RLMD) strategy to enhance the SDN-enabled networks reliability. Tao introduced an SDN controller placement clustering algorithm to deploy a reliable SDN controller(s) based on the path quality and weighing the node efficiency. Additionally, he introduced a Multiple Domain Partition (MDP) algorithm to connect the network switches with these controllers according to the controller-load balancing rate and the node attractability.

P. Kampanakis et al. [44] investigated the coupling between MTD and SDN in details. He discussed the potential countermeasures attacks that could take to circumvent them, and the overhead of implementing MTD in SDN networks. However, he evaluated the performance using one traditional of Cisco's platform kit [45].

Nathan et al. [46] explored the application of moving-target defenses in the context of real-time systems and enterprise-computing applications such as power grids. He relied on the randomization-based techniques as a promising class of defense that offer low overhead and high protection against both data-only attacks and control-flow hijacking. In addition, He discussed the worst case execution time and average case execution time without sacrificing entropy, and how to deploy a protected mixed-criticality workloads.

Bradley et al. [47] discussed the implementation of mixed time and event-triggered architecture to maintain safety and availability in CPS systems. He presented such approach based on the ARINC 653 architecture to provide a reliable and predictable operation upon normal condition, and reconfiguration and rapid detection upon attacks. In addition, He leveraged an Advanced Emergency Braking System (AEBS) and a hardware-in-the-loop testbed as a case study to evaluate the effectiveness of the proposed approach.

Bradley et al. [48] discussed the new attack vectors in autonomous vehicles CPS systems and its challenges again, especially the composition of much amount of legacy software, third party applications, and remote Application Programming Interfaces (APIs). He presented the implementation of instruction set randomization (ISR), and address space randomization (ASR) to protect against code injection and code reuse attacks in CPS systems. He considered the problem of code reuse attacks, detecting code injections, and fast reconfigurations that maintain the stability and safety of autonomous vehicle controller(s). Based on ASR, and ISR, the presented approach prevents attackers from grabbing enough information necessary to code reuse attacks and perform code injections.

Jairoet al. [49] proposed an MTD strategy for multi-vehicle systems to protect against cyber-attacks. He characterized the trade-off between the performance degradation and impact mitigation. Additionally, he illustrates the viability of the strategy in two applications, (1) vehicular platooning, (2) Unmanned Aerial Vehicle (UAV) formation. Moreover, he introduced different types of MTD mechanisms to cover more general control systems framework, at both the sensors units and the controller(s) level.

Samir et al. [50] proposed SD-CPC as an SDN Controller Placement Camouflage (CPC) based on the game theory for MTD. It relied on the Zero-Sum game to model stochastic game between an attacker and the system defender. In addition, SD-CPC considers the network vulnerabilities, evaluates the risk level in real-time using Bayesian Attack Graph (BAG), to contentiously change the controller(s) placement. SD-CPC is mainly applied at the SDN Application layer. SD-CPC presents a top-level application development in the SDN architecture model. In SD-CPC, the Zero-Sum game presents the security/defense solution, and the BAG analysis presents the security risk assessment process. Regardless of the different network used and the different

results, MystifY uses a real-time feedback from our embedded IDS to monitor the network state and guide the migration process. SD-CPC used heuristic calculations based on the stochastic game theory and some BAG analysis to assess the security level of the network nodes. Hence, MystifY's replication and migration modules, provides accurate and optimized migrations. The migration decision occurs based on live indicators not just assumptions based on historical calculations.

Samir et al. [51] discussed the need of addressing CPP challenge in Smart Grids. It did not present any security solution for this challenge. It just simulated the grid and analyzed the tradeoff between CPP metric to select to optimal secure location later on. In addition, it did not present on any MTD technique, unlike MystifY which mainly enables a dynamic controller/workload migration inducing multidimensional spatiotemporal MTD for the SDN controller(s).

It is worth noting that none of the presented research works managed to address the SDN controller resiliency in SD-CPS applications. The SDN controller is a single point-of-failure which makes it a very motivating target for attacks. In this paper, MystifY is presented as a novel approach that enables transformative manipulations of the SDN controller operation to secure the network infrastructure of the SD-CPS application. The unique features offered by MystifY were individually presented in previous research leaving many doors open for attacks and failures. However, based on the best of our knowledge, none of the presented systems managed to comprehensively present this unique set of features on one framework.

3. Threat model

Researchers noted that SDN controller is a major single point-of-failure with massive impact on the network operation and stability. Given the contemporary oblivious defense mechanisms to secure such critical component, attackers would be very motivated to target such controllers to either take control of the network, or the SD-CPS application using it.

The controller is the main component in SDN networks. It handles all control commands, traffic signaling, and policies. Compromised controllers can grant attackers the ability to break system policies, manipulate system information, and perform severe malicious actions. In its basic essence, an SDN controller is a programmed server software running on a host somewhere in the network. Attackers can target such valuable asset directly (through that programmed platform), or indirectly (through the main host machine vulnerability) and control the entire network.

In this paper, we assume that resourceful attackers may try to defeat the system access control; on the host level, the attacker is capable of initiating a wide range of zero-day attacks to disrupt the hosting server of the SDN controller. Additionally, he can allocate the placement of such hosting server, scan it for possible weaknesses, and initiate attacks to cripple the enclosed applications as well.

Therefore, we consider that such powerful attackers would be careful enough not to be detected by any defense tool. For that, they will waste a considerable amount of time to find the hosting servers, and to decide which class of attacks can be exploited to cripple them. Further, we assume that attackers may gain access to one or more controllers and manage to manipulate their behavior. Finally, we assume that the attackers has no control of any of MystifY's modules to help them in controlling the exact placement of the SDN controller(s).

4. MystifY framework description

The main goal of MystifY MTD framework is to ensure a resilient SDN controller operation for the SDN-enabled power grids even in presence of potential attacks. The proposed framework applies a real-time spatiotemporal diversification to frequently change the location of the running SDN controller(s), confusing the attackers. As mentioned in Section 3, the target is to protect the running controller from attackers exploiting either host vulnerabilities or controller programming vulnerabilities.

MystifY is supported by three interrelated pillars enabling it to induce some real-time changes to the controller operational configuration.

The first pillar is the "AllocatoR" module, a grid aware controller placement mechanism that carefully selects the optimal location for the SDN controller and the optimal alternative locations for MystifY to use. The "AllocatoR" module applies a complicated algorithm that considers the various runtime changing grid needs to fill the controller hopping list for the second and third pillars to operate on.

The second pillar is an evolutionary version of a Proactive Attack-and-Failure Resilience (PAFR) [8]; a live migration module. MystifY customizes and manages PAFR to enable controller live migration between a set of predetermined heterogeneously configured and allocated locations to complicate controller tractability and to avoid host-based attacks.

The third pillar is an evolutionary version of "Repoxy" [9]; a smart SDN proxy. MystifY adapts, customizes, and enhances Repoxy's ability to work with multiple controllers, enabling multiple controller management of same switch, dynamic workload migration, and controller misbehavior detection.

In this paper, Smart Grid is considered a case study for an SD-CPS crucial application. We integrated MystifY with a customized version of PYGRID [32]; a novel SDN-enabled Smart Grid simulator to investigate the effectiveness and efficiency of the proposed system. Fig. 2 illustrates the main components participating in the construction and evaluation of MystifY framework.

In the following sections, we will describe the MystifY managed handling of the module set towards a resilient SDN controller operation.

4.1. The allocatoR module

The "AllocatoR" module is considered as the first mastermind pillar in MystifY. It represents a grid-aware module relying on node placement and clustering algorithms. The "AllocatoR" module offers a solution for the Controller Placement Problem (CPP) in SD-CPS environment, determining the required number of SDN controller(s) and their optimal locations within the targeted network topology.

In the "AllocatoR" module, we exploit four latency metrics introduced in [7,52,53] to solve the CPP problem. Various research studies introduced the average and worst case latencies as the most important metrics that influence the network performance significantly. As a consequence, in this paper, we managed to evaluate the performance metric for the average and worst latency metrics between both the "nodes-to-controller" and "controllers-to-controllers".

In MystifY, the "Host Selector" module relies on the calculated "nodes-to-controller" latencies to select a list of controller migration locations for MystifY to operate on. In addition, the "Shuffler" module relies on the real-time feedback information from the "AllocatoR" module to orchestrate the workload migration pattern. This information depends on the calculated latency metrics between each "controller-to-controller".

A. Controllers selection calculations

Generally, clustering algorithms rely on different resilience and performance metrics to achieve the optimal placement results. In SD-Smart Grid networks, the SDN network can be described as network graph G(N,E);

N = set of nodes n = $\{1, 2, 3, \ldots\ldots, n\}$.
E = set of edges between n nodes.
Therefore, the AllocatoR's main goal is to allocate the optimal placement S' such that $|S'| = c$, where c is the number of SDN controller(s) connected to the network at any of S' location.

• Latency metrics between "nodes-to-controller":

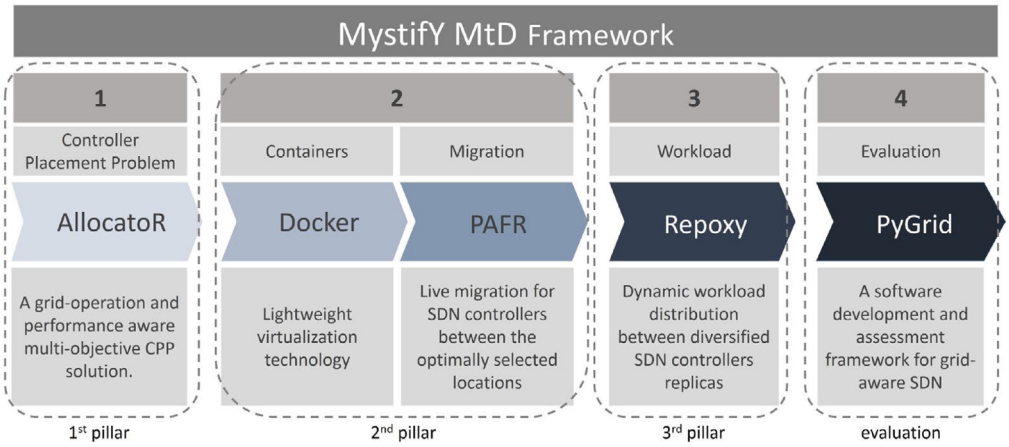

Fig. 2. Mystify main system components.

$L_{avg}(S')$: The average distance between SDN controller(s) nodes and other nodes connected to it, is calculated based on the following equation;

$$L_{avg}(S') = \frac{1}{n}\sum_{v\in V}\min_{(s\in S')} d(v,s) \tag{1}$$

$d(v,s)$: The shortest path from node v to node s, v ϵ N and s ϵ N.
$L_{wc}\left(S'\right)$: The worst latency or the distance between SDN controller(s) nodes and other nodes connected to it, is calculated based on the following equation;

$$L_{wc}\left(S'\right) = \max_{(v\in V)}\min_{(s\in S')} d\left(v,s\right) \tag{2}$$

• Latency metrics between "controller-to-controller":
$L_{avgcc}\left(S'\right)$: The average distance between the SDN controller(s) themselves, is calculated based on the following equation;

$$L_{avgcc}\left(S'\right) = \frac{1}{k}\sum_{c_i,c_j\in S'} d\left(c_i,c_j\right) \tag{3}$$

$d\left(c_i,c_j\right)$: The shortest path between controller c_i and controller c_j, as c_i, c_j $\epsilon S'$.
$L_{wcc}\left(S'\right)$: The worst latency or distance between the SDN controller(s) themselves, is calculated based on the following equation;

$$L_{wcc}(S') = \max_{(c_i,c_j\in S')} d(c_i,c_j) \tag{4}$$

The following Algorithms 1 and Algorithm 2 demonstrate the applied CPP operation and the latency calculations in real-time under normal condition and in presence of attacks.

4.2. The migrator module, MystifY-managed PAFR

MystifY adopts and customizes PAFR migrator to enable the controller mobility between hosts. MystifY-managed PAFR relies on "Docker" [54,55], Linux containers virtualization technology to encapsulate the SDN controller along with its applications. Encapsulating the controller and its applications enables MystifY to easily instantiate controllers remotely on any location on the grid from a cashed image without losing the network state. Enabling such feature massively reduces the downtime due failures and enables easy network re-programmability for more elastic response to unexpected events.

Using Linux containers enables MystifY to exploit the intrinsic resource isolation offered by the technology to isolate the running controllers from the underlying host to limit host-based attacks. There are many other container Linux-based technologies that are available, such as "LXC" [56] and "OpenVZ" [57]. However, we managed to select "Docker" due to its flexibility, cost-effectiveness with fast deployment.

Before migration, we assume that the controller and all the running applications are encapsulated in one of the MystifY managed "Docker" containers. Mystify migrator module relies on "RunC" [58] as an integrated export tool to allow the container files to run independently from the Docker-management demon service. Running the container in such way, enabled MystifY to execute the container in an unprivileged mode in the user space for easier management, migration, replication, and resurrection.

The migration process involves two main processes; a checkpoint of the running container with the enclosed controller that would result in a controller stop, and a restore processes. The restore process re-instantiate the container to the exact same state at check point in a new location.

To enable live checkpointing, MystifY relies on "CRIU" tool [59] to briefly freeze the live container to enable snapshotting the container memory and any open files. The dumped snapshots are saved with a unique time stamp for restoration.

(A) Live checkpointing/restore and host-to-host migration

Researchers noted that SDN controller stores the network state in the controller internal memory to ensure high response rate. Keeping such vital information locally complicated network state sharing or external cloning in case of failure. The recent advances in SDN controllers tried to enable controller state sharing through external shared storages, by manipulating the way it is saved by customizing its source code [60]. However, such attempts limit the controller response massively and open the doors for contentions.

To avoid controller customization MystifY uses the encapsulated controller state and "CRIU" to dump the container memory into persistent set of shareable files only at the migration time with no impact on the controller operation.

MystifY migrator module manages "CRIU" to briefly halt the live (container) "RunC" process and its sub-processes (controller and/or applications); and save all used files and the allocated memory space to a share location. This location will be shared between both the source host and the destination host. As shown in Fig. 3, we will use these files later to restore the container to the required state.

In this paper, MystifY adopts PAFR migrator to enable the controller mobility between hosts and easily instantiate controllers from a cashed image. Such migration must be done without losing the application layer connection and the infrastructure network state. Therefore, some SDN controllers' instances must maintain same application access and system state information to fulfill the requirements of the SDN controller application and ensure state synchronization in Mystify' framework.

Algorithm 1: node-to-node average latency

```
Input: The network topology and its coordinates
Result: Average latency between nodes, optimal and alternative nodes
while change=1:5 do
    cc=1
    K_pos_iter4=[ ]
    top=[ ]
    L_avg_ic4=[ ]
    DDT=[ ]
while i1 eq 1:n do
    while i2 eq i1+1:n do
        while i3 eq i2+1:n do
            while i4 eq i3+1:n do
                if (i1 eq POS_K_AVG) and (i2 eq POS_K_AVG) and (i3 eq POS_K_AVG) and (i4 eq POS_K_AVG)
                then
                    Tx=Td
                    top=[Tx(i1,i2),Tx(i1,i3),Tx(i1,i4),Tx(i2,i3),Tx(i2,i4),Tx(i3,i4)]
                    Tx([i1,i2,i3,i4],[i1,i2,i3,i4])=0
                    DDT=[Tx(i1,find(Tx(i1,:)));Tx(i2,find(Tx(i2,:)));Tx(i3,find(Tx(i3,:)));Tx(i4,find(Tx(i4,:)))]
                    DD=min(DDT)
                    K_pos_iter4(cc,:)=[i1 i2 i3 i4]
                    L_avg_ic4(cc)=sum(temp)
                    cc=cc+1
                else
    [Lavg_IC(k) pos1IC]=min(L_avg_ic4)
    Lavg_IC(k)=Lavg_IC(k)/4
    pos_k4_LavgIC=K_pos_iter4(pos1IC,:)
    POS_k_avg_IC(k,:)=pos_k4_LavgIC
    k=k+1
```

Algorithm 2: node-to-node worst latency

```
Input: The network topology and its coordinates
Result: Worst latency between nodes, optimal and alternative nodes
while change=1:5 do
    cc=1
    K_pos_iter4=[ ]
    top=[ ]
    L_max_ic4=[ ]
    DDT=[ ]
while i1 eq 1:n do
    while i2 eq i1+1:n do
        while i3 eq i2+1:n do
            while i4 eq i3+1:n do
                if (i1 eq POS_K_MAX) and (i2 eq POS_K_MAX) and (i3 eq POS_K_MAX) and (i4 eq POS_K_MAX)
                then
                    Tx=Td
                    top=[Tx(i1,i2),Tx(i1,i3),Tx(i1,i4),Tx(i2,i3),Tx(i2,i4),Tx(i3,i4)]
                    Tx([i1,i2,i3,i4],[i1,i2,i3,i4])=0
                    DDT=[Tx(i1,find(Tx(i1,:)));Tx(i2,find(Tx(i2,:)));Tx(i3,find(Tx(i3,:)));Tx(i4,find(Tx(i4,:)))]
                    DD=max(DDT)
                    K_pos_iter4(cc,:)=[i1 i2 i3 i4]
                    L_max_ic4(cc)=max(temp)
                    cc=cc+1
                else
    [Lmax_IC(k) pos1IC]=max(L_max_ic4)
    pos_k4_LavgIC=K_pos_iter4(pos1IC,:)
    mm=L_max_ic4(end:-1:1)
    POS_k_max_IC(k,:)=pos_k4_LmaxIC
    k=k+1
```

To achieve these goals, and to make sure that all the data is persistent and available to all the SDN containers' images, we bind the Docker volumes to a location on each container's disk. These volumes are mounted on a remote repository to host PYGRID IDS application (an example for a security application to protect the SDN controllers running at the infrastructure layer, this application will be explained in Section 5.1). Hence the application folder is simultaneously available for all the SDN instances. In addition, such volumes will host the encapsulated controller state and "CRIU" dump memory to maintain system state synchronization.

(B) The restoration process

To lunch a new controller or to re-instantiate a checkpointed one, we need to have access to two sets of files, the memory dump and the offline container image incase or re-instantiation. The memory dump are than 20 MB which enables a very fast instantiation, quick recovery, and easy container migration. Whilst, the offline container image files are large, about 500MB in average in our experiments with OpenDayLight [61], POX [62], or Floodlight [39] SDN controllers. Fortunately, those files are transferred offline with no impact on the migration process.

MystifY uses a remote shared repository for all the controller images used by the system accessible to all potential heterogeneous hosting locations. Diversifying the hosting locations and their configuration limits the attacker capabilities for attack reply or exploiting/financing the same weakness between hosting servers. MystifY only transfers the memory dump image between hosts at the time of migration. Using shared repositories to host the base image ready containers, massively lessens the time required to transfer all the files between hosts in

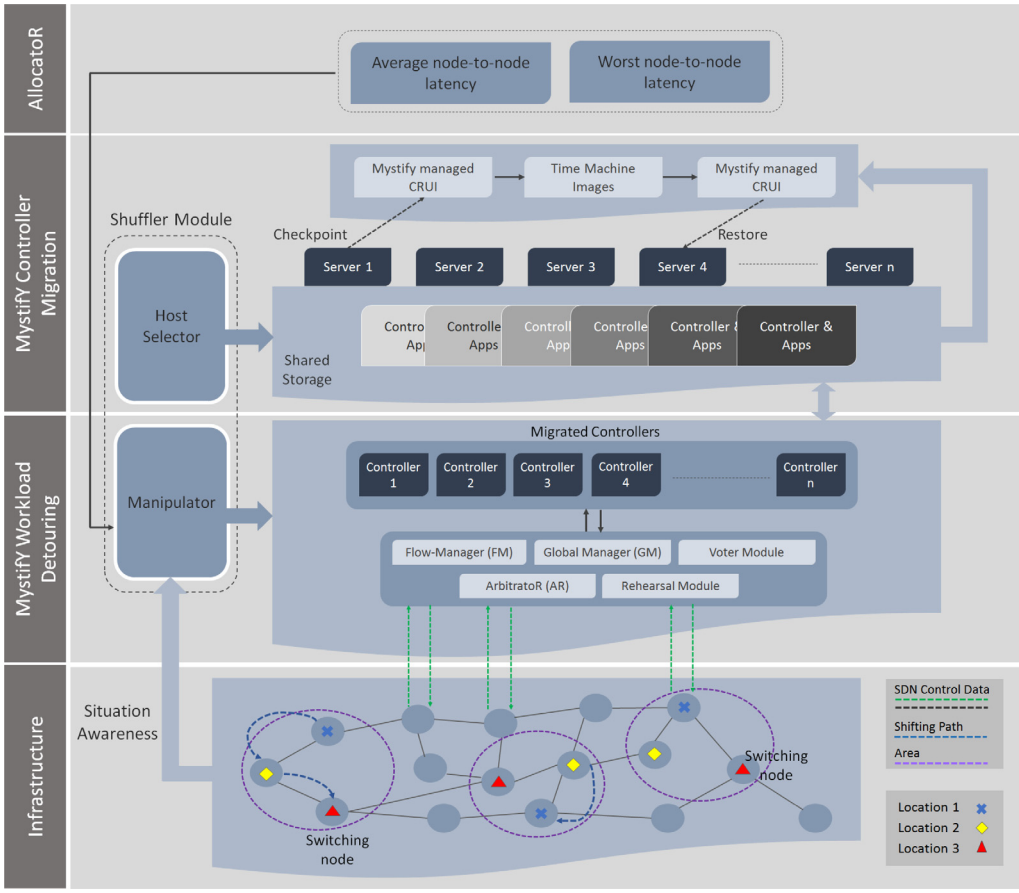

Fig. 3. MystifY detailed system architecture.

case of failure or live migration. The entire process is summarized in Fig. 3.

MystifY can apply random or preprogrammed checkpoints, or upon certain event triggers. This process starts by halting the container, exporting its memory contents into set of files on the shared location, marked with a unique timestamp. To ensure the consistency of the network status, MystifY usually uses the most recent image for recovery. However, archiving old images can be very useful for the forensic analysis after any unexpected failure/ attacks.

Our experiments resulted an approximate downtime of "~0.15" seconds during the unavailability-of- SDN controller service. This time can be negligible as is assumed to be a network congestion by all attached switches. In OpenFlow protocol [63], controllers have to react to the incoming echo messages from attached switches as a sign of aliveness. Any intubation in such responses would result in the switches to terminate the connection. During our experiments, the migration procedure completed before the echo messages times out. In addition, all attached switches did not report any controller connection interruptions during the entire migration procedure.

The experiments of testing the effect of migrating a running SDN controller between two preconfigured hosts were run 7 times, and the average downtime due to migration process was about "~0.15" seconds. The actual approximate values were; t_{d1} = {0.12965}, t_{d2} = {0.16548}, t_{d3} = {0.14042}, t_{d4} = {0.15822}, t_{d5} = {0.15063}, t_{d6} = {0.13478}, t_{d7} = {0.15718}, where "t_d" is the migration downtime at each test. It is the time that the controller service was unavailable to the network testbed. This time was measured during the period when any of the network switches was unable to reach the SDN controller on TCP port "6653".

The migration process starts by checkpointing the container, killing the process on the source location, initiating an ARP update to change the IP/MAC assignment of the source server's network interface to match the new one, updating the network address translation tables, and finally restoring the encapsulated controller files to the destination location.

4.3. The proxy module, MystifY-managed Repoxy

MystifY adopts and customizes our novel SDN proxy, "Repoxy" [9], to enable multiple controller management of same switch. Further, MystifY elevates proxy's feature to enable detecting misbehaving SDN controllers. In this paper, MystifY's proxy is a distributed abstraction layer isolating the active controllers from the underlying switches. Such isolation enabled seamless controller host switching when MystifY used the migrator module to migrate the active controller between heterogeneous hosts for spatial diversity induction.

In this section, we will illustrate how MystifY manages and customizes "Repoxy" to enable dynamic workload migration between replicated controllers for enhanced resilience. With such replication, MystifY realizes the temporal diversification of running controllers for MTD and enhances the controller resilience against intrusions as illustrated latter.

Additionally, MystifY-managed proxy to enable dynamic alternation of the management role between the running controllers by migrating the workload between these controller replicas based on the shuffling module feedback as described in subsection D. MystifY's proxy ensures the consistency of the network state between replicated controllers. Further, MystifY's proxy, uses an offline database and a customized

rehearsal module to maintain an archived copy of controller switch interaction enabling offline state synchronization between newly ported controllers. Such added feature enables a dynamic porting and departure of SDN controllers from the active controller pool based on the network state.

At each packet arrival, the "Voter" module (described in next subsection) selects the most appropriate controller to serve the network, and manage its distributed switches. The operational details of each module are illustrated later in next subsections.

(A) Protocol messages

The OpenFlow protocol is the communication protocol widely used to define the switch controller interaction. In this paper, we built MystifY to work mainly with OpenFlow protocol. Other protocol support can be included in the future. The SDN controller main rule is to define the flow entries in the attached switches. The specification classify the network packets between switches and SDN controller to three main types, controller-to-switches, symmetric, and asynchronous.

The first is the most important type, the controller-to-switches packets. In these packets, controllers manipulate the attached switches with a packet and expects a "Reply" packet as a response with the same X_{id}. The X_{id} is a unique identifier sued for packet identification by the SDN controllers and switches.

The second type is the asynchronous packet. These packets are sent spontaneously by the switches when requesting for controller guidance.

The third is the symmetric packets. It describes the kind of packets sent periodically or on the beginning of a controller/switch attachment (such as the "Hello" messages).

(B) MystifY-managed Repoxy

MystifY's proxy intercepts the switch packets, processes them through all connected controllers, and resends the response from the selected controller while discarding the rest after validation. This process involved three main players, the Flow-Manager (FM), ArbitratoR (AR), and the Global-Manager (GM), as shown in Fig. 2. The FM collects the incoming packets, passes it to the AR. The AR organizes the packets from the different sources whether it arrives from a switch or one of the controllers. The AR sends the packet to the GM for validation and voting.

• The Flow-Manager (FM) module

The FM module handles the connections and data exchange between the proxy's attached switches and the replicated controllers. As we are relying OpenFlow packets, the FM uses TCP connections for data exchange. Upon a new switch connection, the FM assigns it a unique connection ID as a unique network address managed by the proxy. This module is also responsible for the connection state management. It reports to the AR any switch detachments or disconnections. In addition, it handles the packet re-fabrication to maintain switch/controller expectation. Incoming packets should be coming from the expected party. As mentioned before, the proxy is a seamless layer, then all switches treat MystifY's proxy as a unique managing controller.

• The ArbitratoR (AR) module

The AR module's main task is to handle incoming packet flow from the FM module. Upon reception of a new packet, the AR module adds it to the GM's input queue pipeline. The AR module also handles packet buffering and correction as discussed latter by the end of this section.

• The Global-Manager (GM) module

The GM module is an elastic component built for innovation. The GM module encompasses many sub-modules that operate on the incoming packets. The current implementation includes the "Voter" module, which decides which controller's feedback to be forwarded, and the "Rehearsal" module which handles the new controller porting. More sub-modules can be easily ported to the GM module for extended functionalities.

The GM module encompasses a secure database for the auditing and forensic analysis. It stores short term logs for packet mismatch events or any errors in the packet queue. The GM module uses packet arrival time to detect mismatches. Upon packet arrival, the AR places it in the queue for further processing, while packets in the waiting queue are checked for timeouts.

(C) Organized replicated controller response

Given the fact that we are working with multiple controllers with heterogeneous operational and location characteristics, their response to the incoming packets would require explicit synchronization and ordering. The ordering process uses the X_{id} as a guideline (a transaction ID). The X_{id} is an identifier for all OpenFlow packets. Each controller generates his own X_{id} field for all outgoing packets. Using more than one controller world require the proxy to synchronize these values to facilitate packet matching for further processing. The reply packets for any switch request should contain the correct X_{id} value in the packets header. Reply packets must match the X_{id} value of the request packets. If a switch/controller receives a packet with an invalid/unexpected X_{id}, it ends the connection immediately. Furthermore, for messages that use the same identifier value for all packets in any message sequence, MystifY's proxy uses the packet types to synchronize all X_{id}. It is a multistage process with multiple input/output queues. As mentioned before, we have three main packet types, the controller-to-switches, symmetric, and asynchronous. The last two types would not result in a X_{id} synchronization issue. They are independent from any other packets. MystifY's proxy directly forwards these packets to the destination with no change. Therefore, the first type is always our main concern.

• Packet ordering

As we are working with multiple controllers, we need to synchronize the packet transmission and the delivery schedule. We do not want one controller to receive a reply for a message not yet sent by him. This can occur if his clone running on a faster machine sent this message to the switches before, and the reply was the switch's response.

In this case, we consider the worst-case scenario that allow request or reply packets of the same type to arrive spontaneously at any time. The proxy waits for packet arrival, and then it scans the entire input queue searching for a reply packet for this particular request. If found, then it was sent as a response to a previous request. The proxy sends this request packet to an output queue. If not found, then it is sent directly to the switch with a copy staying in the output queue. Upon the reception of a reply, it is stored in the input queue to be checked for a matching request. If found, it is removed from the queue and forwarded to the destination. If not, it waits for a match. The matching process is illustrated in the next subsection.

• X_{id} Matching Process

The ordered requests and replies must be matched for further processing. The proxy searches for matching request/reply packets based on the type, then the proxy corrects the X_{id} accordingly to cause a match.

Let us assume a "Barrier Request" packet is received after a "Feature Request" packet. Then, a "Feature Reply" packet came followed by a "Barrier Reply" packet. As all these packets are there with no missing packets, the proxy matches the replies, and corrects the X_{id} identifier of such packets. After this packet ordering, the X_{id}-Synchronizer module takes place. Then, a copy of any input request is stored in a local hash table waiting for a matching reply, to modify its X_{id} identifier as the input request X_{id} identifier waiting in that table.

• Packet Distribution

Once the matching process completes, the X_{id}-Synchronizer module output queue should contain all the packets from all controllers and switches. The wrapper module takes these packets, separates them by based on the sender type, and send them to the appropriate destinations.

(D) The voter and verifier modules "detecting misbehaviors"

At this stage of the implementation, the "Voter" module is built to follow the FIFO concept where responses from the 1st responding controller wins. Considering that, all controllers' instances are clones from each other, differences between these controllers' state/configuration can be declared as a misbehavior or anomaly situation.

Upon arrival of a new packet, the "Voter" module inspects the incoming packet. All echo packets are handled by the proxy to maintain switch controller aliveness status. Control request packets from connected switches are forwarded to all attached replicas. The "Voter" module waits for at least two responses, and waits for the verifier module to clear the response. The verifier module compares the responses searching for discrepancies as an indication of an intrusion. If no discrepancies, the 1st responder feedback is sent to the requester switch. The rest of the unchecked feedback is sent to the verifier module to further search for potential misbehaving controllers then discarded if no red flags are raised.

If any discrepancy is detected, the controller responsible, is to be revoked from the pool and a report is to be sent to the administrator. MystifY uses the shared repository to re-instantiate a new controller to replace such misbehaving controller. The "Rehearsal" module will bring that newly ported controller "up-to-speed" with the network state, and then it will be allowed to connect to MystifY's proxy and act as a replica.

(E) The rehearsal module

This module handles the new controller porting process. Once a new controller joins the network, MystifY needs to ensure a consistent network state before allowing the controller to participate in network management. This module is invoked rarely as it is much easier to clone one of the running controllers. In this process, a containerized controller is paused momentarily and the container is cloned with all network state in memory.

In the case of all failed controller, or a desperate need for a fresh controller introduction like the case of adding a new type of controllers, the "Rehearsal" module is invoked. This module will allow the new added controller to join the network in a passive mode. MystifY uses the archived switch traffic communication to reply these messages to the new controllers. In the current implementation, the "Reversal" module is in a very early alpha release. Replying the switch feedback is not a straightforward process. Further development will be conducting in our sequel versions. For now, we use replicated controllers to increase the controller pool adding more controllers to ensure safe and reliable MystifY operation.

4.4. The shuffler module

This is a very critical component in the proposed framework. The "Shuffler" module is composed of two main components, the "Host Selector" and the "Manipulator".

First, the "Host Selector" which selects the list of controller migration locations for MystifY migration module to operate on. The "Host Selector" module follows the same protocol mainly to selects the migrating locations based on the "nodes-to-controller" latency calculated by the "AllocatoR". The goal is to minimize any added delays due to migrations and to ensure the delay homogeneity for the workload detouring.

Second, the "Manipulator" which is responsible for orchestrating the workload migration pattern based on the real-time feedback from the CPS application provisioned through the situation awareness module.

The situation awareness module feeds the "Manipulator" module with real-time feedback about the grid message priority. In the current implementation, the situation awareness module is a simple program allowing the system administrator to define the delay tolerance for the communicating pairs within the network classified into a set of categories. The components IP "identifier", class, and acceptable delay are stored in a secure database accessible to the "Manipulator" module. The system checks the incoming packet content header to determine the category of the sending source. The "Manipulator" module the database to access the list of preconfigured classified categories defining the latency tolerance for the communicating smart grid equipment.

The situation awareness module dynamically adjusts this tolerance. The current implementation uses a simple database-based classification approach. In our sequel papers, we will use Machine Learning algorithms to automatically categorize the incoming packets based on the content. In the current implementation, upon the reception of any "Packet-in" messages with a header showing the communicating host IP, the manipulator-module checks the IP against its local database to determine the sender category and the allowable latency tolerance. Each controller provided by the "AllocatoR" module comes with a statistical information about the delay involved with its operation, "node-to-node" and "controller-to-controller". The "Manipulator" module priorities the feedback based on the "controller-to-controller" delay not to exceed the allowed delay tolerance by the transmitting equipment class.

5. Simulation results and analysis

Smart Grid is one of the most crucial CPS application that relies on gathering and acting on real-time information in large-scale networks to serve millions of citizens every day. Therefore, we chose this network to evaluate our proposed MystifY framework. For a comprehensive evaluation, we deeply analyzed the system response from different points-of-view, a performance-related ones, and a security-related ones.

1. Evaluate the downtime due to SDN controller migration.
2. Evaluate the down-time due to workload detouring.
3. The effect of optimal placements of SDN controller(s) within the network.
4. Evaluate the attacker's ability to trace the operational SDN controller within the network.

5.1. Evaluating the controller migration impact on the grid operational aspects

In order to evaluate the first two case perspectives, MystifY relies on PYGRID framework to simulate an SDN-enabled Smart Grid. PYGRID was introduced as a complete software development and assessment framework for grid-aware software defined networking based on Python [64]. Additionally, it uses various Python scripts to construct grid traffic and routes between the emulated power components.

In this paper, our simulation scenarios are conducted using IEEE 24-Bus power system to demonstrate the effectiveness and potent of the whole proposed MystifY solution. PYGRID simulator analogizes the IEEE 24-Bus system into a total of twenty-four emulated virtual hosts.

Fig. 4 illustrates the overall system architecture porting of PAFR, Repoxy, and PYGRID frameworks. It addition, it shows the represented twenty-four emulated virtual host, its connections, and the detailed configuration of TCP/IP layers. Our simulation scenario uses eight "Docker" stations to represent controller migration tier. Each "Docker" station hosts an SDN container with the following specifications; Ubuntu 14.04, Mininet 2.1.0, and POX 0.1.0 SDN controller.

Further, the new emulated IEEE 24-Bus network is splitted into four areas, each area is managed by a single virtual switch. Therefore, MystifY's proxy is the middle tier of the new architecture model detouring workload between four SDN controllers and connecting them to the infrastructure emulated hosts. MystifY's proxy handler is configured to listen on TCP port "6633" to accept the control data from down-layered virtual switches.

(A) Evaluating the down-time due to SDN controller mobility

The main goal of this scenario is to demonstrate the migration down time impact on the grid. In this scenario, we managed to migrate one of the selected controllers "controller ID 3, 192.168.100.3" to "Docker"

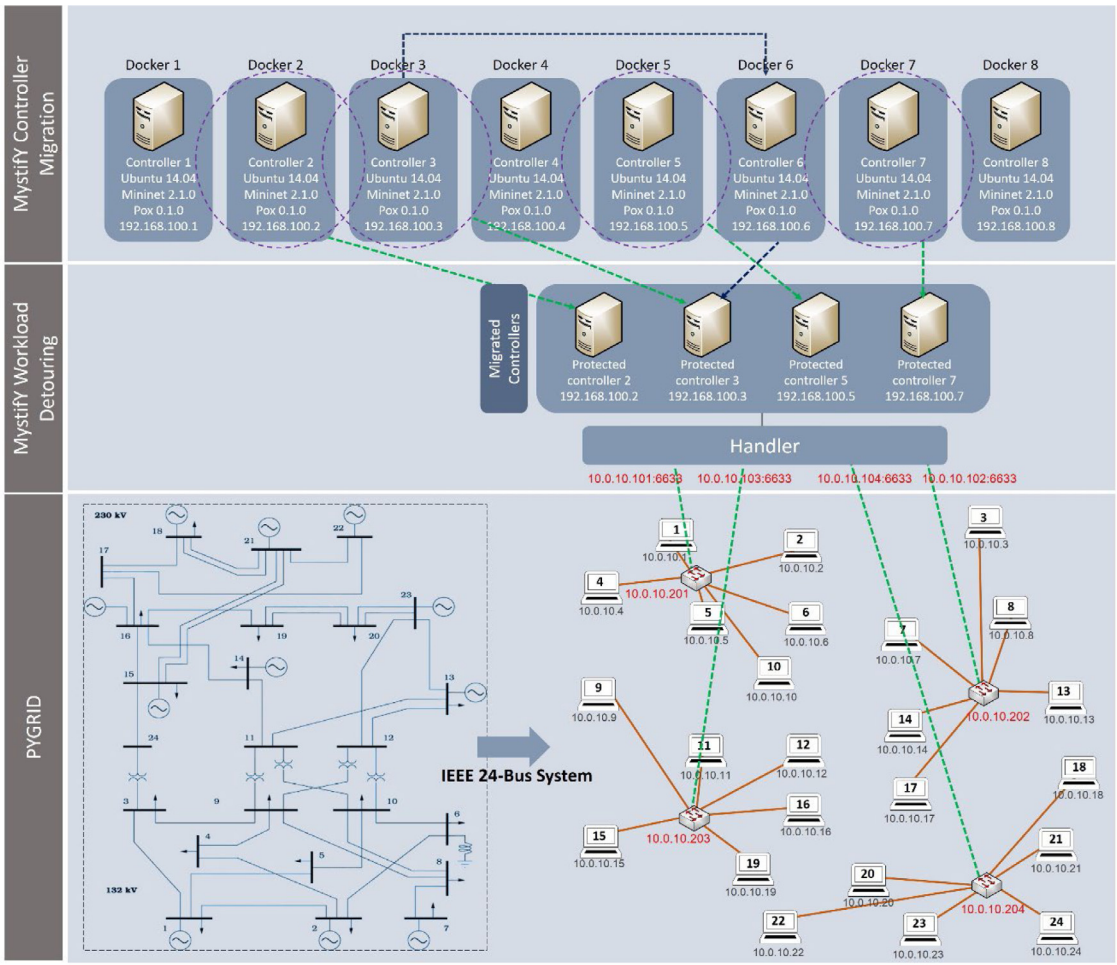

Fig. 4. PYGRID simulation scenario.

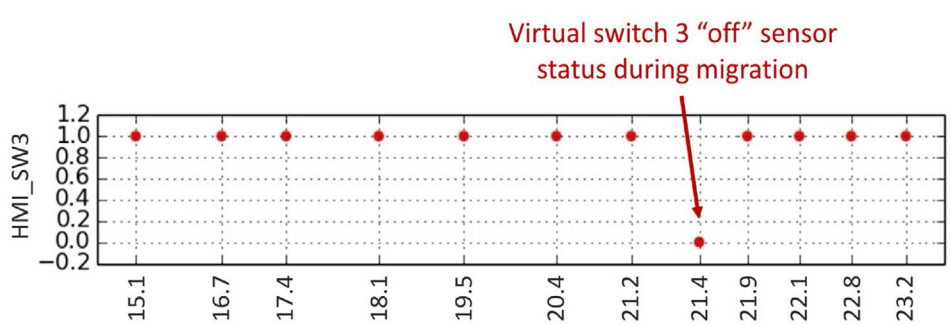

Fig. 5. HMI for virtual switch 3 operational status.

station number "6". As shown in Fig. 4, for about "0.5 s", virtual switch 3 was operationally down. Once the migration process is completed, MystifY managed to bring the controller online again and virtual switch "3" was a live again. Fig. 5 demonstrates the HMI interface for this switch status during the migration process.

Additionally, Fig. 6 illustrates virtual switch "3" OpenFlow received traffic during the experiment duration time. Fig. 7 represents the stored power traffic in PYGRID SQL database in real-time. The resulted PYGRID data matches IEEE 24-Bus standard specification data shown Table 1. These data emphasize our contribution, MystifY managed to bring a new SDN controller to the PYGRID network again in real time. Then, PYGRID called the appropriate script and adopted the network again to the standard specifications.

(B) Evaluate the down-time due to workload detouring

In order to compensate the down time due to controller live migration, we used the MystifY-managed proxy's ability to detour the workload between running controllers to avoid using the migrating controller during migration time. Fig. 8, showed that with MystifY integrated framework, we managed to achieve a zero downtime for controller migration resulting in enhanced security with no impact on the performance.

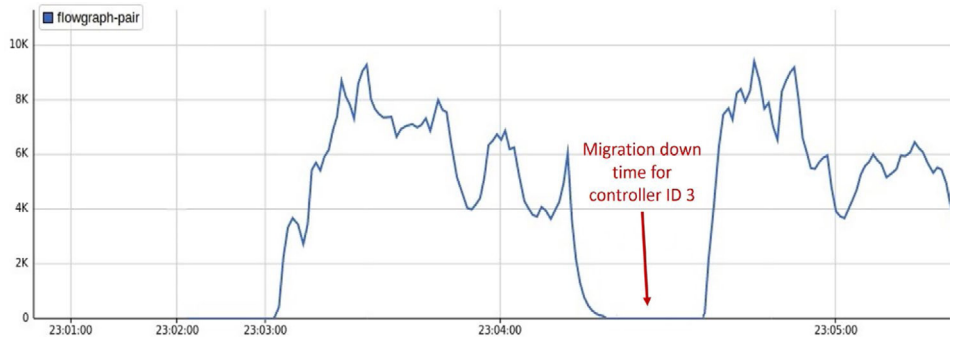

Fig. 6. Virtual switch 3 control data traffic.

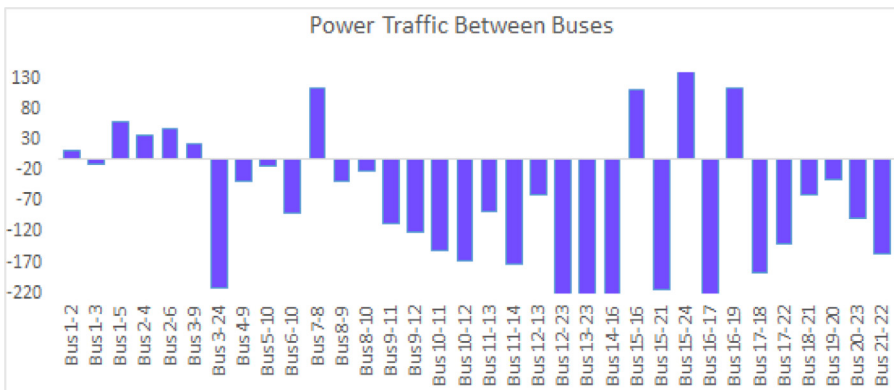

Fig. 7. HMI power traffic distribution.

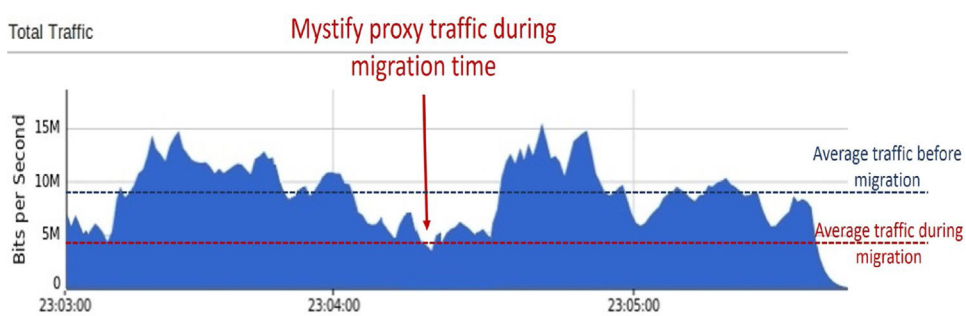

Fig. 8. MystifY proxy module evaluation.

Table 1
EEE 24-Bus power data specifications.

Branch	Power (MW)	Branch	Power (MW)
Bus 1–2	11.94	Bus 11–13	−86.15
Bus 1–3	−7.97	Bus 11–14	−171.77
Bus 1–5	60.03	Bus 12–13	−60.51
Bus 2–4	38.44	Bus 12–23	−227.70
Bus 2–6	48.50	Bus 13–23	−225.30
Bus 3–9	22.90	Bus 14–16	−367.55
Bus 3–24	−211.21	Bus 15–16	112.30
Bus 4–9	−36.15	Bus 15–21	−214.92
Bus 5–10	−11.71	Bus 15–24	215.54
Bus 6–10	−88.59	Bus 16–17	−322.68
Bus 7–8	115.00	Bus 16–19	115.08
Bus 8–9	−36.92	Bus 17–18	−186.94
Bus 8–10	−21.19	Bus 17–22	−139.09
Bus 9–11	−105.92	Bus 18–21	−60.29
Bus 9–12	−120.47	Bus 19–20	−33.17
Bus 10–11	−151.18	Bus 20–23	−97.29
Bus 10–12	−166.74	Bus 21–22	−156.46

Table 2
Sample data for three GSP nodes.

Substation	Network Ref. ID	Substation number	Latitude	Longitude
Aberthaw power station	245414	510 049	51.3882	−3.4026
Abham S.G.P.	136	306 004	50.4718	−3.7319
Alverdiscott S.G.P.	132	206 001	51.0059	−4.1371

5.2. Evaluate the effect of optimal placements of SDN controllers on the network operational aspects

In this paper, we exploited the Western Power Distribution (WPD), UK Grid Supply Points (GSP) network capacity map [65] to simulate a simple real mini-grid network topology, as shown in Fig. 9. The selected topology map is analogized as a forty-eight nodes representing all substations. It is important to note that, we assumed the connections between these nodes according to the network capacity and coordinates. Table 2 shows a sample data for three GSP nodes

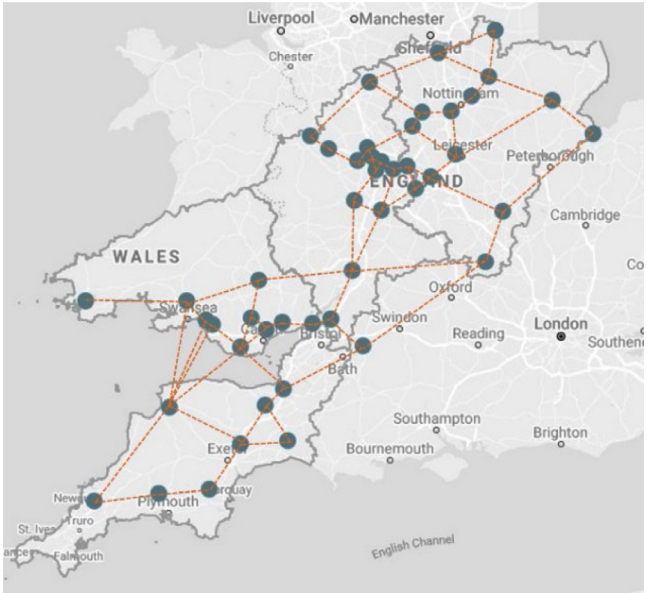

Fig. 9. UK GSP network capacity map.

Table 3
IEEE1646 standard service characteristics.

Speed	Application	Latency
Very high	Streaming VT and CT samples	<2 ms
High	Event notification	2~10 ms
Medium	Non-critical information	10~100 ms
Low	External message delivery	>100 ms

Further, in the energy industry, IEEE1646 international standard [66] defines the standard communication delivery time and performance requirements for the electric power substation automation process. Table 3 presents the standard industrial measurements and latency values for IEEE1646. Therefore, it is very important to get the magnitude of the CPP solution results in relevance to such characteristics, with help of the "Manipulator" module.

Consequently, in this paper, the "AllocatoR" Module is developed to import our grid topology and compute the performance latency metrics. Hence, it determines the best locations to host the SDN controller(s) (optimal and alternative locations) according to the CPP calculations results. In this phase, we rely on "Haversine" formula [67] to calculate the geometric distances between the network nodes.

In addition, we managed to use the "random-to-optimal" ratio technique in order to determine the optimal number of SDN controller(s) required to serve the network giving the best latency results. For example, in case of an average latency between "nodes-to-controller", as shown in Fig. 10(a), five distributed SDN controller(s) are enough to serve the network needs, and there will be no much latency improvement when adding any other of SDN controller(s) to the network. However, in case of "controller-to-controller", as shown in Fig. 10(b), four distributed SDN controller(s) are enough to serve the network needs in this case.

As a conclusion, the best number of K controllers are either four or five in the studied grid. However, to avoid long simulation time, we intended to use $K = 4$ as the best number of controllers used by MystifY modules. Table 4, shows the calculated latency values for each of the studied metrics in case of selecting only four SDN controllers.

In this paper, we calculated the performance latency metrics by determining the distances in "Km" between the selected controllers and the surrounding nodes in the network grid. Knowing the direct relation between the latency in "Sec" and the distance in "Km" between

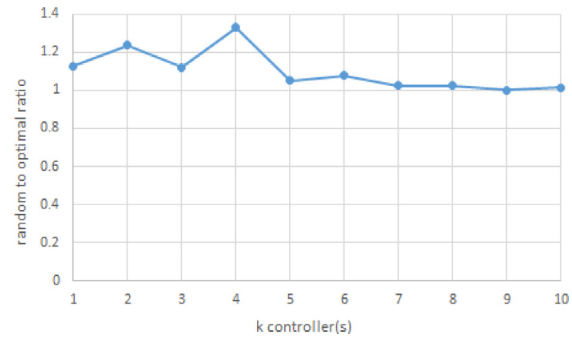

Fig. 10(a). Random-to-optimal ratio for average latency "nodes-to-controller".

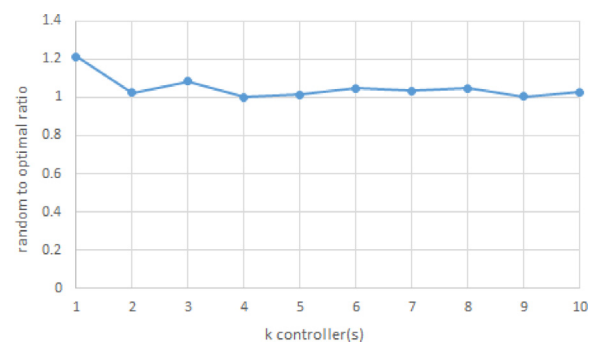

Fig. 10(b). Random-to-optimal ratio for worst latency "controller-to-controller".

Table 4
Calculated latencies for the selected controllers in the studied metrics (Km)

Metric	Optimal selection	Alt. 1	Alt. 2	Alt. 3	Alt. 4
Latency between "nodes-to-controller"					
Average latency	30.2603	30.6971	32.3708	34.1350	35.7861
Worst latency	71.4203	73.5955	75.1903	76.6583	97.0680
Latency between "controller-to-controller"					
Average latency	30.2603	30.6971	32.3708	34.1350	35.7861
Worst latency	200.7620	240.908	251.1744	297.2675	340.4054

the nodes, we could calculate the latency according to the used transmission media. Therefore, we used the distance as a latency indicator through the rest of the paper.

In order to evaluate the selection of the alternative locations for each SDN controller at certain area, the Cumulative Distribution Function (CDF) studies are done for all applied CPP metrics. As shown in Fig. 11, the CDF demonstrates all the available placements for each selected controller at certain area. The curves show that, all possible locations for the selected controllers are placed at the lower end. In addition, it demonstrates that how the selection of alternative placements far from the optimal ones will influence the system's performance and increase the traffic latency. In case of "nodes-to-controller" latency, the variation occurred within the CDF curves seems to be minuscule, as depicted in Figs. 11(a), and 11(b). The reason beyond that, is due to selecting alternative controllers relatively close to those selected in the optimal case, resulting in slight variations in the latency values.

However, in case of the latency calculations in the other two metrics for the "controller-to-controller" latency case, a significant change occurs in the calculated CDF curves, as depicted in Figs. 11(c) and 11(d). The huge shift found in the CDF curves is due to the fact that the distance between the selected controllers increased significantly after excluding the selected ones from the pool, leading to a huge increase in the calculated latencies between them.

In addition, In order to determine the alternative controller locations for runtime migration, we repeated the same calculations again for the studied four metrics excluding the previously selected ones from

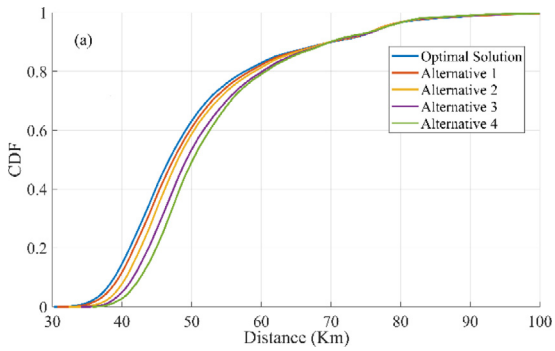

Fig. 11(a). CDF vs. distance for "nodes-to-controller" average latency.

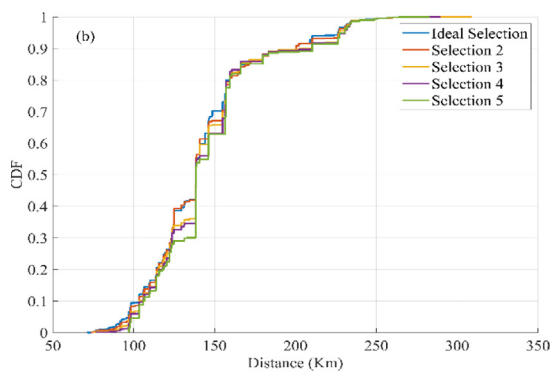

Fig. 11(b). CDF vs. distance for the "nodes-to-controller" worst latency.

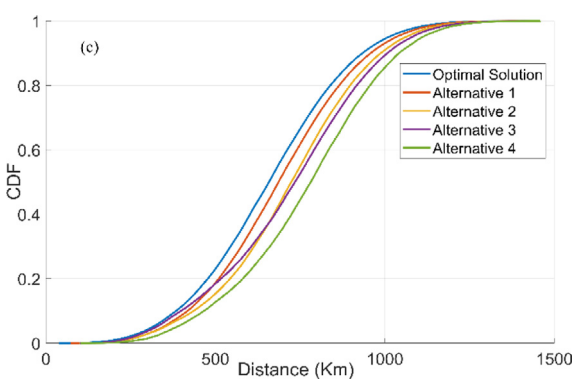

Fig. 11(c). CDF vs. distance for the "controller-to-controller" average latency.

the potential selection pool. In order to expand the shuffler's abilities, we determined extra four alternative controllers for random shuffling. Table 4, shows the calculated latencies for the selected extra alternative controllers for each of the four studied latency metrics.

Despite the performance reduction occurred due to exploiting other controllers aside the optimal ones, they will be exploited in the proposed system. Further, it worth noting that there is a tradeoff between the system's performance and its security. Therefore, it is important to use only controllers with the optimal locations from the perspective of the studied metrics. The next section briefly demonstrates the system's security evaluation after applying shuffling technique between the selected controllers is described briefly.

5.3. Evaluating the attacker's ability to trace the active SDN controllers

Using the "AllocatoR" module selections, the "Shuffler" module uses the selected alternative controllers to begin the shuffling effect.

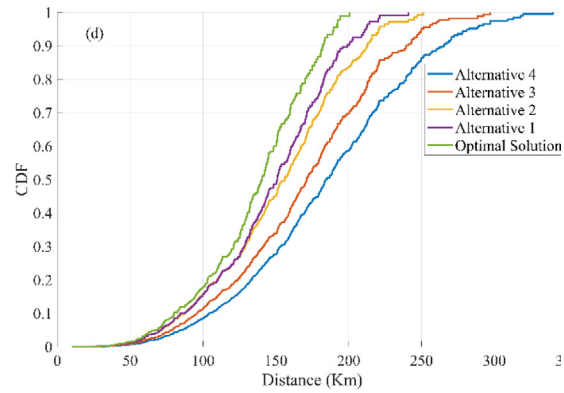

Fig. 11(d). CDF vs. distance for "controller-to-controller" worst latency.

We anticipate that applying a random shuffling mechanism will induce enough confusion for the attacker complicating the controller operations to be traced.

Our assumption is that attackers will take a considerable amount of time to allocate the SDN controller, search for vulnerabilities, and to coordinate the attack tools targeting specific host or controller vulnerability. The goal is to win the race by migrating the controllers or the workload before discovering them by the attacker.

In this section, we defined a new term called the Exposure Factor (EF) as an indicator for the system's security. This factor reflects the progress of the attacker exposing the system changes. We described the EF term as the ratio between the simultaneous changes of the network's selected controller versus the discovered ones by the attacker.

In Section 3, we assumed that, after a while the attacker might be able to access one or more controllers and manage to manipulate their behavior. Furthermore, we argued that it might take MystifY, a while to realize the controller misbehavior. As demonstrated before, the "Shuffler" module selects random controllers from the controller's selection poll and start shuffling between them randomly. Therefore, the system alternates between the network controllers at different time stamps.

To determine the EF, From MystifY's perspective, according to the "AllocatoR" module selections (determined in Section 5.2), the "Shuffler" applies a random shuffling mechanism to use the selected alternative controllers. To calculate/measure it, the alternation between the selected controllers was tracked, logged, and accumulated during the simulation.

The shuffler is triggered once we realize that the controller misbehavior/malfunction. For example, the experimental testbed in Section 5.1, shows that switch 3 was down, and hence its connected live controller was down. We managed to run this scenario randomly to the 4 connected switches. In addition, a simple type of DoS attack (explained in Ref. [8]) is applied to PYGRID emulated network (Mystify down layer) to target any of these switches. Then, we monitored and tracked the running controller during the simulation time. Therefore, MystifY framework is able to detect the DoS attack, or any kind of attack targets the host controller itself.

Fig. 12 shows the obtained EF change over time for each of the studied metrics alone. Although, the "Shuffler" module selects random controllers at each time stamp, the attacker could successfully predict multiple controllers from them. However, the continuous shuffling immures the attacker abilities to predict the selected controllers and manipulate the system.

Moreover, we demonstrated that the system could be more complicated and increase the attacker's confusion, when the shuffling mechanism becomes more system aware. Increasing its awareness can occur by determining the possible metrics that could be used during sending the data based on the maximum latency threshold. This means that,

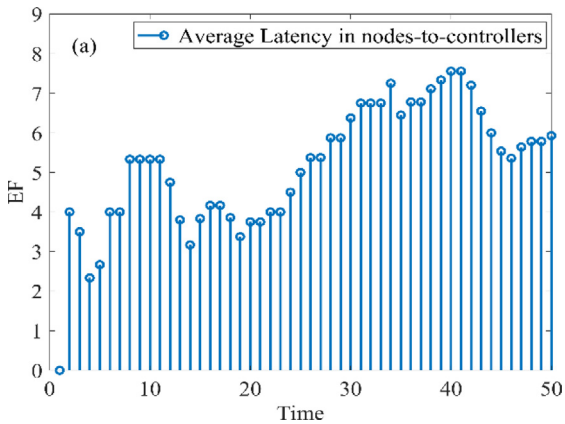

Fig. 12(a). EF over time targeting average latency metric for the "nodes-to-controller".

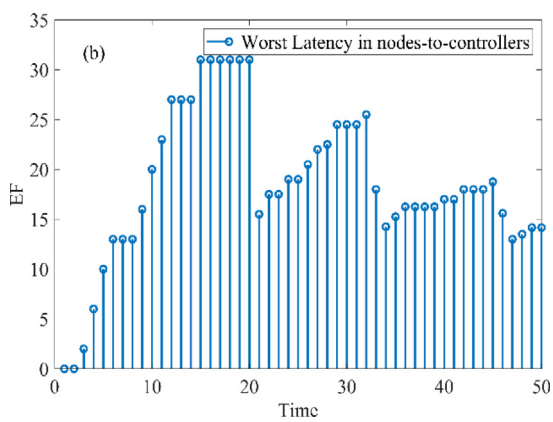

Fig. 12(b). EF over time targeting worst latency metric for the "nodes-to-controller".

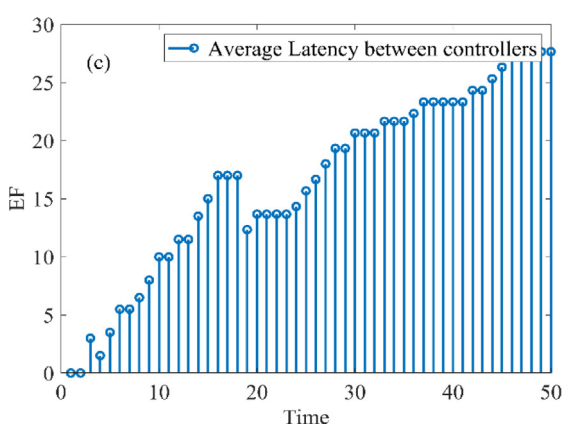

Fig. 12(c). EF over time targeting average latency metric for the "controller-to-controller".

based on the transmission latency threshold more than one metric could be used. Therefore, MystifY's "Shuffler" could have a multidimensional shuffling mechanism. Therefore, the "Shuffler" module could select the controllers from the averaged metrics instead of only one.

Fig. 13, demonstrates the EF after taking into consideration the multi-objective metrics. We assumed that the data supposed to be transmit, should maintain a latency value that does not exceed a "300 km" distance. Therefore, the system could select any controller(s) from the selection poll for any of the studied four metrics. As a consequence, the EF is significantly enhanced, as depicted in Fig. 12.

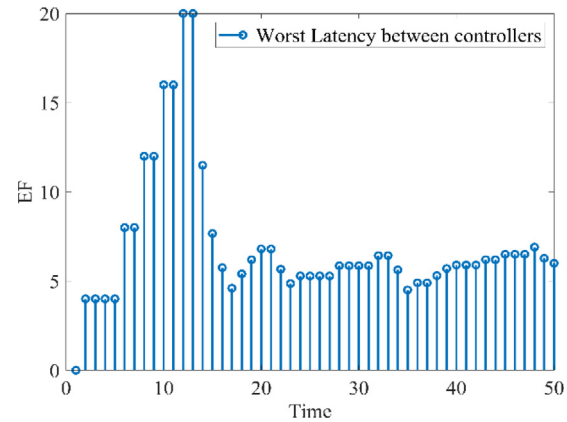

Fig. 12(d). EF over time targeting worst latency metric for "controller-and-controller".

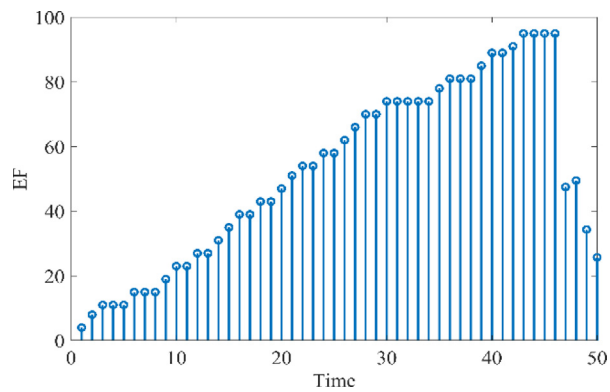

Fig. 13. EF over time for multi-objective metrics mechanism.

6. Conclusions

In this paper, we presented MystifY, a novel Moving-Target Defense (MTD) approach to secure the network infrastructure of mission critical Software Defined Cyber–Physical System (SD-CPS). MystifY secures the Software Defined Network (SDN) controller by inducing spatiotemporal diversification to obscure the controller operation from the attacker reach. In Space, MystifY enables a controller live migration to avoid host-based attacks. In time, MystifY enables a multi-controller management of same network set to achieve runtime workload detouring among multiple controllers, and control misbehavior detection. For evaluations, we exploited Smart Grids as a case study for a crucial application in CPS systems. We integrated MystifY with a novel SDN-enabled Smart Grid simulator (PYGRID) to investigate the effectiveness and efficiency of the proposed system. Our extensive results showed the positive influence of MystifY, securing the SD-CPS network infrastructure with no impact on the SD-CPS performance. Our future work includes employing Machine Learning algorithms to further enhance the situation awareness of the MystifY components of the internals of the targeted application.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is sponsored in part by the Commonwealth Cyber Initiative, USA, an investment in the advancement of cyber R&D, innovation,

and workforce development, Grant id HS-4Q21-005, and SWVA node ProCyEd, and other grants. For more information about CCI, visit www.cyberinitiative.org. Authors would like also to express their appreciation for the "IoT and Cyber Security lab", CIS, VMI, USA, for supporting and hosting the activities related to this manuscript.

References

[1] E. Byres, A. Ginter, J. Langill, How Stuxnet Spreads - A Study of Infection Paths in Best Practice Systems, White Paper, Tofino Security, 2011.
[2] A. Bhardwaj, V. Avasthi, S. Goundar, Cyber security attacks on robotic platforms, Netw. Secur. 2019 (10) (2019) 13–19.
[3] S. Mansfield-Devine, Ransomware: taking businesses hostage, Netw. Secur. 2016 (10) (2016) 8–17.
[4] N. Anand, S. Babu, B.S. Manoj, On detecting compromised controller in software defined networks, Comput. Netw. 137 (2018) 107–118.
[5] S. Dong, K. Abbas, R. Jain, A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments, IEEE Access 7 (2019) 80813–80828.
[6] Smyth, D. O'Shea, V. Cionca, S. McSweeney, Attacking distributed software-defined networks by leveraging network state consistency, Comput. Netw. 156 (2019) 9–19.
[7] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, ACM SIGCOMM Comput. Commun. Rev. (2012).
[8] M. Azab, J.A.B. Fortes, Towards proactive SDN-controller attack and failure resilience, in: 2017 Int. Conf. Comput. Netw. Commun. ICNC 2017, 2017, pp. 442–448.
[9] M. Azab, A. Hamdy, A. Mansour, Repoxy: Replication proxy for trustworthy SDN controller operation, in: Proc. - 17th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. Trust, 2018, pp. 55–60.
[10] Q. Jia, K. Sun, A. Stavrou, MOTAG:moving target defense against internet denial of service attacks, in: Proceedings of the 2013 IEEE 2013 22nd International Conference on Computer Communication and Networks ICCCN 2013, Bahamas, Caribbean, 2013.
[11] Q. Duan, E. Al-Shaer, H. Jafarian, Efficient random route mutation considering flow and network constraints, in: Proceedings of the 1st IEEE International Conference on Communications and Netw. Secur. (CNS '13), 2013, pp. 260–268.
[12] D. Kewley, R. Fink, J. Lowry, M. Dean, Dynamic approaches to thwart adversary intelligence gathering, in: Proceedings of TheDARPA Information Survivability Conference and Exposition II, DISCEX 2001, Anaheim, Calif, USA, 2001, pp. 176–185.
[13] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, J. Tront, MT6D: a moving target IPv6 defense, in: Proceedings of the Military Communications Conference (MILCOM '11), IEEE, Baltimore, Md, 2011, pp. 1321–1326.
[14] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, J. Tront, The blind Man's bluff approach to security using IPv6, IEEE Secur. Priv. 10 (4) (2012) 35–43.
[15] H. Okhravi, A. Comella, E. Robinson, J. Haines, Creating a cyber moving target for critical infrastructure applications using platform diversity, Int. J. Crit. Infrastruct. Prot. 5 (1) (2012) 30–39.
[16] B. Salamat, A. Gal, T. Jackson, K. Manivannan, G. Wagner, M. Franz, Multi-variant program execution: Using multi-core systems to defuse buffer-overflow vulnerabilities, in: Proceedings of the IEEE International Conference on Complex, Intelligent and Software Intensive Systems, 2008, pp. 843–848.
[17] H. Okhravi, J. Riordan, K. Carter, Quantitative evaluation of dynamic platform techniques as a defensive mechanism, in: A. Stavrou, H. Bos, G. Portokalidis (Eds.), Research in Attacks, Intrusions and Defenses, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, Switzerland, 2014, pp. 405–425.
[18] C. Lei, D.-H. Ma, H.-Q. Zhang, Optimal strategy selection for moving target defense based on markov game, IEEE Access 5 (2017) 156–169.
[19] A. Prakash, M.P. Wellman, Empirical game-theoretic analysis for moving target defense, in: Proceedings of the Second ACM Workshop on Moving Target Defense, ACM, 2015, pp. 57–65.
[20] R. Colbaugh, K. Glass, Predictability-oriented defense against adaptive adversaries, in: Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on, IEEE, 2012, pp. 2721–2727.
[21] A. Chowdhary, S. Pisharody, D. Huang, SDN based scalable MTD solution in cloud network, in: Proc. ACM Workshop Moving Target Defense, 2016, pp. 27–36.
[22] A.C. Pappa, Moving Target Defense for Securing Smart Grid Communications: Architectural Design, Implementation and Evaluation (Ph.D. dissertation), Elect. Comput. Eng. Iowa State Univ., 2016.
[23] A.C. Pappa, Moving Target Defense for Securing Smart Grid Communications: Architectural Design, Implementation and Evaluation (Ph.D. dissertation), Iowa State University, 2016.
[24] P.J. Fleming, R.C. Purshouse, Evolutionary algorithms in control systems engineering: a survey, Control Eng. Pract. 10 (11) (2002) 1223–1241.
[25] J. Zabczyk, Mathematical Control Theory: An Introduction, Springer Science & Business Media, Boston, Mass, 2009.
[26] E. Alpaydin, Introduction to Machine Learning, MIT Press, 2014.
[27] M. Chew, D. Song, Mitigating Buffer Overflows By Operating System Randomization, Technical Report CMU-CS-02-197, 2002.
[28] G.S. Kc, A.D. Keromytis, V. Prevelakis, Countering code-injection attacks with instruction-set randomization, in: Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS, 2003, pp. 272–280.
[29] M. Christodorescu, M. Fredrikson, S. Jha, J. Giffin, End-to-end software diversification of internet services, in: Moving Target Defense, in: Advances in Information Security, vol. 54, Springer, 2011, pp. 117–130.
[30] M. Albanese, A. De Benedictis, S. Jajodia, K. Sun, A moving target defense mechanism for manets based on identity virtualization, in: IEEE Conference on Communications and Netw. Secur. (CNS), 2013, pp. 278–286.
[31] A. Chowdhary, A. Alshamrani, D. Huang, H. Liang, Mtd analysis and evaluation framework in software defined network (mason), in: Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, ACM, 2018, pp. 43–48.
[32] M. Samir, M. Azab, M.R.M. Rizk, N. Sadek, PYGRID: A software development and assessment framework for grid-aware software defined networking, Int. J. Netw. Manage. 28 (5) (2018) 1–15.
[33] E. Molina, E. Jacob, Software-defined networking in cyber–physical systems: A survey, Comput. Electr. Eng. 66 (2018) 407–419.
[34] U. Ghosh, P. Chatterjee, S. Shetty, A security framework for SDN-enabled smart power grids, in: Proc. - IEEE 37th Int. Conf. Distrib. Comput. Syst. Work. ICDCSW 2017, 2017, pp. 113–118.
[35] M. Govindarasu, Moving target defense for securing smart grid communications: Architectural design, implementation and evaluation, 2016.
[36] J. Tian, R. Tan, X. Guan, T. Liu, Enhanced hidden moving target defense in smart grids, IEEE Trans. Smart Grid 3053 (c) (2018) 1–15.
[37] A. Aydeger, N. Saputro, K. Akkaya, M. Rahman, Mitigating crossfire attacks using SDN-based moving target defense, in: Proc. - Conf. Local Comput. Networks, LCN, 2016, pp. 627–630.
[38] [Online]. Available: http://mininet.org/.
[39] R. Wallner, R. Cannistra, An SDN approach: Quality of service using big switch's floodlight open-source controller, Proc. Asia-Pac. Adv. Netw. 35 (2013) 14.
[40] J.B. Hong, S. Yoon, H. Lim, D.S. Kim, Optimal network reconfiguration for software defined networks using shuffle-based online MTD, in: Proc. IEEE Symp. Reliab. Distrib. Syst. Vol. 2017–September, 2017, pp. 234–243.
[41] J. Steinberger, B. Kuhnert, C. Dietz, L. Ball, A. Sperotto, H. Baier, A. Pras, G. Dreo, DDoS defense using MTD and SDN, in: IEEE/IFIP Netw. Oper. Manag. Symp. Cogn. Manag. a Cyber World, NOMS 2018, 2018, pp. 1–9.
[42] [Online]. Available: https://onosproject.org/.
[43] T. Hu, P. Yi, J. Zhang, J. Lan, Reliable and load balance-aware multi-controller deployment in SDN, China Commun. 15 (2018) 184–198.
[44] P. Kampanakis, H. Perros, T. Beyene, SDN-based solutions for Moving Target Defense network protection, in: Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014 WoWMoM 2014, 2014.
[45] [Online]. Available: https://www.cisco.com/en/US/prod/iosswrel/onepk.html.
[46] B. Nathan, B. Ryan, K. Roger, S. Howard, W. Bryan, Moving target defense considerations in real-time safety- and mission-critical systems, in: MTD'20, 2020.
[47] P. Bradly, D. Abhishek, Z. Zhenkai, Moving target defense for the security and resilience of mixed time and event triggered cyber-physical systems, J. Syst. Archit. (2022).
[48] P. Bradly, Z. Zhenkai, K. Xenofon, Integrated moving target defense and control reconfiguration for securing Cyber-Physical systems, J. Microprocess. Microsyst. (2020).
[49] G. Jairo, C. Alvaro, Moving Target Defense Attack Mitigation in Multi-Vehicle Systems, Chaptere7, http://dx.doi.org/10.1007/978-3-030-10597-6_7.
[50] M. Samir, M. Azab, E. Samir, SD-CPC: SDN controller placement camouflage based on stochastic game for moving-target defense, Comput. Commun. (ISSN: 0140-3664) 168 (2021) 75–92.
[51] M. Samir, E. Samir, M. Azab, M.R.M. Rizk, N. Sadek, On managing controller placement problem for grid-aware software defined networking, in: 2019 IEEE Global Conference on Internet of Things (GCIoT), 2019, pp. 1–5, http://dx.doi.org/10.1109/GCIoT47977.2019.9058403.
[52] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia, Pareto-optimal resilient controller placement in SDN-based core networks, in: Proc. 2013 25th Int. Teletraffic Congr ITC 2013, 2013.
[53] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, M. Hoffmann, Heuristic approaches to the controller placement problem in large scale SDN networks, IEEE Trans. Netw. Serv. Manag. 12 (1) (2015) 4–17.
[54] C. Boettiger, An introduction to Docker for reproducible research, with examples from the R environment, 2014.
[55] K.T. Seo, H.S. Hwang, I.-Y. Moon, O.Y. Kwon, B.-J. Kim, Performance comparison analysis of linux container and virtual machine for building cloud, Adv. Sci. Technol. Lett. 66 (2014) 105–111.
[56] [Online]. Available: https://linuxcontainers.org/.
[57] [Online]. Available: https://openvz.org/Main_Page.
[58] [Online]. Available: https://runc.io/.
[59] [Online]. Available: https://criu.org/Main_Page.
[60] M. Jammal, T. Singh, A. Shami, R. Asal, Y. Li, Software defined networking: State of the art and research challenges, Comput. Netw. 72 (2014) 74–98.

[61] J. Medved, R. Varga, A. Tkacik, K. Gray, OpenDaylight: Towards a model-driven SDN controller architecture, in: Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014, WoWMoM 2014, 2014.
[62] S. Kaur, J. Singh, N.S. Ghumman, Network programmability using POX controller, in: Int. Conf. Commun. Comput. Syst. no. December 2015, 2014, p. 5.
[63] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69.
[64] [Online]. Available: https://www.python.org.
[65] Edmunds, Calum, Simon Gill, England network analysis, 2017.
[66] D.M. Laverty, J. O'Raw, D.J. Morrow, M. Cregan, R. Best, Practical evaluation of telecoms for Smart Grid measurements, control and protection, in: IEEE PES Innov. Smart Grid Technol. Conf. Eur., 2011, pp. 1–5.
[67] G. Van Brummelen, Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry, Princeton University Press, 2013, pp. 161–169.

Mohamed Azab is an assistant professor at the Department of Computer and Information Sciences, Virginia Military Institute, VA, USA. He is also affiliated with, The SmartCI Research Center, VT-MENA. Mohamed received his Ph.D. in Computer Engineering in 2013 from The Bradley Department of Electrical and Computer Engineering at Virginia Tech. He has multiple provisional patents, book chapters, and reference publications in archival journals and respected conference proceedings. His research interests lie in the area of cybersecurity and trustworthy engineering ranging from theory to design and implementation. His recent research crosscuts the areas of cyber security, Software Defined Networking (SDN) architectures and protocols, high performance and cloud computing, ubiquitous Internet of Things (IoT), and Cyber–Physical Systems (CPS). Mohamed is the founder of the Cyber Security and IoT lab, hosting Mohamed's Ph.D. and Masters students' research activities. Mohamed's work is supported by multiple active research grants. Mohamed acted as a keynote speaker in multiple prestigious conferences. He served as a member of the steering committee of conferences, workshops, and archival journals.

Mohamed Samir received his Ph.D. in electrical communication engineering from Faculty of Engineering, Alexandria University, Egypt, in 2020. Since 2013, his research on Resilient Software Defined Networking (SDN) for Cyber–Physical System (CPS) Security has been supported by Smart-CI, Alexandria University, Egypt. He is currently working as an information security consultant at the Arab Academy for Science, Technology and Maritime Transport (AAST-MT). He is a certified CEH, CCNP-Security, MCSE plus security, and FCNSA. He is a specialist in enterprise data centers security operation and design. Mohamed Samir was a recipient of a certificate of appreciation from the minister of State for Administrative Development, Egypt, in 2006. He is a member of the International Association of Computer Science and Information Technology (IACSIT).

Effat Samir is a Ph.D. Research at The department of Electrical and Computer Engineering, Old Dominion University, USA. Effat received her Bsc. and M.Sc. in the Department of Communication and Electronics, Faculty of Engineering, Alexandria University, Egypt in 2013 and 2017. She has worked on multiple research projects focusing on both physical and application layers. She has two book chapters among various publications in archival journals and conference proceedings. She developed huge research interests lie in the Cybersecurity field, including Cryptography and authentication techniques, Blockchain networks and smart contracts, Internet of Things (IoT), and Machine learning.