Latency-aware joint virtual machine and policy consolidation for mobile edge computing

Latency-aware Joint Virtual Machine and Policy Consolidation for Mobile Edge Computing

Thiago A. L. Genez*, Fung Po Tso*, Lin Cui†,
*Department of Computer Science, Loughborough University, LE11 3TU, UK
†Department of Computer Science, Jinan University, Guangzhou, China
Email: {t.a.lopes-genez, p.tso}@lboro.ac.uk; tcuilin@jnu.edu.cn;

***Abstract*—To guarantee an efficient and high-performance environment for mobile devices to perform offloading with low end-to-end delay, it is important to ensure no network policies are violated. In this paper, we explore the simultaneous, dynamic virtual machine (VM) and policy consolidation, and formulate the *Policy-VM Latency-aware Consolidation problem for Mobile Edge Computing*, which is shown to be NP-Hard. We propose the *PL-Edge*, an efficient scheme to jointly consolidate network policies and virtual machines for mobile edge computing to reduce communication end-to-end delays among devices and virtual machines. Our simulation results demonstrate that the proposed PL-Edge can significantly reduces policy-flows end-to-end delay by nearly 45% while adhering strictly to the requirements of network policies.**

I. Introduction

A common approach to overcome ever-increasing computation demand on resource-constraint Internet-of-Things (IoT) devices is to offload the application's computation to remote "Cloud" (i.e. data centres) [1]. Nevertheless, offloading workloads reduces the devices' computational cost at the expense of their communications cost. It has been reported that both application performance and user experience are often greatly impaired because the network latency between the devices and cloud date centres are high and unpredictable [2]. This requires the need for bringing the "Cloud" closer to devices and users. As a result, "cloudlets", micro data centres deployed in close proximity to users, were introduced in mobile networks in a bid to cut the end-to-end latency [3]. This is essentially the emerging of the mobile edge computing (MEC) paradigm, where a small, edge-located clusters of resource-rich servers can be dispersed at the edge of the carrier's network [1].

Similar to cloud data centers, servers in a cloudlet are also virtualised and provisioned as virtual machines (VMs) [4]. We consider a Software Defined Network (SDN) based cellular network architecture [5], [6] to provide efficient and flexible communications paths between VMs and user equipments (UEs), such as mobile or IoT devices. To reap the benefits of using cloudlets in mobile edge computing, base stations (BSs) are connected to cloudlets in order to provide ubiquitous computing resources, in terms of VMs, for UEs. In this case, the presence of the cellular backhaul is paramount to the operation of a cellular cloudlet-based infrastructure [1], [5]. Thanks to the cellular backhaul connectivity, VMs can be migrated between cloudlets to maintain a low latency communication with UEs. Driven by UE movements and by the aim of maintaining a low latency communication, VMs are migrated passing through the cellular backhaul to reach others cloudlet deployed in the network. An unexpected cloudlet overhead or malfunction may also call for a temporary migration of VMs to a nearer cloud using the cellular backhaul [7]. However, similar to the cloud, traffic from/to VMs are normally governed by network policies [8]. This mean that, in many cases, traffic between UEs and VMs will not exactly exchanged directly among them, but it may be forced to across a set of network functions, such as firewall, load balancers, proxies, etc., before reaching the final destination [9].

As UEs may be moving over time, this mean following dynamic VM migrations, network policies will need to be redeployed; otherwise carriers will face several network policy violations, which could lead to severe consequences such as services disruption or network blackout. In this paper we jointly optimise VM and network policy migration to minimise the end-to-end latency among UEs and VMs whilst ensuring that no network policy is violated. To the best of our knowledge, this is the first paper in the literature that tackles network polices in a cloudlet-based cellular network.

This paper explores the simultaneous, dynamic virtual machine (VM) and policy consolidation, and formulates the PL-Edge, a *Policy-VM Latency-aware Consolidation problem for Mobile Edge Computing* to reduce the end-to-end delay (latency) between user equipments (UEs) and virtual machines (VMs) in a cloudlet-based network whilst ensuring that no network policy is violated. By reducing the multiple knapsack problem (MKP) to PL-Edge, we prove its NP-Hardness. To solve PL-Edge efficiently, we propose a two-step heuristic for migrating network policy and VM accordingly. We have evaluated PL-Edge in ns-3, and our results have shown that it can reduce the average latency by 45% while maintaining a constant throughput for policy-flows among UEs and VMs.

This paper is organised as follows. Section II presents related work, while Section III describes the problem model of joint policy and VM consolidation for a mobile cloudlet-based network infrastructure. The proposed PL-Edge is presented in Section IV, while results are discussed in Section V. Final remarks and conclusion are presented in Section VI.

II. Related Work

Hu et al. quantified in [10] the impact of employing cloudlets on cellular networks. They state that running application on cloudlets can improve application's response time and significantly reduce UE's battery consumption. Actually,

offloading applications' workloads to a remote cloud through cellular networks is costly due to the lower bandwidth, higher end-to-end latency and higher battery consumption. To solve these issues, a mobile cloudlet-based network architecture is proposed by Sun and Ansari in [5]. This architecture is aimed at providing seamless and low end-to-end delay between UEs and cloudlets to facilitate the offloading of applications' workloads. The concept of cloudlets to mitigate the end-to-end delay is also presented by Taleb et al. in [4]. They described a concept of *Follow-Me-Cloud*, whereby not only data but also mobile services intelligently should follow their respective users. All these works advanced the concept of bringing the cloud in close proximity to mobile users (via cloudlets), but none of them have mentioned network policies in their design. Network policies are extremely important for any network have efficient data traffic management since it directly impacts the end-to-end latency between any communication nodes.

III. Problem Modelling

A. Problem Notations

*1) **Cloudlet**:* We consider a cloudlet as a rack of servers. Let $C_i = \{V_i, S_i\}$ be the i^{th} cloudlet of the mobile infrastructure, where $V_i = \{v_1, v_2, \ldots\}$ is a set of VMs that are hosted by a set of servers $S_i = \{s_1, s_2, \ldots\}$. Let r_a be the vector that denotes the physical resource requirements of $v_a \in V_i$, such as r_a could have three components that capture three types of physical hardware resources, such as CPU cores, memory size, and I/O operations. Accordingly, the available amount of physical resource of server $s_b \in S_i$ is given by a vector h_b. Thus, to denote that s_b has sufficient physical resource to accommodate v_a, we use the relation $\preceq$ as follows: $\sum_{r_k \in A(s_b)} r_k + r_a \preceq h_b$, where $\sum_{r_k \in A(s_b)} r_k$ is the total requirements of all VMs already hosted by s_j. Assuming n available cloudlets, $\mathcal{C} = \cup_{i=1}^{n} C_i$ is the set of all cloudlets presented in the mobile infrastructure, while $\mathcal{S} = \cup_{i=1}^{n} S_i$ and $\mathcal{V} = \cup_{i=1}^{n} V_i$ are the set of all servers and VMs, respectively.

*2) **Mobile Infrastructure**:* Let $B_i = \{U_i, C_i\}$ be the i^{th} base station of the mobile infrastructure, in which $U_i = \{u_1, u_2, \ldots\}$ is the set of all UEs connected to B_i, and $C_i = \{V_i, S_i\}$ is the B_i's dedicated cloudlet. We assume that C_i is connected to B_i via high-speed fibre. Assuming a mobile infrastructure composed by m base stations, $\mathcal{B} = \{B_1, \ldots, B_m\}$ is the set of all base stations, while $\mathcal{U} = \bigcup_{i=1}^{m} U_i$ is the set of all UEs connected to the carrier's infrastructure. In a real deployment of cloudlets in a cellular network, base stations localised in remote areas will probably share cloudlets deployed nearby the backhaul portion of the network [5]. However, without loss of generality, we assume that cloudlets and base stations are one-to-one correspondence, i.e., $|\mathcal{C}| = |\mathcal{B}|$. Also, for the sake of discussion simplicity, we consider that only one VM is provided for each UE to offload workloads [5], and thus UEs and VMs are also one-to-one correspondence, i.e., $|\mathcal{U}| = |\mathcal{V}|$.

It is important to emphasise the presence of the cellular backhaul is still important to the maintenance of cloudlets. Apart from VM migrations triggered by UE movements, there are many edge computing situations in which VMs migration have to pass through the backhaul as well [7]. For example, an unexpected flash crowd may overload a small cloudlet and make it necessary to temporarily move some parts of the current workload to another cloudlet or the cloud. Or when a cloudlet failure due to a site catastrophe such as rising flood water, a spreading fire, or approaching enemy: the currently-executing VMs can be moved to a safer cloudlet without disrupting the offloading service. In these cases, besides the migration of VMs, the migration of policies is extremely important to keep UE-VM communications alive.

*3) **Middleboxes**:* Service function chaining (SFC) is defined [9] as "*an ordered set of service functions that must be applied to selected packets and/or frames as a result of classification*". SFC requires the placement of service functions (also called *middleboxes* [11]) and the adaptation of traffic-forwarding policies on the underlying network to steer packets through an ordered chain of service components. The best practical use case of SFC in today's mobile network is the deployment of middleboxes among P-GWs (gateways) and external networks (such as the Internet), which is far from the edge of the network, where UEs are localised [1]. However, as we attempt to bring cloud closer to UEs with cloudlets nearby base stations, we also attempt to bring middleboxes closer to UEs in order to shorten the path that policy-flows between UEs and VMs need to transverse necessary middleboxes.

Let $\mathcal{M}$ be a set of middleboxes. Each middlebox $m_i \in \mathcal{M}$ is represented by a 3-tuple $\langle type, state, capacity \rangle$. The property $m_i.type$ defines the service function provided by m_i, such as intrusion detection system (IDS), deep packet inspection (DPI) or even necessary logging/charging carrier's applications. Each middlebox may be available at multiple locations, such as inside the cloudlet or in the backhaul portion of the cellular network. Middleboxes are usually stateful and need process both directions of a session for correctness. The $m_i.state$ is used to store the internal state and processing logic, for m_i, while the $m_i.capacity$ is essentially the throughput of m_i.

*4) **Flows**:* We consider the traffic in the mobile network is routed using a flow-based routing mechanism [6]. Let $\mathcal{F} = \{f_1, f_2, \ldots\}$ be a set of flows, where each flow $f_i \in \mathcal{F}$ is composed by a 5-tuple $\langle src : port, dst : port, rate \rangle$. The $f_i.src$ and $f_i.dst$ specify, respectively, the source and destination of f_i including IP addresses and ports, while $f_i.rate$ indicates the data rate exchanged among $f_i.src$ and $f_i.dst$. For a better management and utilisation of the entire network, flows are usually governed by network policies.

*5) **Policy**:* Let $\mathcal{P}$ be a set of network policies in which each policy $p_i \in \mathcal{P}$ is composed by a 4-tuple $\langle seq, list, len, in, out \rangle$. In a real environment, one policy can be attributed to multiple flows and vice versa. However, for the sake of discussion simplicity, we assume that flows and policies are one-to-one correspondence. For each flow $f_i \in \mathcal{F}$, there is a policy $p_i \in \mathcal{P}$ that governs its path to ensure the middleboxes crossing. The attribute $p_i.seq$ defines the sequence of types of middleboxes that the flow f_i matches p_i should transverse in order, e.g., $p_i.seq = \{IDS, DPI\}$, while

$p_i.list$ indicates the list of deployed middleboxes assigned to p_i, , e.g., $p_i.list = \{IDS_x, DPI_y\}$. The $p_i.len$ denotes the size of this list, while $p_i.in$ and $p_i.out$ represent the first (ingress) and the last (egress) middlebox, respectively. A policy p_i is called *satisfied* if all required middleboxes are correctly allocated to p_i concerning types and order: $p_i.list[k].type == p_i.seq[k], \forall k \in \{1, \ldots, p_i.len\}$. If $p_i = \emptyset$, then all flows matching p_i are not governed by any policies.

B. Policy and VM Consolidation Problem

Since we argue in [8] that treating VM and policy management separately can lead to sub-optimal network utilisation and policy violations, this paper address the UE–VM consolidation in conjunction with the dynamism of policy reconfiguration regarding the network latency minimisation.

We denote A to be an initial allocation of VMs, UEs, and middleboxes. $A(u)$ returns the base station B_j that UE u is currently connected with, while $A(v)$ returns the 2-tuple $\langle C_i, s\rangle$ saying that v is localised in the cloudlet C_i and hosted by a server $s \in S_i$ of C_i. The set of VMs hosted by s is given by $A(s)$, while the set of middleboxes that are allocated to the policy p_k is given by $A(p_k)$, i.e., $p_k.list$. Lastly, $A(m)$ refers to all flow-policies that use the middlebox m on its path. We also denote the variable $x_{u,v}$ that assumes the value of 1 if the VM v "belongs" to the UE u; otherwise, the value of positive infinity is assigned to $x_{u,v}$. Assuming $L_{u \leftrightarrow v}$ as the average end-to-end latency for the communication pair UE u and its VM v, the *Policy-VM Consolidation problem for Mobile Edge Compuing* (PVC-MEC) in this paper can be defined as:

Definition 1. *Given the set of UEs $\mathcal{U}$, VMs $\mathcal{V}$, servers $\mathcal{S}$, policies $\mathcal{P}$, and middleboxes $\mathcal{M}$, we need to find an allocation A that minimises the average end-to-end latency for each UE–VM communication pair presented in the mobile network:*

$$\text{Minimise} \quad \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} L_{u \leftrightarrow v} \cdot x_{u,v} \quad \text{Subject to:}$$

$$\sum_{v \in \mathcal{V}} x_{u,v} = 1 \qquad \forall u \in \mathcal{U} \qquad (C1)$$

$$p_k \text{ is satisfied} \qquad \forall p_k \in \mathcal{P} \qquad (C2)$$

$$A(u) \neq \emptyset \text{ and } |A(u)| = 1 \qquad \forall u \in \mathcal{U} \qquad (C3)$$

$$A(v) \neq \emptyset \text{ and } |A(v)| = 1 \qquad \forall v \in \mathcal{V} \qquad (C4)$$

$$\sum_{v_k \in A(s_j)} r_k + r_v \preceq h_j, \qquad \forall s_j \in \mathcal{S} \qquad (C5)$$

$$\sum_{p_k \in A(m)} f_k.rate \leq m.capacity \qquad \forall m \in \mathcal{M} \qquad (C6)$$

The constraints (C1) determine that each UE u has only one VM v available for offloading purposes, while the constraints (C2) describe that all policy requirements for the flows among u and v on middleboxes traversal are fulfilled. The constraints (C3) and (C4) determine, respectively, that each UE is connected on one base station and each VM is hosted on one server in one cloudlet. The capacity requirements for servers and middleboxes are reinforced by (C5) and (C6), respectively.

Theorem 1. *The PVC-MEC problem is NP-Hard*

Proof. To show the non-polynomial complexity of PVC-MEC, we will show that the multiple knapsack problem (MKP) [12], which decision version has already been proven to be strongly NP-hard, can be reduced to this problem in polynomial time. Considering a special case of the PVC-MEC problem where a mobile network is composed by only two base stations: $B_1 = \{U_1, C_1\}$ and $B_2 = \{U_2, C_2\}$. Both base stations are connected to different switches of the mobile backhaul. Each base station is directed connected to their cloudlet represented by a rack of one server only, i.e., $C_1 = \{V_1, S_1\}$ and $C_2 = \{V_2, S_2\}$, where $S_1 = \{s_a\}$ and $S_2 = \{s_b\}$. The capacity of each servers is equal to n, i.e., $h_a = h_b = n$. Also, considers $2n$ UEs divided in two equal size groups. One group is connected to B_1, while the other group is connected to B_2, e.g. $U_1 = \{u_1, \ldots, u_n\}$ and $U_2 = \{u_{n+1}, \ldots, u_{2n}\}$. As there is one VM for each UE, there are also $2n$ VMs deployed. These $2n$ VMs are also divided two equal size groups, i.e., $V_2 = \{v_1, \ldots, v_n\}$ and $V_1 = \{v_{n+1}, \ldots, v_{2n}\}$. The resource requirement of each VM is equal to 1, i.e., $r_i = 1, \forall i \in \{1, \ldots, 2n\}$. In summary, all VMs of B_1's UEs are hosted into B_2's cloudlet and vice versa.

There are n flows among U_1 and V_2 as well as n flows among U_2 and V_1. All flows are policy flows and they need to traverse three types of middleboxes in order, e.g., X, Y, and Z. There are deployed one X box, one Y box and several Z boxes. The X and Y middleboxes are attached to switches connecting B_1 and B_2, while the Z boxes are deployed within the cloudlets C_1 and C_2. Suppose the capacity of X and Y are enough to accept all flows that require them. A reasonable solution to minimise the network latency of all UE–VM communication pairs is to migrate all VMs localised in C_1 to C_2 and migrate all VMs in C_2 to C_1. After all migrations have been performed, we have: $C_1 = \{V_2, S_1\}$ and $C_2 = \{V_1, S_2\}$. In this new configuration, the PVC-MEC problem becomes to find an appropriate Z box for each UE–VM pair that minimises the end-to-end latency among them.

Consider each flow f_i to be an item, where $f_i.rate$ is the item size. Each Z box is a knapsack with limited capacity. The profit of assigning f_i to each Z box is the network latency. The PVC-MEC problem becomes to find an allocation of all flows to Z boxes that minimises the average latency. Thus, the MKP problem is reducible to the PVC-MEC problem in polynomial time, and hence the PVC-MEC problem is NP-hard.

IV. Policy and Virtual Machine Consolidation for Mobile Edge Computing

This section introduces the *PL-Edge*, a policy-VM latency-aware consolidation scheme for mobile edge computing.

A. VM Migration Latency Reduction Model

Let u and v be an UE and its VM, respectively, and $C_i = \{V_i, S_i\}$ a cloudlet. The current allocation of $v \in V_i$ is given by $A(v) = \langle C_i, s\rangle$, where $s \in S_i$ is a server of C_i. Let $A(v) \rightarrow \langle C_j, \hat{s}\rangle$ represent the migration of v from its current location to a server $\hat{s} \in S_j$ of another cloudlet $C_j = \{V_j, S_j\}$. Since we

do not consider migrations of VMs within the same cloudlet, the set of all servers that are available to host v_i is given by:

$$\mathcal{S}(v) = \left\{ \hat{s} \mid \sum_{v_k \in A(\hat{s})} r_k + r_i \preceq \hat{h}, \forall\ \hat{s} \in \mathcal{S} \setminus S_i \right\} \quad (1)$$

The aim of migrating v from its current cloudlet to another one is to reduce the delay in communication with u. However, we have to analyse if this migration will first bring a latency reduction between s (the server where v is currently hosted) and all ingress & egress middleboxes that all flows among u and v have to across due to policies governance.

By taking the advantage of a SDN-based cellular core network, the end-to-end latency between nodes (such as switches, middleboxes, and cloudlets) can be measured by monitoring tools included in the SDN controller [13]. Let all flows from v to u be governed by policies defined in $P(v,u) \in \mathcal{P}$, and all flows from u to v by policies in $P(u,v) \in \mathcal{P}$. By hosting v, let $U(s)$ be the maximum latency between s and all ingress & egress middleboxes. The $U(s)$ value is calculated as follows:

$$U_v(s) = \max_{\forall p_k \in P(v,u)} (v \xrightarrow{f_k} p_k.in) + \max_{\forall p_{k'} \in P(u,v)} (p_{k'}.out \xrightarrow{f_{k'}} v), \quad (2)$$

where $v \xrightarrow{f_k} p_k.in$ is the latency that f_k has to face from v to the ingress middlebox $p_k.in$ (specified by the policy p_k), while $p'_k.out \xrightarrow{f_{k'}} v$ is the latency that $f_{k'}$ faces from the egress middlebox $p'_k.out$ to v according to p'_k. Note that f_k is a flow from v to u that has to pass through $p_k.in$ as part of its initial route, while $f_{k'}$ is a flow from u to v that across $p'_k.out$ as part of its final route. Assuming that the migration $A(v) \rightarrow \langle C_j, \hat{s} \rangle$ takes place, it is expected that $U_v(\hat{s}) < U_v(s)$.

Migrating a VM also generates data traffic between the source and destination hosts. The amount of traffic depends on the VM's memory size, its page dirty rate, the available bandwidth for the migration and some other hypervisor specific constants [14]. Let $s \xrightarrow{v} \hat{s}$ denote the migration of v from s to $\hat{s}$, the migrate time can be given as follows [14]:

$$U(s \xrightarrow{v} \hat{s}) = M \cdot \frac{1 - \left(\frac{R}{L}\right)^{\gamma+1}}{1 - \left(\frac{R}{L}\right)}, \quad (3)$$

where $\gamma = \min(\lceil \log_{R/L} \frac{T \cdot L}{M} \rceil), \lceil \log_{R/L} \frac{X \cdot R}{M \cdot (L-R)} \rceil)$ is the number of pre-copy cycles, M is the total memory size of v_i, R is the page dirty rate of v_i, and L is the available bandwidth for migration among s and $\hat{s}$. The T and X are parameters of the user-configured hypervisor that represent the minimum required progress for each pre-copy cycle and the maximum time for the final stop-copy cycle, respectively.

Migrating v from s to $\hat{s}$ should not contribute to the increase of the end-to-end latency among u and v; otherwise, the communication among them may become slower. Motivated by the migration $A(v) \rightarrow \langle C_j, \hat{s} \rangle$, the expected *gain* in reduction the latency between the current server s and all ingress & egress middleboxes is defined as:

$$G(s \xrightarrow{v} \hat{s}) = U_v(s) + U(s \xrightarrow{v} \hat{s}) - U_v(\hat{s}) \quad (4)$$

A candidate server $\hat{s}$ is considered to host v if the gain of migrating v to $\hat{s}$ results in a positive balance, i.e., $G(s \xrightarrow{v} \hat{s}) > 0$, since we assume that increasing one unit of the gain collaborates in reducing one unit of the total latency among u and v. The gain of migrating v from s to s is considered to be null, $G(s \xrightarrow{v} s) = 0$, since $U(s \xrightarrow{v} s) = 0$ and $s \notin \mathcal{S}(v)$.

B. Policy Migration Latency Reduction Model

When performing policy migration, to preserve the correctness and fidelity of flows, the destination middlebox must receive the internal middlebox state associated with the migrated flows, while the old middlebox still keeps the internal state associated for the remaining flows. Clearly, the middlebox states must be able to be cloned, shared, moved and merged. To support this feature, we adopt the architecture of OpenNF [15], which is a control plane with carefully designed APIs for managing middleboxes and network policies.

Let m be the current ith middlebox of a policy $p_k \in \mathcal{P}$, i.e., $m = p_k.list[i]$. Let $m \xrightarrow{p_k,i} \hat{m}$ denote migrating the ith middlebox of p_k from m to $\hat{m}$. The feasible space of candidate middleboxes for m is denoted by:

$$M(p_k, i) = \Big\{ \hat{m},\ \forall \hat{m} \in \mathcal{M} \setminus m \,\Big|\, \hat{m}.type \text{ is equal to } m.type, \sum_{p_j \in A(\hat{m})} f_j.rate \leq (\hat{m}.capacity - f_k.rate) \Big\} \quad (5)$$

We assume that the middleboxes assignment for any policy p_k is an atomic operation, i.e., either all required middleboxes are assigned for p_k or none is assigned. Supposing p_k is an assigned policy for a flow f_k, the migration of middleboxes defined in $p_k.list$ is detailed in the following.

1) Migration of Intermediate Middleboxes: Intermediate middleboxes are middleboxes that are neither in the first ($i = 1$) nor in the last position ($i = p_k.len$) of $p_k.list[i]$, since those end middleboxes are either connected directly with the end points of the communication, e.g., the UE and its VM. Due to the advantage of having a SDN-based core network, let $U(m_1 \rightarrow m_2)$ denote the end-to-end latency between the middleboxes m_1 and m_2. Similar to the previous gain calculation for VM migration, the expected $gain$ in reducing the end-to-end latency that flows governed by p_k will face if the migration $m \xrightarrow{p_k,i} \hat{m}$ takes place is defined as follows:

$$\begin{aligned} G(m \xrightarrow{p_k,i} \hat{m}) = {} & U(p_k.list[i-1] \rightarrow m) + U(m \rightarrow p_k.list[i+1]) \\ & + U(m \xrightarrow{\sigma_{p_k}} \hat{m}) \\ & - U(p_k.list[i-1] \rightarrow \hat{m}) - U(\hat{m} \rightarrow p_k.list[i+1]) \end{aligned} \quad (6)$$

where $U(m \xrightarrow{\sigma_{p_k}} \hat{m})$ is the time that takes to migrate the internal states of the policy p_k from m to $\hat{m}$.

2) Migration of End Middleboxes: The migration of end middleboxes involves the migration of either $p_k.in$ (ingress) or $p_k.out$ (egress). The key difference is that UEs and VMs are included in the gain calculation, since end middleboxes communicate directly to the endpoints of the communication. Hence, in this case, the expected gain is defined as follows:

$$\begin{aligned} G(p_k.in \xrightarrow{p_k,1} \hat{m}) = {} & U(f_k.src \rightarrow p_k.in) + U(p_k.in \rightarrow p_k.list[2]) \\ & + U(p_k.in \xrightarrow{\sigma_{p_k}} \hat{m}) \\ & - U(f_k.src \rightarrow \hat{m}) - U(\hat{m} \rightarrow p_k.list[2]) \end{aligned} \quad (7)$$

The migration gain of the egress middlebox is similar to Equation 7. Thus, a middlebox $\hat{m} \in M(p_k, i)$ is considered to be an alternative for the current $m = p_k.list[i]$ if the gain of migrating m to $\hat{m}$ result in a positive balance, i.e., $G(m \xrightarrow{p_k,i} \hat{m}) > 0$. Also, the *gain* of migrating the middlebox m to m is considered to be null, i.e., $G(m \xrightarrow{(p_k,i)} m) = 0$.

C. Migration Decomposition

According to our previous paper [8], the migration of VMs and policies are *decomposable* since they are independent from each other. Without loss of generality, the overall gain calculation for a flow f_k guided by p_k is given as follows:

$$G(s \xrightarrow{v} \hat{s},\, p_k.in \xrightarrow{p_k,1} \hat{m}, \tilde{m} \xrightarrow{p_k,2} \acute{m},\, p_k.out \xrightarrow{p_k,3} \ddot{m}) = G(s \xrightarrow{v} \hat{s}) + G(p_k.in \xrightarrow{p_k,1} \hat{m}) + G(\tilde{m} \xrightarrow{p_k,2} \acute{m}) + G(p_k.out \xrightarrow{p_k,3} \ddot{m}) \quad (8)$$

According to Equation 8, the gain calculations can be treat independently during migrations. Due to space limitation, please, see [8] for more details regarding migration decomposition.

D. Policy and VM Migration Algorithms

The proposed PL-Edge scheme is comprised in two steps: first the migration of policies and then the migration of VMs.

*1) **Policy Migration**:* Let f_k be a flow governed by a policy p_k. We define a $(n+1)$-tier directed acyclic graph (DAG) as the *latency network* from $f_k.src$ to $f_k.dst$, in which $n = p_k.len$ is the number of middleboxes that f_k has to transverse. This DAG is composed of only one entry node (*source*) and one exit node (*sink*). Flows originate from the entry node and terminate at the exit node. If the entry node represents an UE, then the exit node represent its VM and the last node-tier is composed of nodes that represent the candidates servers that are able to host the VM (Equation 1). Each of the n remaining node-tiers represents all possible middleboxes according to $p_k.list$. For example, the first node-tier includes nodes that represent all candidates middleboxes according to $p_k.list[1]$ (Equation 5).

Figure 1 illustrates a (n+1)-tier DAG as a latency network for f_k. The weight of each edge is initialised as the corresponding *gain* between two adjacent nodes, while the weights of all nodes are zero. The problem of maximising the *gain* between *source* and *sink* becomes in finding the shortest path from *source* to *sink* assuming edges with negative weights. We attempt, therefore, in minimising the end-to-end latency among *source* and *sink* by maximising the gain among them.

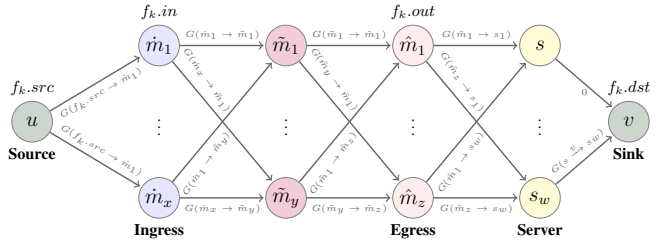


Fig. 1. Latency network example: Suppose a flow f_k from an UE u to its VM v has to across the middleboxes types $\dot{m}$, $\tilde{m}$, and $\tilde{m}$ according to a policy $p_k \in P(u, v)$, where v is currently hosted in s.

Algorithm 1 Step I: Policy Migration

```
1: procedure POLICY_MIGRATION
2:     for each Base station B_i = {U_i, C_i} ∈ B do
3:         for each UE u in U_i do
4:             v = the u's VM
5:             s = the current server that host v
6:             S = S(v) ∪ {s}   ▷ the candidate servers for v (Eq. 1) plus s
7:             ρ = the server preference array to accept v
8:             P = P(u, v) ∪ P(v, u)          ▷ all policies among u and v
9:             Call Migrate_Policies (P, S, ρ)
10:            Call Algorithm 2 to perform the step II
11: procedure MIGRATE_POLICIES (P, S, ρ)
12:    for each p_k in P do
13:        for i = 1 to p_k.len do
14:            M(p_k, i) = the candidate middleboxes for p_k.list[i] – Eq. 5
15:        G = construct the latency network for p_k as a (n+1)-tier DAG
16:        (ŝ, list) = Shortest_Path(G)   ▷ nodes in the shortest path
17:        for i = 1 to p_k.len do
18:            if p_k.list[i] ≠ list[i] then
19:                Perform policy migration: p_k.list[i] → list[i]
20:                Update the allocations A(p_k.list[i]) and A(list[i])
21:        Update routing for policies that have been migrated
22:        ρ[ŝ] = ρ[ŝ] + 1              ▷ Update the preference server array
```

Algorithm 2 Step II: VM Migration

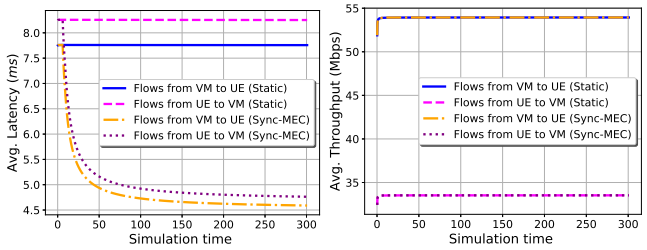
```
1: procedure VM_MIGRATION (ρ, S, v)
2:     ŝ = the server in S that has the highest ρ[s] value.
3:     if ŝ is not the current server that hosts v then
4:         Perform the migration of v from s to ŝ
5:         Update routing for v and the allocations A(v), A(s), and A(ŝ)
```

Algorithm 1 performs the policy migrations for each UE–VM communication pair by creating the a latency network for each flow of such pair. Since there may be several policy-flows among them, the migration of VMs are carried out in the next step, when all policy-flows among them have already been analysed. Algorithm 1 performs the policy migration by checking the UEs of each base station in a round-robin fashion.

*2) **Migration VM**:* According to Equation 4, migrating a VM v from its current server to a different one will yield different *gain*, which means that v can rank order candidate servers for migration. During the first step of policy migrations, the ρ array store the preference server to host v. The server that results in the highest ρ value will be selected to host v. If there is no tie, v is migrated to $\hat{s}$ if it is not already there. In case of a tie, the tiebreaker is performed by the server that results in the highest *gain* according to Equation (4). If the tie persists, the tiebreaker is performed randomly. Algorithm 2 shows the VM migration step of the proposed *PL-Edge*.

V. Evaluation Results

In this section, the effectiveness of the proposed *PL-Edge* is assessed and compared via simulation using *ns-3*. VMs and UEs are modelled as a collection of socket applications communicating with each others in the mobile infrastructure. We defined *policy-flows* as traffic flows that have to transverse a sequence of middleboxes as specified in their governing policies. In this evaluation, all traffic flows are randomly generated during the initialisation of the simulation, and all flows are policy-flows. Each policy-flow is configured to transverse 1 to 3 middleboxes and five types of middleboxes



(a) Latency (ms) (b) Throughput (Mbps)

Fig. 2. Average values of latency (left) and throughput (right)

were considered in the evaluation. For each middlebox type, there are 5 to 10 middleboxes deployed. The capacity of each middlebox was setup to 1Gbps As a result, the average path length of all flows was 3.4 hops. A centralised SDN controller is implemented to collect all latency network information to perform the proposed *PL-Edge* scheme. In order to compare the effectiveness of the proposed scheme in migrating policy and VM to reduce the average network latency of all flows, we have run the same simulation but using the *Static strategy*, which does not consider any migration of policies and VMs. The locations of VMs and policies using this strategy do *not* change during the simulation time once started.

Figure 2 shows the average network latency (in ms) and throughput (in Mbps) for a scenario that comprises 8 base stations. Each base station contains 10 to 400 active UE-VM communication pairs, where each UE-VM pair contains 1 to 5 active upload flows (UE to VM) and 1 to 5 active download flows (VM to UE). All flows are implemented as constant bit rate (CBR). In the simulation, the data rate (in Mbps) for flows from VM to UE and flow flows from UE to VM are taken from the uniformly distributed interval $[40; 50]$ and $[30; 40]$, respectively. The graphs of Figure 2 present 4 curves. Two curves corresponding to the results given by the proposed *PL-Edge* while the other two represent the results when static strategy is applied. We can easily observe from Figure 2a that the proposed *PL-Edge* scheme significantly reduces the average network latency while maintaining the average throughput of policy-flows (Figure 2b). For instance, once the proposed scheme is triggered, the average network latencies of both upload (UE to VM) and download (VM to UE) flows are considerably reduced by approximately 45% of the simulation time when compared to the results when no migration scheme is applied. As a preliminary study, the proposed *PL-Edge* scheme was able to provide a significant reduction on the end-to-end network latency among UEs and VMs, which is an important feature for mobile edge computing that considers a scenario where VMs are available for UEs offload applications' workloads.

VI. Conclusion

This paper proposes PL-Edge, a policy-VM latency-aware consolidation scheme for mobile edge. We have studied the network latency reduction in a cellular SDN-based infrastructure by jointly considering virtual machine and network policy dynamic (re)allocation. The proposed scheme is aimed at providing a low end-to-end latency between a user equipment (UE) and a virtual machine (VM) in the cloudlets. We first proved that this jointly optimisation problem is NP-Hard, and then we proposed the PL-Edge scheme to minimise the end-to-end latency between UEs and VMs by not performing only VM migrations but also policy migrations since network policies is a reality in any computer network. Preliminary results have shown that the proposed scheme for mobile edge computing was able to reduce the average latency by up to 45% while maintaining the throughput of policy-flows constant, besides strictly satisfying requirements of network policies. Future works include a scheme that also involves the network policy migration on the communication among VMs in cloudlets (within the carrier's premises) and VMs in the cloud (outside carrier's premises).

Acknowledgment

The work has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grants EP/P004407/1 and EP/P004024/1.

References

[1] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer (Long. Beach. Calif).*, vol. 50, no. 1, pp. 30–39, jan 2017.

[2] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–14, 2017.

[3] Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. Hauptmann, and M. Satyanarayanan, "Early Implementation Experience with Wearable Cognitive Assistance Applications," in *Work. Wearable Syst. Appl. - WearSys '15*, 2015, pp. 33–38.

[4] T. Taleb, "Toward carrier cloud: Potential, challenges, and solutions," *IEEE Wirel. Commun.*, vol. 21, no. 3, pp. 80–91, 2014.

[5] X. Sun and N. Ansari, "Green Cloudlet Network: A Distributed Green Mobile Cloud Network," *IEEE Netw.*, vol. 31, no. 1, pp. 64–70, 2017.

[6] V.-G. Nguyen, T.-X. Do, and Y. Kim, "SDN and Virtualization-Based LTE Mobile Network Architectures: A Comprehensive Survey," *Wireless Personal Communications*, vol. 86, no. 3, pp. 1401–1438, feb 2016.

[7] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You Can Teach Elephants to Dance: Agile VM Handoff for Edge Computing," in *2nd ACM/IEEE Symposium on Edge Computing*, October 2017.

[8] L. Cui, R. Cziva, F. P. Tso, and D. P. Pezaros, "Synergistic policy and virtual machine consolidation in cloud data centers," in *IEEE INFOCOM 2016 - 35th IEEE Int. Conf. Comput. Commun.* IEEE, 2016, pp. 1–9.

[9] J. Halpern and C. Pignataro, " Service Function Chaining (SFC) Architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015. [Online]. Available: https://tools.ietf.org/html/rfc7665

[10] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the impact of edge computing on mobile applications," in *ACM SIGOPS AsPac Wkshop on Systems*, 2016.

[11] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, February 2017.

[12] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[13] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-Defined Latency Monitoring in Data Center Networks," *Passiv. Act. Meas. (Pam 2015)*, vol. 8995, pp. 360–372, 2015.

[14] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware Steady State VM Management for Data Centers," in *Int'l. Conf. on Research Networking*, 2012.

[15] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling Innovation in Network Function Control," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 163–174, Aug. 2014.