Abstract—Malicious co-residency in virtualized networks poses a real threat. The next-generation mobile networks heavily rely on virtualized infrastructure, and network slicing has emerged as a key enabler to support different virtualized services and applications in the 5G network. However, allocating network slices efficiently while providing a minimum guaranteed level of service, as well as providing defense against the threat of malicious co-residency in a mobile core network, is challenging. To address these questions, in our previous work, we proposed an optimization model to allocate slices. It provided a static and manual allocation of slices. In this work, we analyze the defense against the malicious co-residency using our optimization-based allocation, and we extend our work to dynamically allocate slices. We propose a dynamic slice allocation framework for the 5G core network. The proposed framework provides user-interaction to request slices and any required services that need to run on a slice(s). It can accept a single or multiple allocation requests, and it dynamically allocates them. Additionally, the framework allocates slices in a balanced fashion across available resources. We compare our framework with the First Come First Serve and First Available allocation scheme.

Index Terms—5G slicing, network slicing, 5G availability, 5G optimization, slice allocation, co-location

I. INTRODUCTION

Network Slicing has been proposed to cope with the ever-growing demand for flexibility and scalability in 5G mobile network [1], [2]. The recent advancements in Network Function Virtualization (NFV) has enabled next-generation mobile networks to employ concepts like network slicing to satisfy diverse requirements from various new applications [3]. The Next Generation Mobile Network Alliance (NGMN) defined network slicing as running multiple services with different requirements such as performance, security, availability, reliability, mobility, and cost as an independent logical network on the shared physical infrastructure [1], [4]. An end-to-end slice is created by pairing the RAN and core network slice, but the relationship between both slices could be 1-to-1 or 1-to-M [5], [6].

One of the key requirements for network slicing is resource isolation between different slices [3], [1]. However, guaranteeing resource isolation between slices that share the common physical infrastructure is challenging [4]. The sharing of common physical resources between slices could lead to information leakage and side-channel attacks [7], [8]. The side-channel attacks can be used to determine co-residency and extract valuable information (e.g., cryptographic keys [9]) from the victim slices or perform Denial-of-Service attacks [10]. There are several types of side-channel attacks that can be used to determine co-residency by using different shared resources such as CPU cache, main memory, and network traffic [9]. Therefore, it is paramount to provide defense against malicious co-residency and minimize side-channel attacks.

Network slicing in a 5G network presents a unique challenge that is not present in the previous or current mobile networks. The challenge is how to allocate slices optimally and dynamically to efficiently use the mobile network resources as well as guarantee minimum requested resources. To address this question, we proposed an optimization-based slice allocation (optimization model) in the 5G core network [11]. The proposed optimization-based allocation was designed to allocate slices based on Central Processing Unit (CPU) utilization and link delay. The optimization model provided intra-slice isolation for network functions within a slice. Intra-slice isolation provides a variable degree of physical separation between slice components. For instance, if a slice requires an intra-slice isolation level that is equal to 1, then only a single component (network function) of the slice will be hosted on a given hypervisor. The allocation was mathematically simulated through MATLAB. All slice requests were provided as one input to the optimization model. This work was extended for utilization in DDoS mitigation, and it was evaluated by using a real testbed [10]. However, the allocation of slices in the tested was done manually and in a static fashion and the slice requests were still provided as one input to the optimization model. There was no mechanism to deal with requests arriving in real-time.

We address these points in this paper. Our contributions are (1) analysis of the impact of optimization-based slice allocation on malicious co-residency (2) discussion on additional measures that can be taken to further minimize the risk of co-residency and (3) we propose Dynamic Slice Allocation Framework (DSAF) that can allocate slice dynamically in a real-time. DSAF can perform seamless slice allocation and provide on-demand intra-slice isolation. Only user interaction required in DSAF is when a user requests a slice allocation; every other procedure is automated. DSAF implements our optimization model that can fulfill several requirements of the 5G mobile core network. We compared our proposed framework with the First Come First Serve and First Available (FCFSFA) allocation scheme. Both are evaluated on a real testbed. We assume that core network slices can be allocated independently of radio network slices. Therefore, we do not
discuss the radio resource slicing.

The rest of this paper is organized as follows. In section II, we present the literature review on co-residency and 5G testbeds. Section III provides an overview of the optimization model for 5G network slicing. The threat model is presented in section IV. We discuss the evaluation of defense against malicious co-residency in section V. Section VI provide an overview of the framework and experimental evaluation is discussed in section VII. The conclusion is presented in section VIII.

II. RELATED WORK

Malicious Co-Residency: The next-generation mobile networks will have similar co-residency issues as cloud networks since both networks share some properties, i.e., shared resources and multi-tenancy. The work on the co-residency issue in the 5G network is currently limited due to its infancy. Therefore in this section, we describe some of the state-of-the-artwork on co-residency detection in cloud networks, which would still be applicable in the 5G network because both networks share virtualized infrastructure.

Network traffic is one of the shared resources that can be used to determine co-residency. In [12], A. Bates et al. used network traffic watermarking to detect co-residency with the victim. In the proposed scheme, the attacker launches multiple Virtual Machine (VM) instances called FLOODER that communicate with the CLIENT, which is outside the cloud network. The CLIENT sends legitimate traffic to the target (victim) server that resides inside the cloud network. The FLOODER VMs flood the network with traffic to cause network delays and the CLIENT analyzes these delays to determine which FLOODER is co-resided with the target server. Another aspect of network traffic can be exploited by analyzing Round Trip Time (RTT) to detect co-residency. Such a method is discussed by A. Atya et al. in [7]. In the proposed work, TCP handshake is used to measure the RTT (in some cases, ICMP was also used) to determine co-residency. RTT is calculated from multiple sources and vantage points to increase the accuracy of detecting co-residency. A migration scheme was proposed to defend against co-residency attacks. An extension of their work is also discussed in [13].

CPU cache-based side-channel attacks are commonly used to detect co-residency with the victim VM. Authors Y. Zhang et al. [14] used L2 cache to detect co-residency in the cloud environment. The objective of their works was to use side-channel to detect undesired co-residency. The basic idea of HomeAlone was to coordinate with other friendly VMs and analyze the cache usage to determine if there are any undesired VMs hosted on the same hypervisor.

5G Testbeds: A Practical Open Source Solution for End-to-End Network Slicing (POSENS) has been proposed by G. Aviles et al. [15]. POSENS uses open source software and hardware to create end-to-end slices. Authors used srsLTE for radio access network and OpenAirInterface for the core network. The independence of slices and performance throughput is discussed in the paper. It also supports an efficient and flexible deployment of network slices. L. Zanzi et al. [16] demonstrated a real testbed named OVerbooking NEtwork Slices (OVNES). In the testbed, authors considered three different vertical slices, i.e., Public Safety communications, enhanced Mobile BroadBand (eMBB) for voice calls, and eMBB for the Internet (best-effort). OpenEPC is used to emulate mobile core and several LTE devices to generate traffic.

III. OPTIMIZATION MODEL

In our previous work [10], we proposed an optimization model to mitigate DDoS attacks. The proposed model mitigated DDoS attacks using intra-slice (between slice components) and inter-slice1 (between slices) isolation. In addition to providing defense against DDoS, it can optimally allocate slices. Our model allocated slices to the least loaded2 servers and finds the minimum delay path (Eq. 1). The optimization model also fulfills several requirements of the 5G network. It can guarantee the end-to-end delay and provide different levels of slice isolation for reliability and availability as well as it assures that allocation does not exceed the available system resources. In our model, we only considered CPU, bandwidth, Virtual Network Function (VNF) processing delay, and link delay. Intra-Slice isolation could increase the availability of a slice. If all components of the slice are hosted on the same hypervisor, any malfunction could result in the slice unavailability. However, different levels of intra-slice isolation can ensure that full or partial slice remains available.

A summary of the optimization model presented in [10] is provided here for better readability. More details can be found in [10]. We use an undirected graph \( G_p = (N_p, L_p) \) to represent the physical 5G core network topology. All the nodes in the network (i.e., servers, switches, routers and other devices present in the network) are represented by \( N_p \) and \( L_p \) denotes all the physical links between the nodes. A slice request is denoted by a directed graph \( G_v = (N_v, L_v) \), where \( N_v = (N_c \cup N_d) \) contains all the slice virtual network functions, the control and data plane virtual functions are represented by \( N_c \) and \( N_d \), respectively and \( L_v \) represents requested links. Each edge in the directed graph is represented by \((i, j) \in L_v\). Each slice request is associated with end-to-end delay \((d_{E2E})\), intra-slice isolation \((K_{c}, K_{d})\), and each VNF in a slice is associated with a computing demand \((R^c)\), and bandwidth (BW) requirement between VNF \( i \) and VNF \( j \) \((R^b)\). The description of all parameters is provided in table I.

\[
\text{Minimize} \quad \sum_{i \in N_v} \sum_{k \in N_p} (\sigma_k + R^b) u^k_i + \sum_{(i,j) \in L_v} \sum_{(e,f) \in L_p} T_{e,f} y_{e,f}^{i,j} (1)
\]

The objective function of our optimization model (eq. 1) allocates the slice to the least loaded physical nodes and find

\footnote{Please note that in this paper we did not consider the inter-slice isolation}

\footnote{All servers have same max. CPU capacity so least loaded is also least utilized}
TABLE I: Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N_p )</td>
<td>Set of physical Nodes</td>
</tr>
<tr>
<td>( L_p )</td>
<td>Physical links between nodes</td>
</tr>
<tr>
<td>( \sigma_k )</td>
<td>Current CPU allocation of physical node ( k )</td>
</tr>
<tr>
<td>( \sigma_{ef} )</td>
<td>Current BW allocation of physical link</td>
</tr>
<tr>
<td>( \sigma_{max} )</td>
<td>Maximum CPU capacity of physical node ( k )</td>
</tr>
<tr>
<td>( \sigma_{max} )</td>
<td>Maximum BW capacity of physical link</td>
</tr>
<tr>
<td>( \Delta_k )</td>
<td>VNF ( i ) processing delay</td>
</tr>
<tr>
<td>( \Delta_k )</td>
<td>Physical node ( k ) processing delay</td>
</tr>
<tr>
<td>( \Delta_i )</td>
<td>VNF ( i ) processing delay</td>
</tr>
<tr>
<td>( N_c )</td>
<td>Requested set of slice control plane functions</td>
</tr>
<tr>
<td>( N_d )</td>
<td>Requested set of slice data plane functions</td>
</tr>
<tr>
<td>( N_v )</td>
<td>Requested set of slice VNFs (( N_c \cup N_d ))</td>
</tr>
<tr>
<td>( L_v )</td>
<td>Virtual links of a slice</td>
</tr>
<tr>
<td>( R^i )</td>
<td>Physical link delay between node ( e, f )</td>
</tr>
<tr>
<td>( R^i )</td>
<td>Physical link delay between node ( e, f )</td>
</tr>
<tr>
<td>( d_{L2E} )</td>
<td>Physical link delay between VNF ( i, j )</td>
</tr>
<tr>
<td>( K_{rel}^c )</td>
<td>Requested intra-slice isolation for Control Plane</td>
</tr>
<tr>
<td>( K_{rel}^d )</td>
<td>Requested intra-slice isolation for Data Plane</td>
</tr>
<tr>
<td>( u_k )</td>
<td>Indicates the assignment of VNF ( i ) to EPC node ( k )</td>
</tr>
<tr>
<td>( y_{ef} )</td>
<td>Indicates the assignment of link ( (e, f) )</td>
</tr>
</tbody>
</table>
| \( y_{ef} \) | End-to-End Delay:

\[
\sum_{(i,j) \in L_v} \sum_{e,f \neq f} T_{ef} y_{ef} + \sum_{i \in N_v} \left( \Delta_i + \sum_{k \in N_p} \Delta_k u_k \right) \leq d_{L2E} \tag{9}
\]

\[
T_{ef} = \frac{\sigma_{ef}}{\sigma_{max}^e} \delta + T_{ef,init} \quad \forall (e, f) \in L_p \tag{10}
\]

Constraint (9) guarantees end-to-end delay for a slice in the current network state. End-to-end delay includes link delay, VNF processing delay, and physical node processing delay. Each time when a virtual link \( (i,j) \in L_v \) is assigned to a physical link \( (e, f) \in L_p \), it increases \( T_{ef} \). \( T_{ef} \) is a function of link utilization, and it is calculated using eq. (10), where \( T_{ef,init} \) is the initial delay assigned to the link \( (e, f) \in L_p \) and \( \delta \) is the maximum increase in delay.

3) Intra-Slice Isolation

\[
\sum_{i \in N_v} u_k \leq K_{rel}^c \quad \forall k \in N_p, K_{rel}^c = 1, 2, 3... \tag{11a}
\]

\[
\sum_{i \in N_d} u_k \leq K_{rel}^d \quad \forall k \in N_p, K_{rel}^d = 1, 2, 3... \tag{11b}
\]

It might be required to have different levels of intra-slice isolation for control and data plane. In constraints (11), \( K_{rel}^c \) and \( K_{rel}^d \) ensure the intra-slice isolation for control plane and data plane, respectively. Intra-slice isolation can improve the availability of a slice.

IV. THREAT MODEL

Assumptions: Our threat model assumes that the target network supports network slicing. The Evolved Packet Core (EPC) allows migration of the slice components (the slice operator or users can migrate the slice(s)), and the slice operator supports multi-tenancy. Adversaries do not know about the operators’ allocation scheme, and adversaries can successfully determine the co-residency with the victim slice.

Adversary: The adversaries can launch multiple VMs and check for the co-residency with the victim. If co-residency is found then, the attacker could launch the next attack(s). If not, remove the slice and launch new ones and repeat the process. We assume that adversaries are not colluding.

V. EVALUATION OF DEFENSE AGAINST THE MALICIOUS CO-RESIDENCY

A. Simulation Setup

In our simulation, we use MATLAB to calculate the optimization solution and perform pre/post-processing of data.
The preprocessing of data involves reading current network topology, slice requests, and updating the optimization model. In the post-processing, we update the network topology after a slice is allocated. AMPL is used to model optimization algorithm, and CPLEX 12.9.0.0 (ILOG) is used as a MILP solver. The optimization model is evaluated on Intel Core i7-8700 3.2 GHz with 32 GB RAM. We simulate 200 servers as shown in Fig. 1 (we used similar topology to [11]). Other parameters used for the evaluation are listed in Table II.

![Simulation Topology](image)

**Fig. 1: Simulation Topology**

In our simulation, we vary the level of intra-slice isolation using the $K_{rel}$ parameter. This parameter provides the upper limit for how many VNFs can be placed on one physical server. For $K_{rel}$ 1 to 10, all slices request the same level of intra-slice isolation. The average overall CPU utilization of the entire system is also restricted at 50%, 75%, 80%, 85%, 90%, and 95% ($\pm 0.5\%$). For each average CPU utilization (ACU), we vary the $K_{rel}$ (e.g., at average CPU utilization 50%, $K_{rel}$ will be varies from 1 to 10).

<table>
<thead>
<tr>
<th>TABLE II: Simulation Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>$N_p$</td>
</tr>
<tr>
<td>$\sigma_{K}^{init}$</td>
</tr>
<tr>
<td>$K_{rel}$</td>
</tr>
<tr>
<td>$N_v$</td>
</tr>
<tr>
<td>$R^{ij}$</td>
</tr>
<tr>
<td>$R^i$</td>
</tr>
<tr>
<td>$\Delta_i$</td>
</tr>
<tr>
<td>$\Delta_k$</td>
</tr>
<tr>
<td>$\delta$</td>
</tr>
<tr>
<td>$T_{p,init}$</td>
</tr>
<tr>
<td>Total Attacker Slice Requests</td>
</tr>
<tr>
<td>Target Slice</td>
</tr>
</tbody>
</table>

In each simulation, the attacker requests allocation of a slice and determines if there is a co-residency with the victim slice (i.e., one or more VNFs of the victim slice are allocated on the same hypervisor as the attacker). If co-residency is found, it is considered as a success (we assume that attacker will move to the next step of their objective in the real-world), and if no co-residency is found then, it is considered as failure. In either case, we deallocate the attacker slice and request a new slice. We repeat this process 500 times and calculate the average success rate. The success is defined as if one or more VNFs of a victim slice are allocated on the same hypervisor as the attacker. We only generate 500 attacker requests once at the beginning of the simulation. To simulate a more realistic scenario, every 60 seconds a legitimate slice is deallocated, and a new slice is allocated. The target slice is never deallocated.

![Comparison of Co-Residency Success rate for different average CPU utilization](image)

**Fig. 2: Comparison of Co-Residency Success rate for different average CPU utilization**

### B. Results and Discussion

Fig. 2 shows the relationship between different levels of intra-slice isolation and the success rate of getting a co-residency with any network function of the target slice. In the figure, there is a relatively higher chance of getting co-residency when $K_{rel} \leq 3$ and lower ACU because of two reasons. First, at $K_{rel} \leq 3$, the network functions are more spread across the network, which increases the chances of getting co-residency with a specific target slice. Second, at relatively lower ACU there are more opportunities to get co-allocation. Whereas at higher $K_{rel} \geq 4$ and $ACU \geq 80\%$, we see a significant decrease in the success rate of getting a co-residency. For instance, $K_{rel} = 1$ and $ACU = 75\%$, the success rate is 56% whereas $K_{rel} = 1$ and $ACU = 80\%$, the success rate is only 29% (almost 50% reduction in success rate). At $K_{rel} \geq 4$, the slice would have a certain degree of isolation as well as at $ACU > 75$ present a more realistic scenario for the slice operator because the network resources will be better utilized. Please note that a detailed analysis

ACU < 50\% does not yield meaningful results due to low resource utilization. Therefore, we have not shown those results here.
of the optimization model’s performance and efficiency is presented in [10], [11].

C. Defense
There are few additional methods that can be employed to reduce the threat of malicious co-residency.

- Migrate the target slice to a different location (hypervisor). It could be a slice operator or the user (if allowed) that can migrate the slice.
- Detect anomalous behavior for slice allocation requests (e.g., monitor IP/MAC/unique user ID or some other parameters) and take necessary preventive measures.
- Limit the number of slice requests allowed per user within a specified period and the total number of requests.
- Randomize the time between slice requests and creation, thus making it harder to infer the allocation scheme.

VI. DSAF: Dynamic Slice Allocation Framework
To automate the process of slice allocation in 5G core network, we propose a framework. In the proposed framework, slices can be allocated dynamically. The Dynamic Slice Allocation Framework (DSAF) consists of five components as shown in Fig. 3.

![Fig. 3: DSAF Logical Topology in our experiments: The brown boxes represent the physical servers and blue boxes represent the framework components. Solid lines show the logical communication paths between framework components](image)

TABLE III: Example Slice CPU Requests Database

<table>
<thead>
<tr>
<th>Slice#</th>
<th>VNF#</th>
<th>CPU (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>V1</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>V2</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>V3</td>
<td>0.4</td>
</tr>
<tr>
<td>S2</td>
<td>V1</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>V2</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>V3</td>
<td>0.75</td>
</tr>
</tbody>
</table>

- The Orchestrator: The Orchestrator or the slice manager is responsible for managing slices, facilitating on-demand slice allocation, and coordinating different components of the framework as well as user interactions.
- Optimization Module: The optimization module implements our optimization model [10]. It reads the current state of the system allocation that includes remaining CPU, link bandwidth, and delays as well as network topology and processes the incoming request. If a solution is found, the allocation is stored in a database, and the system allocation statistics are updated accordingly.

- Database (DB): The database stores request information, allocation scheme, remaining system resources, and performance statistics. Table III and IV shows an example of CPU requests and allocation database, respectively.
- O Agent: The Optimization agent (O Agent) is responsible for communicating with the Orchestrator and the Optimization Module. It receives the slice allocation from the Orchestrator and forwards it to the optimization module and communicate back the results.
- H Agent: The Hypervisor Agent (H Agent) is an integral part of the framework (runs on each hypervisor). It is responsible for allocating slices in real-time, starting applications for each slice, and reporting slice statistics to the Orchestrator.

TABLE IV: Example Slice CPU Allocation Database

<table>
<thead>
<tr>
<th>Slice#</th>
<th>VNF#</th>
<th>Allocated Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>V1</td>
<td>S1</td>
</tr>
<tr>
<td></td>
<td>V2</td>
<td>S2</td>
</tr>
<tr>
<td></td>
<td>V3</td>
<td>S3</td>
</tr>
<tr>
<td>S2</td>
<td>V1</td>
<td>S4</td>
</tr>
<tr>
<td></td>
<td>V2</td>
<td>S5</td>
</tr>
<tr>
<td></td>
<td>V3</td>
<td>S4</td>
</tr>
</tbody>
</table>

The dynamic slice allocation process is shown in Fig. 3 and it is described in the following steps (each step corresponds to the circled number in Fig. 3):

1) The Orchestrator provides user-interaction and waits for a slice request
2) Once a slice request is received, it interacts with the O Agent to find the allocation scheme.
3) The O Agent starts an instance of the optimization model and pass the slice request to the optimization module. The optimization module reads the current network state.
from the database and finds the best solution to allocate
the slice (if feasible). If no solution is found, the request
is denied, and the response is sent to the Orchestrator.
The flow diagram of the allocation process is shown in
Fig. 4.

4) If a solution is found in step 3, then the slice allocation
will be stored in a database.
5) The Orchestrator receives an accepted or denied re-
response from the O Agent.
6) If the response is slice request accepted, the Orchestrator
retrieves the allocation scheme from the database and
sorts the retrieved allocation scheme according to the
slice(s) that will be allocated on each hypervisor.
7) The Orchestrator sends the information to target H
Agent(s). The information includes slice name, IP ad-
dress, CPU (GHz), HDD, RAM, bandwidth, and any
application to start after the creation of VNF.
8) If the allocation is successful, the H Agent sends a
successful response to the Orchestrator.

![Fig. 4: Dynamic Slice Allocation Optimization Flow Diagram](image)

This process is repeated for every request, although DSAF
should be able to process multiple requests at the same time.

VII. EXPERIMENTAL EVALUATION OF DSAF

The implementation of a complete 5G network is still in
the early stages, and evaluating a fine-grained network slicing
solution like ours in a 5G network is very challenging. There
are very few testbeds and even fewer available to academia for
testing and experimentation. We are not aware of any publicly
available simulator or emulators that can provide complete
simulation or emulation of 5G RAN and core network with
slicing support. Another challenging issue is the cost of 5G
hardware and software. Therefore, we made the best use of
currently available tools (open source) and generic hardware
to evaluate our solution.

![Fig. 5: DSAF Physical Topology. All links are 1 Gbps](image)

A. Experimental Setup

To evaluate DSAF, we created a testbed using seven servers
where five servers (P1 to P5) are used to allocate slices.
We are aware that our topology is different from the typical
NFV architecture framework, such as MANO [17], but for
simplicity of implementation, we use topology, as shown in
Fig. 5. The optimization module and the database are hosted
on the same server (Remote Server), and the Orchestrator is
hosted on a separate server. The hardware specification for all
nodes are listed in Table V. The slice request parameters are
listed in Table VI. For simplicity, each request arrives every
three seconds (random interval can also be used), and they are
allocated in the order of arrival. We note that slice requests
do not expire.

![TABLE V: Experiment Topology Hardware Specification](image)

We implemented the DSAF in Python [18] (i.e., the Or-
chestrator, O, and H agents). The Optimization module uses
the same tools and configurations, as described in section V.
OpenVZ [19] is used for virtualization. It is an open-source
container-based virtualization platform. OpenVZ allows each
container to have a specific amount of CPU, RAM, and Hard
Drive (HDD). Each container (which hosts one VNF) performs and executes like a stand-alone server. We have installed the CentOS 6 [20] operating system in every container. We used Linux Traffic control (tc) [21] to allocate bandwidth for each container.

We used three scenarios to allocate slices. In each scenario, we compare the DSAF with the First Come First Serve First Available (FCFSFA). In all scenarios, we collected statistics in terms of total slice requests allocated, processing time, and average computation time per slice. In the first scenario, we restricted the allocation to one VNF/hypervisor per slice ($K_1$). In the second scenario, only two or fewer VNFs of a slice ($K_2$) can be allocated on a single hypervisor. In the third scenario, three or less VNFs/slice can be placed on one hypervisor ($K_3$).

In FCFSFA, a slice is allocated based on arrival time and the first available server. We make sure that allocated resources do not exceed the available physical resources. However, FCFSFA cannot guarantee the end-to-end delay. We wrote a Python script to perform allocation for FCFSFA. The FCFSFA is implemented on the same server as the Orchestrator.

B. Results and Discussion

Fig 6 shows slice allocation for all scenarios. In scenario $K_3$, all requests are successfully allocated for both allocation schemes, as shown in Fig. 6c and 6f. However, in scenario $K_1$, DSAF and FCFSFA only allocated 27 and 17 slice requests respectively before $P_1$, $P_2$ and $P_3$ ran out of CPU capacity. For scenario $K_1$, we can only allocate one VNF/hypervisor per slice; therefore, once these three hypervisors ran out of CPU capacity, we cannot allocate any more slices even though $P_4$ and $P_5$ still have significant resources available (because each slice needs three hypervisors for allocation).

An interesting observation to note in Fig. 6 is that FCFSFA (all scenarios) allocates slices in an unbalanced manner. This allocation scheme behaves like a greedy approach, where it will allocate slices at the first available hypervisor. It resulted in a lower number of requests being allocated in $K_1$ and $K_2$, as shown in Fig. 6d and 6e respectively. It could also result in slices competing for resources on one hypervisor sooner, even though the rest of the system is idle as well as a higher chance of slice unavailability if a hypervisor malfunctions. Whereas DSAF optimally allocates slices in all scenarios and spreads them across the entire system leading to less resource contention between slices and in case of a hypervisor malfunction, there is a higher chance that slices could remain partially or fully available. DSAF can allocate more or equal number of slice requests in all scenarios, as shown in Fig. 7.

Fig. 6: Comparison of Allocation Schemes for Different Scenarios

Fig. 7: Total Slice Requests Allocated
and FCFSFA. The processing time overhead is shown in Fig. 8a. The processing time includes the time required to process the user requests, sending and receiving information from the H and O agents. For FCFSFA, the processing time is the time required to retrieve allocation requests and read system topology. Although DSAF requires slightly more processing time because of the communication required between the components of the framework, it still performs comparably to the FCFSFA in all scenarios.

The average computation time per slice for DSAF is measured in the optimization module. The average computation time per slice for FCFSFA is the time required to calculate the allocation of slices and updating DB records. FCFSFA has lower average computation times per slice because there is no optimization performed, as shown in Fig. 8b. DSAF’s average computation time per slice is the cost of allocating more slices as well as providing flexibility when allocating slices.

VIII. CONCLUSION

In this paper, we presented an analysis of the impact of optimization-based slice allocation on malicious co-residency and Dynamic Slice Allocation Framework to allocate slices in a resource-efficient manner. Our optimization model inherently provides proactive defense against malicious co-residency. The success rate of getting co-residency with the target slice decreases with the increase in $K_{rel}$ levels and Average CPU Utilization of the system. The selection of $K_{rel}$ depends on several factors, i.e., cost, security, and performance. For instance, if a slice requires higher level of DDoS protection then lower $K_{rel}$ might be required to provide high availability (e.g., $K_{rel} = 1$). Whereas to reduce the cost and the risk of malicious co-residency, a slice might require higher $K_{rel}$ (e.g., $K_{rel} \geq 4$). Another factor that could impact the selection of $K_{rel}$ is the state of the slice operator’s network (e.g., ACU). Therefore, the selection of $K_{rel}$ greatly depends on the requirements of a slice and state of the operator’s network. The natural defense against malicious co-residency comes at no additional computational cost to the network operator since the cost is already included in the slice allocation.

DSAF provides automation for slice allocation. We compared our framework with the First Come First Serve First Available allocation scheme. The evaluation of both techniques was done on a real testbed. Our results show that the overall proposed framework has comparable overhead to the FCFSFA. The cost of running DSAF is the average computation time that is slightly higher than FCFSFA. However, DSAF can allocate significantly more slices as well as it can fulfill a few requirements of the 5G (e.g., end-to-end delay). DSAF allocates slices in a balanced manner across the network, which means less resource contention between slices until the network reaches a saturation state. In FCFSFA, the resource contention could happen prematurely, because slices are allocated in an unbalanced fashion (i.e., more slices on one hypervisor then the other).

ACKNOWLEDGMENT

The second author acknowledges funding from Canada’s NSERC through the Discovery Grant Program.

REFERENCES


Further investigation, beyond the work in this paper, is needed to give more insight into the selection of $K_{rel}$.