Intrusion Detection and Prevention Systems as a Service in Could-based Environment

Khalid Alsubhi
Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia

Hani Moaiteq AlJahdali
Faculty of Computing and Information Technology Rabig
King Abdulaziz University, Jeddah Saudi Arabia

***Abstract*—Intrusion Detection and Prevention Systems (IDPSs) are standalone complex hardware, expensive to purchase, change and manage. The emergence of Network Function Virtualization (NFV) and Software Defined Networking (SDN) mitigates these challenges and delivers middlebox functions as virtual instances. Moreover, cloud computing has become a very cost-effective model for sharing large-scale services in recent years. Features such as portability, isolation, live migration, and customizability of virtual machines for high-performance computing have attracted enterprise customers to move their in-house IT data center to the cloud. In this paper, we formulate the placement of *Intrusion Detection and Prevention Systems (IDPS)* and introduce a model called *Incremental Mobile Facility Location Problem (IMFLP)* to study the IDPP problem. Moreover, we propose a novel and efficient solution called *Adaptive Facility Location (AFL)* to efficiently solve the optimization problem introduced in the IMFLP model. The effectiveness of our solution is evaluated through realistic simulation studies compared with other popular online facility location algorithms.**

***Keywords*—Facility Location Problem; Intrusion detection and Prevention Systems; Cloud Computing**

I. Introduction

Cloud computing has become a cost-effective model for sharing large-scale services in recent years. Its success is due to the attractive features offered by the underlying virtualization concept, including portability, isolation, live migration, and customizability of virtual machines. Popular examples of cloud-based services are Microsoft Azure, Google AppEngine, and Amazon Elastic Computing Cloud (EC2). Cloud services are generally categorized into three areas: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). In SaaS, a third-party provider host customer's application over the Internet (i.e., Rackspace and SAP Business ByDesign). In PaaS model, both hardware and software are provided and hosted by third-party (i.e., Google App Engine and Microsoft Windows Azure). Finally, IaaS refer to providing virtualized computing resources, usually in terms of VMs (i.e., Amazon EC2, GoGrid and Flexiscale).

Intrusion Detection and Prevention Systems are an essential defensive measure against a range of attacks [44, 47]. In enterprise networked system, IDPSs examine packets sent over networks and trigger alerts when malicious content is discovered and defend against attacks when prevention mode is active. Most issues regarding security in cloud systems are inherited by the current enterprise network [34]. Traditional distributed IDPSs are best practice in providing security for large scale networks. However, the deployment of distributed IDPSs in cloud systems raise many challenges due to the diversity of its services and the complexity of its infrastructure [43].

Network Functions Virtualization (NFV) [1] [2] promises a reprive from the vertically integrated hardware middlebox model followed for decades, by advocating the use of software Network Functions (NFs) running on commodity hardware. This means a reduced acquisition and operational costs, flexible programability, and easier management [31] [42]. Another orthogonal idea is the Software Defined Networking (SDN) that advocates flexible programability in the network. This is done by the separation of the control-plane from the data plane and centralized logical control of the network. SDN simplifies the overall management of the network by allowing deeper programability of the networking devices. Leveraging SDN in environments where NFV are used can leads to several interesting use cases. The high precision control of forwarding elements (switches) provided by SDN can be used to orchestrate traffic patterns between various appliances and NFVs across a data center [22]. In recent years, the cloud has become a mature platform for deploying scalable and cost effective services. With huge growth forecasts, the public cloud industry has grown to become a multi-billion dollar industry [6]. Combining the agility of the cloud with the flexibility of Virtualized Network Functions (VNFs) and the fine-grained control of SDN can bring about a new class of cloud based services for IDPSs [13].

In this paper, we introduce a model in which infrastructure providers support Vritual Intrusion Detection and Prevention Systems (IDPSs) as a Service (IDPSaaS) by leveraging NFV, SDN, and cloud. IDPSaaS services can be enabled or disabled for tenant's Virtual Machines (VMs) on their demands and can be scaled up or down to cope with their service workloads. Moreover, the deployment of multiple IDPS instances of a network functions motivates an interesting challenge, which we call Intrusion Detection and Prevention Systems Placement problem (IDPSP). In order to study the IDPSP problem, we propose Incremental Mobile Facility Location Problem (IMFLP) based on the online facility location problem. IMFLP takes into account the online actions, such as live migrations in cloud, which are ignored in almost all of the existing

models [21]. To the best of our knowledge, it is the first time that the online version of facility location problem has been used to study placement of IDPS. Furthermore, we present an efficient solution for the optimization problem defined in this model called Adaptive Facility Location (AFL). This solution by employing online actions, such as migrations and switches, adjusts the placement of IDPS instances to efficiently adapt to changes in service demands. The effectiveness of our solution is evaluated though realistic simulation studies and empirically compared with several popular online facility location algorithms.

The remainder of this paper is organized as follows. In section II, we formulate the IDPSP problem and present the IMFLP model for studying this problem. We present AFL in section III and conduct experiments to evaluate this algorithm in section IV. The related works are discussed in section V. Finally, we conclude and discuss about future works in section VI.

II. Problem Formulation

As mentioned before, the placement module receives an event of an arrival or leaving of a demand, and by information and functions supported by the management module, adjusts the placement of facilities. In this section, we introduce the Intrusion Detection and Prevention System Placement problem (IDPSP) in section II-A. In section II-B we formally define our model of facility location problem that can be used for modeling the IDPSP problem.

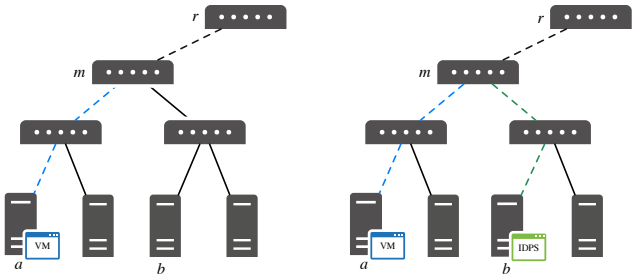
A. Intrusion Detection and Prevention System Placement Problem (IDPSP)

Without loss of generality, we introduce this problem through an example. Suppose that an infrastructure provider offers a IDPSaaS service. From the client's point of view, her VMs can be installed any time, and the IDPSaaS service can be requested and enabled for her VMs at any moment. Moreover, VMs are different and have various service workload on the IDPS instances (*IDPSInst*). Let call each unit of VM's workload as a *demand*. Thus, we can view the problem as dynamic demands that should be served by multiple IDPSInsts.

From the view point of the infrastructure provider, enabling this service incurs certain amount of the installation, operational, and management costs. The installation cost includes the cost of resource consumption of a host machine on which a IDPSInst is installed, and the cost of certain messages between the controller and the host. In our system, all IDPSInsts are same, and therefore the installation cost is same for all IDPSInsts. The operational cost consists of the traffic processing delay cost, and the cost of steering the traffic to the IDPSInst and then to the destination VM. It can be shown that the cost of steering the traffic is related to the distance between IDPSInst and the VM. Finally, the management cost includes the cost of certain statistics collection and syncronization messages between the controller and the IDPSInsts. The management cost is related to the cost of shortest path between the controller(s) and the IDPSInst. Optimizing the management cost is similar to the placement of SDN controllers [8] [29], and is outside of the scope of the current paper.

Considering Figure 1, suppose that a VM exists on host a. As illustrated in Figure 1(a), when there is no IDPSInst enabled (the service-less case), the internet traffic travels the shortest path from the core switch r to the host a with an intermediate switch m. Let $d(r, a)$ represents the cost of the shortest path between r and a. In the service-less case, the cost of traffic traversal is $d(r, a) = d(r, m) + d(m, a)$. On the other hand, as shown in Figure 1(b), when the IDPSInst is installed on a host b (the IDPSInst enabled case), extra costs are paid. Certain amount of b's resources are allocated to the IDPSInst and certain controlling messages from the controller are exchanged with the host b (the installation cost). This installation cost is independent of the *where* IDPSInsts are located, and only depends on the *number* of IDPSInsts. Moreover, IDPSInst adds certain processing delay time t, and the traffic travels a longer path (the operational cost). Delay time t is independent of where the IDPSInst is placed and related to how much traffic is *assigned* to. Additionally, the traffic is steered from core switch r to host b, and from host b to the host a. In this case, the cost of the traffic steering is $d(r, b) + d(b, a) = d(r, m) + 2d(m, b) + d(m, a)$ (We assume that the shortest path cost is symetric). By deducting the cost of service-less case, the extra cost in the IDPSInst enabled case is $2d(m, b)$. Because a and b are in the same level (host level) $d(m, b) = d(m, a)$, and therefore the extra cost is $2d(m, b) = d(m, b) + d(m, a) = d(a, b)$, which is the cost of shortest path between host a containing IDPSInst and host b containing the VM.

There is another complexity dimension that makes the problem even more complicated. Assignments of demands to the IDPSInsts are not irrevocable decisions, and demands can be reassigned to other IDPSInsts. However, these reassignments are not free of charge and associated with certain costs related to the routing reconfiguration and transferring source IDPSInst's internal state to the destination IDPSInst [22]. Furthermore, after assigning more demands to an IDPSInst during the time, this IDPSInst can migrate to another location in order to minimize its distance to the VMs and subsequently reduce the operational costs; however, migrations are not free and are associated with certain cost.



(a) Service-Less Case (b) IDPSInst Enabled Case

Fig. 1. The comparison of the traffic path

Any model describing this problem must consider the dynamic nature of the problem, optimizing the installation and operational cost of the IDPSInsts, and possibility of assignments switches and IDPS migrations.

B. Increamental Mobile Facility Location Model

In this section, we introduce a new model of facility location problem called **Incremental Mobile Facility Location Problem (IMFLP)** to study the IDPSP problem. Before describing our model, we briefly describe why a new model of this problem is needed to be formulated. The details of other existing models will be discussed in the section V-B.

The offline model of facility location problem has been studied comprehensively in the literature [15, 9, 40, 16]. Unfortunately, it cannot describe IDPSP, becuase this model requires demands and their locations to be known in advanced, but in IDPSP, VMs are installed at any moment and subsequently their demands are not known beforehand. In other words, assignments of demands to IDPSInsts are done without knowledge about the future demands. Hence, the online model of this problem should be used. However, the existing online models in the literature (as will be discussed in section V-B) are not representative for our problem, thus we design a new model of this problem. Our IMFLP model relaxes certain constraints of the these models and resolve their limitations in describing IDPSP problem to model migrations and assignments switches.

We describe our model of facility location problem by defining the *space and metrics*, *facilities*, *demands*, and *allowed actions*.

Space and metrics. Given a connected weighted graph $M = (V, E)$ representing the architecture of the data center network, where V denotes the set of nodes (switches or hosts), and $E : V \times V \rightarrow \mathcal{R}^+$ represents the set of network links. $V_{hosts} \subset V$ represents host nodes in which demands and facilities can reside. The shortest path between two nodes $p, q \in V$ is denoted by $d(p, q)$. We also use the notation of $d(V', p)$ to denote the shortest path between the closest node in a subset $V' \subset V$ to a node $p \in V$. Moreover, let $B(p, r) = \{q \in V_{hosts}, r \in \mathcal{R}^+ | d(p, q) \leq r\}$ indicates the nodes within distance r to the node p (the points that lie inside or on the ball with center p and radius r). We assume that the distance metric is symetric and satisfies triangle inequality.

Facility. In IMFLP, a facility represent a VNF instance and is uncapacitated. The location of a facility z in the space is identified by the $\gamma(z) \in V_{hosts}$. We use term *open* or install interchangeably for the installation of a facility. Besides, the notation $C(z)$ represents a set of demands that are assigned to a facility z (z's cluster).

Demand. A demand u denotes a unit of service workload of a VM. Similar to a facility, the location of a demand is given by $\gamma(u) \in V_{hosts}$, which is equal to the node that VM resides. We use term *arrive* to denote that a new demand from a VM should be served. We also assume that each VM has a correct number of demands.

Allowed Actions. In IMFLP following actions are allowed:

- A *facility* can be *opened* in any node $p \in V_{hosts}$ at any time by paying the installation cost $f \in \mathcal{R}^+$. A facility also can *migrate* to another location with the migration cost $k \in \mathcal{R}^+$. we assume that $k < f$. Moreover, a facility can be closed at any time, and its installation cost is refunded. However, if any demand is assigned to that facility, they should switched to a new facility and for each switch, the certain amount of cost as described next is payed.
- A *demand* is allowed to *arrive* and *leave* at any time in any node $p \in V_{hosts}$. The migration of a VM can be modeled by leaving of its demands and their arrivals in the destination node. Furthermore, a demand assignment can be *switched* to another facility by paying the switch cost $h \in \mathcal{R}^+$. We assume that $h \leq k < f$.

Additional notation. Please note, for a demand u and a facility z, instead of $d(\gamma(z), \gamma(u))$ we simply use $d(z, u)$ to represent their distance. In addition, we define $(x - y)_+ = \max(0, x - y)$ for $x, y \in \mathcal{R}^+$.

The model is described as follow. Upon arrival or departure of a demand u_t at time t (the input of our model), a new facility ω or a set of facilities can be opened, closed, or migrated. Likewise, a subset of demands can be switched to other facilities. Therefore, the following costs are defined at time t:

1) **Total installation cost ($\mathcal{C}_{ins}$)** is the cost of installation of a set of facilities F_t at time t.
$$\mathcal{C}_{ins} = |F_t| f \tag{1}$$
Here, $|F_t|$ denotes the number of facilities.
2) **Total operational cost ($\mathcal{C}_{op}$)** represents the operational cost of a set of facilities F.
$$\mathcal{C}_{op} = g \sum_{z \in F_t} \sum_{u \in C(z)} d(u, z) \tag{2}$$
As shown in equation 2, this cost is defined based on the shortest paths between facilities and their assigned demands.
3) **Total migration cost ($\mathcal{C}_{mig}$)** is the cost of migration of a set of facilities since start time until time t.
$$\mathcal{C}_{mig} = k \sum_{i=2}^{t} \sum_{z \in F_i} |\gamma_{i-1}(z) \neq \gamma_i(z)| \tag{3}$$
In equation 3, $\gamma_i(z)$ represents the location of facility z at time i. Please note that term $|\gamma_{i-1}(z) \neq \gamma_i(z)|$ is 1 if $\gamma_{i-1}(z) \neq \gamma_i(z)$, otherwise it is 0.
4) **Total switch cost ($\mathcal{C}_{sw}$)** denotes the switch cost of a set of demands L_t at time t.
$$\mathcal{C}_{sw} = h \sum_{i=2}^{t} \sum_{u \in L_i} |\phi_{i-1}(u) \neq \phi_i(u)| \tag{4}$$
Here, $\phi_i(u)$ represents the facility that demand u is assigned to at time i. Note that $|\phi_{i-1}(u) \neq \phi_i(u)|$ is equal to 1 if $\phi_{i-1}(u) \neq \phi_i(u)$, otherwise it is 0.

The objective of the optimization problem in the IMFLP formulation is to minimize the overall cost ($\mathcal{C}_{overal}$) as defined in equation 5.

$$\mathcal{C}_{overal} = \mathcal{C}_{ins} + \mathcal{C}_{op} + \mathcal{C}_{mig} + \mathcal{C}_{sw} \tag{5}$$

The IDPSP problem can be reduced to the optimization problem defined in the IMFLP model. This optimization problem is NP-hard (facility location problem is NP-hard, and our online model is even more complicated than original problem). Motivated by this observation, we developed an online algorithm for IMFLP model.

III. Adaptive Facility Location (AFL)

In this section, we propose our solution, *Adaptive Facility Location (AFL)*, for the optimization problem introduced in the IMFLP model. We introduce two novel algorithms that use the simple idea of profit and loss for handling a demand arrival and a demand departure.

However, before describing our model, we justify our selection over other candidate approaches. In the area of SDN, some of ubiquitous approaches for modeling the optimization problems are the linear programming [28, 49], simulated annealing [48, 36], and Markov approximation [30, 41]. The linear programming approach solves an offline problem, and is not descriptive enough to model the dynamicity and online nature of these kind of problems. In addition, the linear programming is known that is slow. To deal with the dynamic nature of these optimization problems, simulated annealing and markov approximation are used. In the simulated annealing techniques, at each step again an offline problem is defined, and known to be trapped in the local minimums, and might suffer from the bad initial state. Finally, Markov chain techniques might also affected from bad initial state and slow convergence to the steady state.

A. Demand Arrival

Two functions namely, *migration potential* and *installation potential* are defined to represent how far facilities and assignments of demands are from the *optimal* or *stable* configuration, and how much *profit* is gained by the installation or migration of a facility, respectively. Then by comparing with the cost of certain *actions* (the loss), AFL decides which action is applied.

Installation potential function (Pot_{ins}) is defined as equation 6. This function represents how much of the current cost can be reduced by installation of a facility at a node $p \in V_{hosts}$. In this equation, u_t denotes a new arrived demand at time t. F_{t-1} and L_{t-1} represent a set of opened facilities and demands at time $t-1$ just before arrival of u_t. The first term computes the profit of the situation where u_t is assigned to a facility that can be installed at node p against when u_t is assigned to the closest facility in F_{t-1}. The second term shows that if some demands are switched to a facility that can be installed at node p, how much the operational cost of related to these demands will reduce (recall that each switch incurs switch cost h).

$$Pot_{ins}(p) = g.\Big(\big(d(F_{t-1}, u_t) - d(p, u_t)\big)_+ + \sum_{v \in L_{t-1}} \big(d(\phi(v), v) - d(p, v) - h\big)_+\Big) \tag{6}$$

Migration potential function (Pot_{mig}) is defined in equation 7. This function describes how much the migration of a facility z from its current location to a node p reduces its total operational cost when a new arrived demand u_t is assigned to z as well. This function can be interpreted in another sense as well. Each demand $v \in C_{t-1}(z)$ attempts to reduce its cost by pulling facility z toward its location $\gamma(u_t)$. If a new arrival demand u_t will be assigned to z, u_t also tries to pull facility z toward itself. The potential function $Pot_{mig}(z, p)$ represents how much z becomes more stable by migration form $\gamma(z)$ to p. In other words, z is close enough to each demand $v \in C_{t-1}$ and more closer to u_t in comparison to facilities F_{t-1} including z itself.

$$Pot_{mig}(z, p) = g.\Big(d(F_{t-1}, u_t) - d(p, u_t) + \sum_{v \in C_{t-1}(z)} \big(d(z, v) - d(p, v)\big)\Big) \tag{7}$$

Algorithm 1 shows AFL algorithm (for the sake of simplicity, we drop t subscript, but we insist that the presented algorithm is run at time t). By exploiting the aforementioned functions, AFL attempts to improve the current placement of facilities and current assignment of demands. Upon arrival of a new demand u, AFL considers three actions:

1) **Installation action**: Installation of one new facility in the best place with the best possible switches.
2) **Migration action**: Migrating one of the existing facilities (the best one) without any demand switches and assigning u to this facility.
3) **Assignment action**: Assigning u to the nearest existing facility.

As shown in algorithm 1, AFL computes the installation and migration potentials. By comparing the the computed values, AFL applies the best action. For the installation action, AFL calculates the installation potential Pot_{ins} for every point p in the distance of f from u ($B(u, f)$). AFL selects the best point ω_{ins}, which maximizes the Pot_{ins}. If AFL decided to apply this action, it switch the neighbor demands to ω_{ins}, if this switches reduce the service cost and the deducted service cost is bigger than switch cost h.

For the migration action, AFL computes the migration potential Pot_{mig} for each facility z and for each point p in the space V_{hosts}. Eventually, AFL chooses the best facility ω_{mig} to migrate to point ρ that maximizes Pot_{mig}.

Ultimately, AFL decides which action is applied. The installation action is considered first. If it is beneficial ($Pot_{ins}(\omega_{ins}) - f > 0$), and its profit is greater than best migration action ($Pot_{mig}(\omega_{mig}, \rho) - k$), AFL applies the installation action. Otherwise, the best migration action is considered. If this migration is beneficial ($Pot_{mig}(\omega_{mig}, \rho) - k > 0$),

Algorithm 1 AFL-Demand Arrival

$F \leftarrow \emptyset; L \leftarrow \emptyset;$
for all new demand u **do**
  $L \leftarrow L \cup \{u\};$
  $\rho_{ins} \leftarrow \arg\max_{p \in B(u,f)}\{Pot_{ins}(p)\};$
  $p_{ins} \leftarrow Pot_{ins}(\rho_{ins});$
  $\omega_{mig}, \rho_{mig} \leftarrow \arg\max_{z \in F, p \in V_{hosts}/\{\gamma(z)\}}\{Pot_{mig}(z,p)\};$
  $p_{mig} \leftarrow Pot_{mig}(\omega_{mig}, \rho_{mig}, u);$
  if $(p_{ins} - f > 0) \wedge (p_{ins} - f \geq p_{mig} - k)$ **then**
    $\omega_{ins} \leftarrow$ open a facility at ρ_{ins}
    $F \leftarrow F \cup \{\omega_{ins}\}$
    Switch facility of each demand $v \in L/\{u\}$ if $d(\phi(v), v) > d(\rho_{ins}, v) + h$;
    Assign u to the nearest facility;
  else if $p_{mig} - k > 0$ **then**
    Assign u to ω_{mig};
    Migrate ω_{mig} to point ρ_{mig};
  else
    Assign u to the nearest facility;
  end if
end for

AFL applies this action. Otherwise, it assigns u to the nearest facility.

B. Demand Departure

Similar to the case of demand arrival, AFL defines closing potential and migration potential functions to represent how far the current configuration of a facility whose demand departures is from the stable configuration. Let u'_t denotes a demand departuring at time t, and $z = \phi(u'_t)$ represents the facility to which u'_t was connected at time $t-1$ just before departure.

Closing potential function (Pot_{cls}) is defined in equation 8. This function denotes the profit of *closing* a facility and switching its demands to the closest facilities.

$$Pot_{cls}(z) = g \sum_{v \in C_{t-1}(z)/\{u'_t\}} \big(d(z,v) - d(F_{t-1}, v) - h\big) \quad (8)$$

Migration potential fucntion (Pot'_{mig}) for the departure of a demand is defined by equation 9. It can be interpreted exactly same as the migration potential for a demand arrival.

$$Pot'_{mig}(z,p) = g \sum_{v \in C_{t-1}(z)/\{u'_t\}} \big(d(z,v) - d(p,v)\big) \quad (9)$$

AFL for the departure considers two actions:

1) **Closing action**: Closing facility z and assigning each of its demands to the closest facility in F_{t-1}/z.
2) **Migration action**: Migration of facility z to another location to serve $C_{t-1}(z)/u'_t$ more efficiently.

Algorithm 2 represents AFL's algorithm for handling a demand departure. For the sake of simplicity, we omit subscript t from the notation. AFL computes the closing potential of facility z serving u' and the best migration potential. First, AFL considers the closing action. If closing z is beneficial $(p_{cls} + f > 0)$ and is more profitable than migration action $(p_{cls} + f \geq p'_{mig} - k)$, AFL applies this action. Otherwise, the migration of z is considered, and if this action is profitable $(p'_{mig} - k > 0)$, AFL migrates z to ρ_{mig}. If none of closing and migration actions are beneficial, AFL only remove demand u' from the list of demands.

Algorithm 2 AFL-Demand Departure

$z \leftarrow \phi(u');$
$p_{cls} \leftarrow Pot_{cls}(z);$
$\rho_{mig} \leftarrow \arg\max_{p \in V_{hosts}}\{Pot'_{mig}(z,p)\};$
$p'_{mig} \leftarrow Pot'_{mig}(z, \rho_{mig});$
if $(p_{cls} + f > 0) \wedge (p_{cls} + f > p'_{mig} - k)$ **then**
  Switch facility of each demand $v \in C(z)/\{u'\}$ to the closest facility in $F/\{z\}$;
  Close facility z;
  $F \leftarrow F/\{z\}$;
else if $p'_{mig} - k > 0$ **then**
  Migrate z to point ρ_{mig};
end if
$L \leftarrow L/\{u'\}$

IV. Experiments

We evaluated the effectiveness of our placement algorithm in several simulation studies. We implemented our AFL algorithm in a discrete event simulator and compared it to other five popular algorithms namely: FFL [19], AFL [18], OPTFL [17], RFL [38], and SNFL [20]. The details of these algorithms will be discussed in section V. The OPTFL, AFL, and FFL algorithms have certain input parameters. We ran these algorithms for miscellaneous values of parameters and did not observe substantial difference. Ultimately, their input parameters were set to the values suggested by their authors, specifically for OPTFL $\alpha = 10$ [17], for AFL $\alpha = 18, \beta = 8.0, \psi = 4.0$ [18], and finally for FFL $x = \frac{19}{8}$ [19].

In the last decade a tremendous research has been done to search for an efficient and inexpensive data center networks (DCN) architecture. Several architectures like fat-trees [3], VL2 [24], Portland [39], BCube [25] and DCell [26] have been proposed to address different challenges of current DCN architectures such as scalability, agility, and reconfigurability. For the experiment, we select Al-Fares et al. fat-tree [3] architecture. This architecture is one of the well known DCN architectures [27] [37] [7]. Fat-trees are more scaleable and reliable than conventional tree-based architectures. This topology allows us to leverage identical cheap commodity switches in the all communication layers. In the theory, the over-subscription ratio of this rearrangeable architecture is $1:1$, which means that this architecture is non-blocking; however, in the practice preventing packet reordering might make it difficult to guaranty non-blocking network. The fat-tree topology proposed by [3] is a k-ary tree in which k denotes number of ports and number of pods. This topology

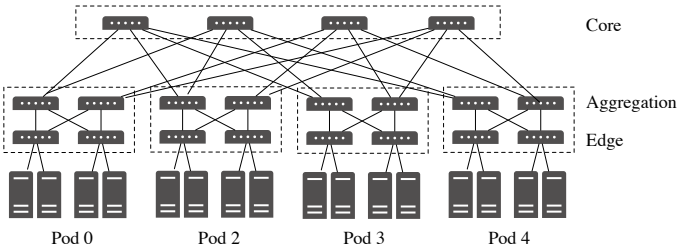


Fig. 2. The fat-tree architecture for 4 pods ($k = 4$)

connects homogeneous switches with the same number of k ports. As depicted in Figure 2, the Al-Fares's fat-tree consists of three switch layers. At the highest level, there are $(\frac{k}{2})^2$ core-switches. Each core-switch is connected to all k pods (i-th port of a core-switch is connected to the i-pod). A pod contains k switches ($\frac{k}{2}$ aggregation-switches and $\frac{k}{2}$ edge-switches). At the second level, aggregation switches are connected to $\frac{k}{2}$ of core-switches upward and $\frac{k}{2}$ edge-switches downward. Furthermore, each aggregation-switch is only connected to edge-switches that are in the same pod. At the third level, edge-switches are linked to the $\frac{k}{2}$ hosts dipping and $\frac{k}{2}$ aggregation-switches mounting. There are $\frac{k^3}{4}$ hosts which are located in the leaves of this architecture. For all experiments, the oversubscribing ratio was set to $1:1$, which means that this architecture is non-blocking.

The demands are generated randomly (only in the leaves) from the uniform and normal distributions. The mean and standard deviation parameters of the normal distribution were set to 0.5 and 0.1, respectively, and each generated value was multiplied by the number of leaves and a demand was generated at the position of the result. Moreover, in all experiments, the value of parameter g was set to 1. All algorithms receive one demand at a time and reconfigure the placement of the facilities to serve this demand upon its arrival. The costs of installation, migration, and switch for all algorithms are collected. For each configuration, the average of 10 tests has been reported as the final result. We have conducted three experiments to evaluate the the behavior of AFL under different circumstances.

A. Impact of Number of Demands

In this experiment, the impact of number of demands on the behavior of AFL is examined. As depicted in Figure 3, five tests for 1024, 2048, 3072, 4096, and 5120 number of demands for uniform (Figure 3(a)) and normal distribution (Figure 3(b)) are conducted. We assign 6, 2, 1 for f, k and h, respectively. The idea behind choosing these values is that we assume that the cost of installation of a facility f is always greater than the cost of migration k and switching h, and the cost of migration is equal or greater than the cost of switching. Moreover, The space is the fat-tree with 1024 hosts. For each test, each algorithm receives one demand at a time and returns a placement of facilities.

TABLE I: AFL's Costs

Demands	Facility	Service	Switch	Migration	Total
64	97.30%	2.70%	0.00%	0.00%	584.8
128	95.51%	4.43%	0.04%	0.02%	1667.7
256	92.40%	7.37%	0.15%	0.08%	3551.9
512	87.54%	11.66%	0.42%	0.38%	6525.8
1024	72.19%	24.36%	2.15%	1.30%	12798.7

Note: We omit word *cost* from the headers. For instance, by the *Facility* we mean *Facility Cost*

As shown in Figure 3, the total cost of all algorithms in the uniform distribution are considerably greater than normal case. The reason is that in the uniform case the demands spread in more hosts in comparison to the normal distribution that demands tend to arrive in the middle hosts. As depicted, AFL outperforms all other algorithms in all cases for both distributions. The average of overall costs of AFL is 11.82% and 14.46% lower than the second best algorithm in the case of uniform and normal distributions, respectively.

B. Impact of Number of Hosts

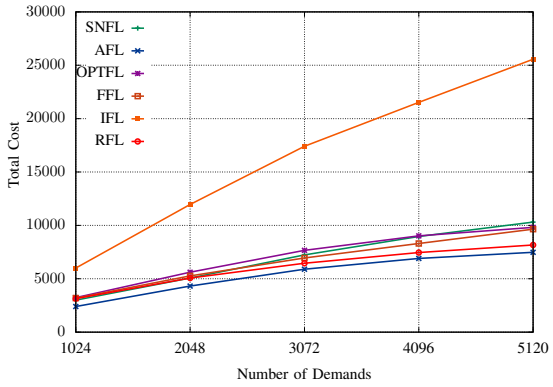
In this experiment, the impact of number of hosts is studied. Fat-trees with 64 ($k = 8$), 250 ($k = 10$), 432 ($k = 12$), 686 ($k = 14$), and 1024 ($k = 16$) hosts are generated and employed as the space for each test. In each test, 1024 demands are generated from uniform and normal distributions. Similar to the previous experiment, values of f, k, and h are set to 6, 2, and 1, respectively.

Figure 4 depicts the results of this experiment. Figures 4(c) and 4(d) represent the results for the uniform and normal distribution, respectively. Similar to the previous experiment, the total costs in the uniform case is noticeably greater than the normal case. As shown, AFL outperforms the other algorithms in both distributions and in all cases. The total cost of AFL is lower than the second best algorithm by the average of 14.16% and 15.66% for the uniform and normal distributions, respectively.

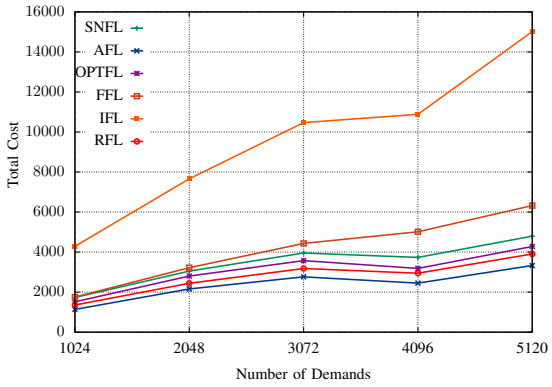
The different costs of AFL for the uniform generated demands are shown by table I. Facility and service costs are the most significant part of the overall cost. By increasing the number of demands, the migration and switching costs increase. In the case of 64 demands, AFL does not migrate or switch, however, when the number of demands increase, AFL migrates certain facilities and switch some of demands in order to reduce the total cost. For instance, the switch and migration costs is 3.45% of the total cost for the 1024 demands. It means that AFL by paying small amount of migration and switch cost saves a significant amount of the facility and service cost. In the case of 1024 points, AFL pays 2914.2 lower than the best second algorithm by paying extra 441.5 migration and switch cost in the average.

C. Impact of Cost Parameters

In this experiment, the impact of the costs parameters (f, k, and h) is investigated. For all tests, the space is fixed to the fat-tree with 1024 hosts (16-ary tree). Note that in this k-ary tree, the maximum distance between two points is 6 (Please note that we fixed the value of g to 1 in all

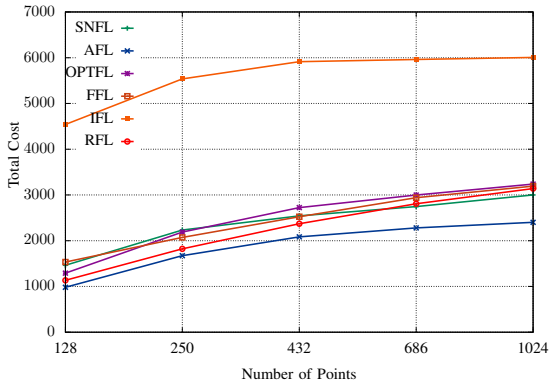


(a) Uniform Distribution

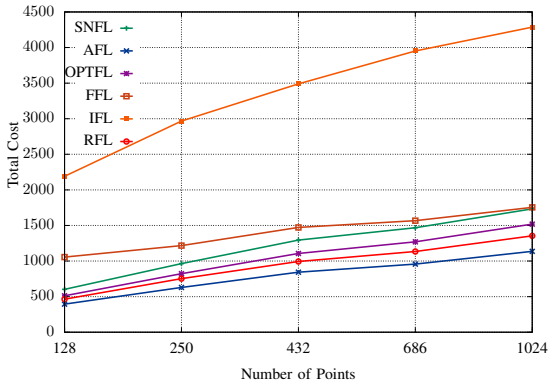


(b) Normal Distribution

Fig. 3. Impact of Number of Demands



(c) Uniform Distribution



(d) Normal Distribution

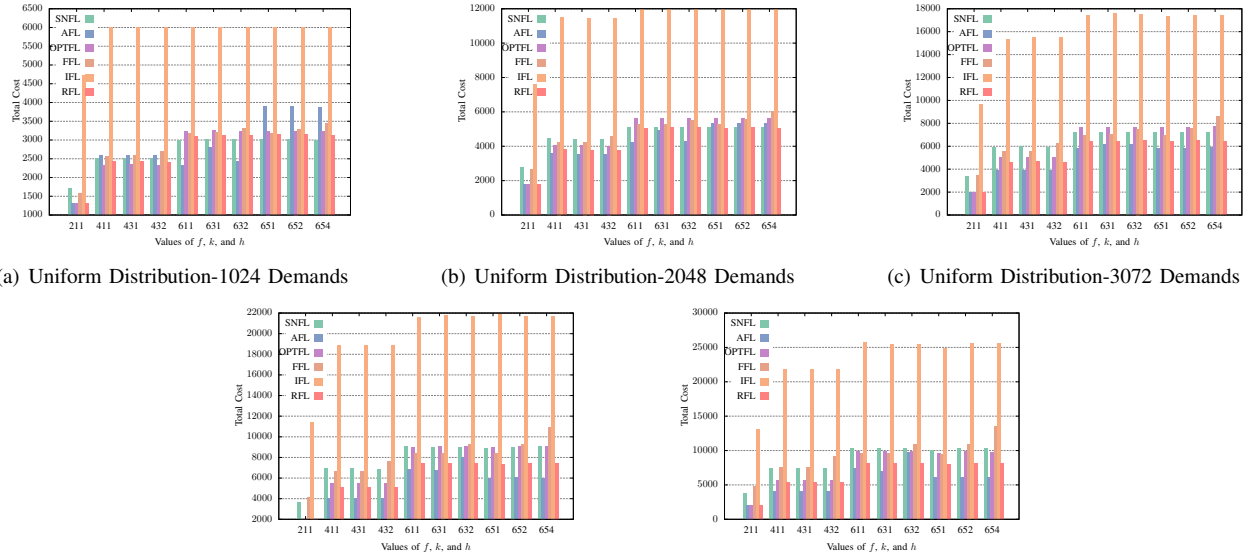
Fig. 4. Impact of Number of Hosts

experiments). Similar to the previous experiment, demands and facilities are located in the hosts. In particular, we vary the cost of installation, switch, and migration to investigate the impact of these parameters on the performance of our algorithm compared to others. We strictly specify that the cost of switching h to be always less or equal than the cost of migration k but cannot exceed the facility installation cost f (i.e, $f < k \leq h$). We run several tests for low, medium, and high values of f, k and h. Specifically for the facility cost f, $2, 4, 6$ are considered as low, medium and high values, respectively. For the migration cost k, the values $1, 3, 5$ and for the switch cost h, the values $1, 2, 4$ are selected as the low, medium and high values, respectively. Ultimately, 10 different configuration of values for the cost parameters are examined. Furthermore, we select the number of demands from 1024, 2048, 3072, 4096, 5120 and randomly place them on leaves based on normal and uniform distribution. Figure 5 shows the overall cost of our algorithm compare to others when changing the installation, switch, and migration cost parameters.

Figures 5(a), 5(b), 5(c), 5(d), and 5(e) represent the results of tests for 1024, 2048, 3072, 4096, and 5120 demands, respectively. As shown, for all configuration of cost values in all tests AFL outperforms the other algorithms except for the three configurations of cost values, 651, 652, and 654 in Figures 5(a), 5(b). However, as can be seen in figures 5(c), 5(d), and 5(e), by increasing the number of demands, AFL again outperforms the other algorithms in these configurations as well. It seems that by increasing the number of demands, AFL converges to the more stable configuration and performs more efficient.

D. Evaluation of Demand Departure

In this experiment, we examine the behavior of our solution for demand departures. Because we did not find any online algorithm in the literature considering demand departures, we compared our algorithm with a famous offline facility location greedy algorithm with approximation ratio of 1.61 [32]. The same configuration as experiments for the impact of number of demands and number of hosts (sections IV-A, IV-B) are used. A fat-tree with 1024 hosts, and values of 6, 2, 1, and 1 for parameters f, k, h, and g, respectively.



(a) Uniform Distribution-1024 Demands

(b) Uniform Distribution-2048 Demands

(c) Uniform Distribution-3072 Demands

(d) Uniform Distribution-4096 Demands

(e) Uniform Distribution-5120 Demands

Note: Please note that x-axis in the figures represent different values for f, k, h as the first, second, and third digit, respectively. For instance, 652 represents 6, 5, and 2 for f, k, and h, respectively.

Fig. 5. The total costs of the algorithms for the different values of f, k, and h for 1024 leaves

V. Related Works

A. Existing Systems

Network Function management solutions in the existing literature can be classified in to two separate groups. 1) Systems that deal with NFs that are deployed on pre-designated static hardware. These include systems such as CoMB [45], SIMPLE [42], xOMB [5] and PLayer [33]. 2) Systems that deal with VM based NF deployments, such as Stratos [23].

Static NF deployments are a step up from the traditional NFs, and introducing software based NFs within pre-placed commodity or specialized hardware. This gives such NFs the ability to use the best of both software and hardware world: multiple NFs can co-exist on the same high speed hardware and work in a coordinated manner to provide superior performance [45]. Since the NF itself is in software, it is easy to update and maintain. The hardware can also evolve independently of the software as the hardware and servers on which the NFs are hosted can be upgraded and replaced. This comes at price - the location of the NF, due to its rigid placement, might not always be ideal. The demand for NFs is not always uniformly distributed with in the data center [23].

B. Facility Location Problem

Facility Location Problem (FLP) is one of the well-known problems in the location theory. This classical optimization problem is concerned with optimal locations of certain facilities to minimize the cost of providing service to demands [46] with the offline settings. This problem is known to be NP-Hard, and several approximation algorithms has been developed for this problem [46]. The best known algorithm is proposed by Li et al. [35] and achieves 1.448 approximation ratio. In addition, several models of this problem with offline settings have been defined in the litereture. Farahani and Hekmatfar [15] provided extensive review of different models of offline FLP, and Boloori Arabani and Farahani [10] overviewed the dynamic models.

TABLE II: Algorithms for OnFLP and IncFLP models

Algorithm	Competitive Ratio		Time complexity [a]
	Adversarial	Random	
RFL	$O(\log n)$[b]	8	$\Omega(\log m)$[c]
OPTFL	$O(\frac{\log n}{\log\log n})$	-	$O(m^2 + \log d_{max})$[d]
SNFL	$4\log n + 1 + 2$	-	$O(m\|M\|\|F\|)$
IFL	$O(1)$	$O(1)$	$O(mm'\|F_{max}\|)$[e]
FFL	14	$O(1)$	$O(m\|F_{max}\|)$

[a] The complexity of processing m^{th} demand
[b] n is the number of all demands
[c] The number of demands at time t
[d] d_{max} is maximum distance in the space
[e] F_{max} denotes the maximum number of facilities opened by the algorithm

Unfortunately, none of the above models are not applicable for our problem, becuase they do not consider the online nature of the problem. Hence, we focus on online models of FLP. Fotakis [21] overviewed the online models of FLP and identified two major models of online version of FLP, namely *Online Facility Location Problem (OnFLP)* and *Incremental Facility Location Problem (IncFLP)*. In addition, some of the other relaxed version are discussed here. Table II represents the well-known algorithms, and their competitive ratios for all models.

1) Online Facility Location Problem: Meyerson et. al. [38] for the first time designed the *Online Facility Location*

Problem (OnFLP). In OnFLP, demands come one at a time and each demand is irrevocablely assigned to a facility upon its arrival; however the location of demands and facilities are not change during the time. Meyerson also proved that there is no algorithm that can be constant competitive against an adversary.

a) Random Facility Location Problem (RFL): Meyerson also proposed the first algorithm for OnFLP [38] called *Random Facility Location (RFL)*. RFL is pretty straightforward for the uniform facility cost. Upon arrival of each new demand u_t, RFL opens a facility with probability $\min\{1, \frac{d(F_{t-1},u_t)}{f}\}$ in location $\gamma(u_t)$.

b) Optimal Facility Location Problem (OPTFL): The first deterministic algorithm for OnFLP is *Optimum Facility Location (OPTFL)* proposed by Fotakis [17], which achieves to the optimum competitive ratio for OnFLP. OPTFL defines the unsatisfied demands L that contains demands no having contributed in opening a new facility. Each $v \in L$ at time t contributes in opening a facility by $d(F_{t-1}, v)$. OPTFL marks each new arrived demand u_t as unsatisfied and appends u_t to L. $S = B(u_t, \frac{d(F_{t-1},u_t)}{\alpha} \cap L)$ is a set of unsatisfied demands that are close to u_t. The potential function of S is defined by $P(S) = \sum_{v \in S} d(F_{t-1}, v)$. If $P(S) < f$, OPTFL opens no facility and assign u_t to the nearest facility. If $P(S) \geq f$, then the algorithm opens a new facility in a location $\omega \in S$ and removes S from L ($L = L/S$). The location ω is the center of the smallest radius ball $S' \subseteq S$ whose potential $P(S') \geq \frac{1}{2}P(S)$.

c) Simple Non-Uniform Facility Location (SNFL): Simple Non-Uniform Facility Location (SNFL) [20] uses the same idea of OPTFL [17], but this algorithm defines the potential function in a different way. There are no unsatisfied demands, and for each point z (either demand or facility) at time step t, the potential function is $p(z) = \sum_{v \in L}(d(F_{t-1}, v) - d(z, v))_+$ in which L denotes the set of previous demands and also the new arrived demand u_t. Upon arrival of demand u_t, SNFL adds u_t to L, computes potential $p(z)$ for all $z \in M$, and finds the point ω maximizing $p(\omega) - f_\omega$. If $p(\omega) > f_\omega$, SNFL opens a new facility at ω ($F_t = F_{t-1} \cup \{\omega\}$) and assigns u_t to ω. If $p(\omega) \leq f_\omega$ then SNFL does not open any new facility and assigns u_t to the nearest existing facility.

2) Increamental Facility Location Problem (IncFLP): Motivated by the framework of incremental clustering [11] and incremental k-median [12], *Incremental Facility Location Problem (IncFLP)* is developed. In contrast to ONFLP, two existing facilities and corresponding demands clusters can be merged in this model. A merge rule procedure determines whether two facilities will be merged. When a new demand u_t arrives, the algorithm applies a facility-opening rule and a merge rule to determine whether a new facility opens, and two existing facilities will be merged.

a) Incremental Facility Location (IFL): Incremental Facility Location (IFL) [18] is the first algorithm proposed for the IncFLP model. IFL introduced a new concept merge ball $B(\omega, m(\omega))$ for each facility ω, in which $m(\omega)$ is the merge-radius. IFL also defines $C(\omega)$ and $Init(\omega)$. In fact, $Init(\omega)$ is a set of demands initially assigned to a just opened facility ω, and $C(\omega)/Init(\omega)$ are demands that initially are assigned to other facilities different from ω, and gradually are assigned to ω by the merge rule ($m(\omega) \subseteq C(\omega)$). Moreover, IFL makes sure that no merge operation increase the total service cost of the demands in $Init(\omega)$ intensely. Hence, it keeps $m(\omega)$ decreasing by maintaining invariant $|Init(\omega) \cup B(\omega, \frac{m(\omega)}{\psi})|.m(\omega)$ lower than or equal to βf, in which ψ and β are appropriate positive constant integers.

3) Relaxed incremental Facility Location Problem (RFLP): Fotakis [19] suggested another model for FLP that is similar to IncFLP, though the demands can be reassigned to a nearest facility. We call this model Relaxed incremental Facility Location Problem (RFLP).

a) Fast Facility Location (FFL): Fast Facility Location (FFL) [19] introduces the final distance function used for the merge rule. The final distance for each facility ω and a point p is defined as $g(\omega, p) = d(\omega, p) + 2m(\omega)$. In this definition $m(\omega)$ denotes ω's replacement distance. For a point p and a facility set F, the replacement distance is $g(F, p) = \min_{\omega \in F}\{g(\omega, p)\}$. FFL works as follows. When a new demand u_t arrives, the algorithm computes $\delta = g(F_{t-1}, u_t)$ and opens a new facility ω in the location of u_t with probability of $\min\{1, \frac{\delta}{xf}\}$. If ω opens, the replacement radius $m(\omega)$ is set to $\frac{\min\{xf, g(F_{t-1},u_t)\}}{6}$. Then, FFL considers every facility z which ω is inside $B(z, m(z))$, and merges z with ω. Notice that the demands assigned to these facilities will be reassigned to the nearest facility, not necessarily to ω.

4) Comparison of IMFLP with the other works: It is worth noting that our formulation of IMFLP is different from OnFLP, because assignment decisions and locations of facilities can be changed. Furthermore, the difference the incremental version [19] is that switch a demand to any other facility is allowed in the IMFLP model. Regarding the RFLP, switches are allowed but is free of charge. Finally, [14, 4] are the most similar model to ours, however; they assumed that the number of demands is fixed and known in advance, and they just change their locations.

VI. Conclusion and Future Works

We formulated the Intrusion Detection and Prevention Systems Placement (IDPSP) problem for cost-effective support as services in Cloud-based environment. Incremental Mobile Facility Location Problem (IMFLP) model was proposed to study this problem, and Adaptive Facility Location (AFL) solution was presented and evaluated for solving the optimization problem in this model. For the future works, we plan to implement this system as a real cloud service. Moreover, we intend to improve upon the proposed placement model. Certain constraints have been ignored in the formulation of the optimization problem in the IMFLP model for the sake of simplicity. We plan to add these constraints in the future model to make IMFLP model more applicable.

References

[1] ETSI Network Functions Virtualisation Introductory White Paper. https://portal.etsi.org/nfv/nfv$_white_paper.pdf$.

[2] OpNFV - White Paper.

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture, 2008.

[4] Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Transactions on Algorithms (TALG)*, 13(2):21, 2017.

[5] James W Anderson, Ryan Braud, Rishi Kapoor, George Porter, and Amin Vahdat. xomb: Extensible open middleboxes with commodity servers. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 49–60. ACM, 2012.

[6] Junaid Arshad, Paul Townend, and Jie Xu. An automatic intrusion diagnosis approach for clouds. *International Journal of Automation and Computing*, 8(3):286–296, 2011.

[7] Md Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *Communications Surveys & Tutorials, IEEE*, 15(2):909–928, 2013.

[8] Md Faizul Bari, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. Dynamic controller provisioning in software defined networks. In *CNSM*, pages 18–25, 2013.

[9] Alireza Boloori Arabani and Reza Zanjirani Farahani. Facility location dynamics: An overview of classifications and applications. *Computers & Industrial Engineering*, 62(1):408–420, February 2012.

[10] Alireza Boloori Arabani and Reza Zanjirani Farahani. Facility location dynamics: An overview of classifications and applications. *Computers & Industrial Engineering*, 62(1):408–420, 2012.

[11] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635. ACM, 1997.

[12] Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 1–10. ACM, 2001.

[13] András Császár, Wolfgang John, Mario Kind, Catalin Meirosu, Gergely Pongrácz, Dimitri Staessens, Attila Takács, and J Westphal. Unifying cloud and carrier network. *Proceedings of. DCC, Dresden, Germany, to appear Dec*, 2013.

[14] David Eisenstat, Claire Mathieu, and N Schabanel. Facility location in evolving metrics. *arXiv preprint arXiv:1403.6758*, pages 1–12, 2014.

[15] Reza Zanjirani Farahani and Masoud Hekmatfar. *Facility location: concepts, models, algorithms and case studies*. 2009.

[16] Björn Feldkord and Friedhelm Meyer auf der Heide. The mobile server problem. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 2017.

[17] Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 14186:637–652, 2003.

[18] Dimitris Fotakis. Incremental algorithms for Facility Location and k-Median. *Theoretical Computer Science*, 361(2-3):275–313, 2006.

[19] Dimitris Fotakis. Memoryless facility location in one pass. *STACS 2006*, pages 608–620, 2006.

[20] Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *Journal of Discrete Algorithms*, 5(1), 2007.

[21] Dimitris Fotakis. Online and incremental algorithms for facility location. *ACM SIGACT News*, 42(1):97–131, 2011.

[22] A Gember, R Viswanathan, C Prakash, R Grandl, J Khalid, S Das, and A Akella. Opennf: Enabling innovation in network function control. Technical report, University of Wisconsin-Madison, 2014.

[23] Aaron Gember, Robert Grandl, Ashok Anand, Theophilus Benson, and Aditya Akella. Stratos: Virtual middleboxes as first-class entities. *UW-Madison TR1771*, 2012.

[24] By Albert Greenberg, James R Hamilton, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, A Maltz, Parveen Patel, Sudipta Sengupta, Albert Greenberg, Navendu Jain, and David A. Maltz. VL2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM 2009 Conf. Data Commun.*, volume 09, pages 51–62, 2009.

[25] Chuanxiong Guo, G Lu, Dan Li, Haitao Wu, and X Zhang. BCube: a high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM 2009 Conf. Data Commun.*

[26] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers, 2008.

[27] Ali Hammadi and Lotfi Mhamdi. A survey on architectures and energy efficiency in Data Center Networks, 2014.

[28] Qichao He, Ying Wang, Wenjing Li, and Xuesong Qiu. Traffic steering of middlebox policy chain based on sdn. In *Integrated Network and Service Management (IM), IFIP/IEEE Symposium on*. IEEE, 2017.

[29] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM, 2012.

[30] Huawei Huang, Song Guo, Jinsong Wu, and Jie Li. Service chaining for hybrid network function. *IEEE Trans. on Cloud Computing*, 2017.

[31] Jinho Hwang, KK Ramakrishnan, and Timothy Wood. Netvm: high performance and flexible networking using virtualization on commodity platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[32] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 731–740, New York, NY, USA, 2002. ACM.

[33] Dilip A Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 51–62. ACM, 2008.

[34] Md Tanzim Khorshed, ABM Ali, and Saleh A Wasimi. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation Computer Systems*, 28(6):833–851, 2012.

[35] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

[36] Jiaqiang Liu, Yong Li, Ying Zhang, Li Su, and Depeng Jin. Improve service chaining performance with optimized middlebox placement. *IEEE Transactions on Services Computing*, 10(4):560–573, 2017.

[37] Yang Liu, Jogesh K Muppala, and Malathi Veeraraghavan. A Survey of Data Center Network Architectures.

[38] Adam Meyerson. Online facility location. *... of Computer Science, 2001. Proceedings. 42nd ...*, pages 0–5, 2001.

[39] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, Amin Vahdat, and Radhika Niranjan Mysore. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM 2009 Conf. Data Commun.*

[40] VT Paschos. *Paradigms of Combinatorial Optimization: Problems and New Approaches*. 2013.

[41] Chuan Pham, Nguyen H Tran, Shaolei Ren, Walid Saad, and Choong Seon Hong. Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach. *IEEE Transactions on Services Computing*, 2017.

[42] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 27–38. ACM, 2013.

[43] Sebastian Roschke, Feng Cheng, and Christoph Meinel. Intrusion detection in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pages 729–734. IEEE, 2009.

[44] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800, 2007.

[45] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *NSDI*, pages 323–336, 2012.

[46] David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274. ACM, 1997.

[47] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. An overview of ip flow-based intrusion detection. *IEEE Communications Surveys and Tutorials*, 12(3):343–356, 2010.

[48] Wentao Wang, Lingxia Wang, and Fang Zheng. An improved adaptive scheduling strategy utilizing simulated annealing genetic algorithm for data center networks. *KSII Transactions on Internet and Information Systems (TIIS)*, 11(11):5243–5263, 2017.

[49] Jie Zhang, Deze Zeng, Lin Gu, Hong Yao, and Muzhou Xiong. Joint optimization of virtual function migration and rule update in software defined nfv networks. In *GLOBECOM IEEE Global Communications Conference*, pages 1–5. IEEE, 2017.