Minimum Cost Load Migration in Distributed Software Defined Networks

Kuangyu Qin[1,2], Chuanhe Huang[1,2(✉)], N. Ganesan[3], and Kewei Liu[1,2]

[1] Computer School of Wuhan University, Wuhan, China
huangch@whu.edu.cn
[2] Collaborative Innovation Center of Geospatial Technology, Wuhan, China
[3] Regional Institute of Co-operative Management, Bangalore, India

Abstract. Software defined networking is a network paradigm which separates the control plane and the data plane. In a large SDN network, multiple controllers are used for handling switches' requests. Because of the unbalanced requests, sometimes some controllers get overloaded. If switch's management can be shifted from the overloaded controller to idle controllers, network performance can be improved. But shifting the switches will bring cost to the system. In this paper, the isolation nodes problem is studied. A minimum cost load migration approach and a heuristic migration algorithm are proposed. The proposed load migration approach can avoid the isolation nodes. The simulation results show that the proposed approach enhances the system performance.

Keywords: SDN · Load migration · Isolation nodes

1 Introduction

Compare to traditional network, the software defined networking (SDN) has the advantage of flexibility, scalability and virtualization [1]. In SDN, for every new flow, the switches request the controller according to the openflow [2] protocol. If the size of the network is large enough, single controller is not sufficient for dealing with the requests from all switches. Although topology shared controllers cluster can manage larger size network than single controller, but when the network size is keep increasing, distributed multiple controller SDN gets better performance. In distributed multiple controllers environment, sometimes a controller gets overloaded, while other controllers do not get fully used. If the system supports the mechanism to migrate the load from the overload controller to the underload controllers, the network performance can be improved.

Researchers have proposed approaches about how to balance the loads between controllers. Those solutions are mostly based on game theory. There is little research or no research considers the isolation nodes problem in SDN load migration. The isolation nodes will cause extra overhead when switches request the controllers. On the other hand, the load of the controller is changing frequently. A very balanced load situation in this moment is not so necessary.

I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 197–207, 2018.
https://doi.org/10.1007/978-3-030-00916-8_19

Consider switches migration will cause topology change and the communication will be affected until routing system gets convergence. The cost of the migration is non-negligible. Thus a migration approach which has lower cost is more useful.

In this paper, we studied the isolation nodes problem and proposed a connected load migration approach which can decrease the controller's load to the safe level with the minimal cost.

2 Related Work

For solving the problem that maps switches to their master controller, the authors of [3] designed the elastic distributed control system ElasticCon and proposed a switch migration protocol. A load balancing algorithm based on lowest utilization policy is proposed in [4]. The literature [5] proposed a lightweight collaborative mechanism for SDN controllers. An architecture which includes super controller and regular controller is proposed in [6]. The network is arranged as clusters. The super controller sends the state vectors to regular controllers and implements the load allocation. The authors of [7] proposed the DALB. The overloaded controller will collect the information of other controllers temporary, it has the low time efficiency. To solve this problem, the literature [8] proposed a load notification mechanism. In the approach, every controller notifies its load to other controllers in regular intervals. The authors of [9] proposed a dynamically switches adjusting approach. The approach adjusts the number of activated controllers and sleeping controllers according to the real time flow situation. The authors used the greedy knapsack and simulated annealing to solve the problem. The literature [10] described a multiple controller hibernation model. The authors proposed a genetic algorithm to get the optimal solution. A pareto-based optimal controller placement approach is narrated in [11]. The authors considered the performance, fault tolerance and load balancing and also proposed a policy for switches migration when fault happened. The authors of [12] proposed the approach for maximum resource utilization on the constraints of processor, bandwidth and memory. By using the non-cooperative game theory, they proposed the switches migration algorithm to maximum the benefits of players. The authors of [13] modeled the dynamic controller allocation for the stable matching problem. With the game theory, they also proposed a layered two phase algorithm to get the Nash equilibrium solution.

3 System Framework and Problem Formulation

3.1 System Framework

The system architecture is shown as Fig. 1. The bottom part is the data accessing and forwarding layer. The middle part is the control layer. The controller receives the requests from the switches and sends the flow table item information back. To implement the load migration among controllers, the system has a service module which known as Load Migration Service (LMS). The LMS is on the top level of the SDN.

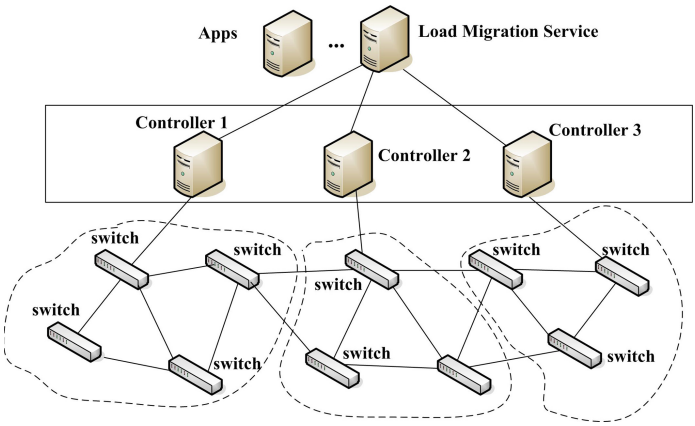


Fig. 1. The system view of the load migration for SDN

3.2 Problem Formulation

We define the network as a graph $G = (V, E)$. The V is the set of switches and the E is the set of links. Each link has latency. The network has k domains. Each domain has a controller to manage its switches. The set of controllers is denoted as $CV = \{c_1, c_2, ..., c_k\}$. Assume the controller $c1$ is the overloaded controller. The set of switches set in each domain is denoted as $S = \{s_1, s_2, ..., s_k\}$ where s_j is the switches set managed by the controller j. Let $S' = \{s_1', s_2', ..., s_k'\}$ be the switches set after migration, where s_j' is the switches set in new domain j. When switches send requests to controller, the controller can be modeled as an M/M/1 queuing model. Suppose packets arrive according to a Poisson process and the processing times of the packets are independent. Let λ_j denotes the total arrival rate of requests at the controller j, the f_i denotes the requests from switch i. Then λ_j can be calculated as follows

$$\lambda_j = \sum_{i \in S_j} f_i \tag{1}$$

Let controller's processing capacity as u. According to the queuing theory, the expected mean service time τ_j of controller j can be calculated as follows:

$$\tau_j = \frac{1}{u_j - \lambda_j} \tag{2}$$

Let p_{ij} denote the path from switch i to controller j. The latency of the path is the sum of latency for each link on this path. For each switch i, the response time includes the round trip latency for the path to controller and the processing time for the request. It can be denoted as

$$t_{ij} = 2d_{ij} + \tau_j, \; d_{ij} = \sum_{e \in p_{ij}} d_e \tag{3}$$

The load migration service monitors the controller. When the mean service time of a domain is higher than the warning level, the load migration service will calculate a plan to start loads migration. If the switch i migrates from controller A to controller B, the topologies of both domains are changed. The controller will update the topology by LLDP and it will take a few seconds. Before the routing system gets convergence, the switch i can not send new flows. Assume during the convergence the cost of communication interruption for each flow is α, the topology convergence cost for switch i is

$$C^i_{topo} = \alpha f_i \tag{4}$$

As the switch i is migrated from domain A to domain B, the information for switch i should be synchronized from controller A to controller B. The information includes the hosts, flows, groups, and meters related to this switch. Assume the size of these information is m_i and the cost for sending each byte is β. The cost of the synchronization is

$$C^i_{syn} = \beta m_i \tag{5}$$

The cost for migrating switch i is

$$C_i = C^i_{topo} + C^i_{syn} = \alpha f_i + \beta m_i \tag{6}$$

If a controller is overloaded, migrate many switches at once is easy to cause congestion. A better way to reduce the side effects is migrating switches one by one. As previous definition, the s_1 is the original switch set and the $s_1{}'$ is the switch set after migration. The switches which need to be migrated is

$$q = s_1 - s_1{}' \tag{7}$$

The total cost for migration is

$$C = \sum_{i \in q} C_i \tag{8}$$

The goal of the approach is to find a minimum cost migration plan that after migration the mean service time for each domain will be lower than the safe value t_s.

$$\begin{aligned} &\min \sum_{i \in q} C_i \\ &s.t.\ \tau_j < t_s, j = 1, .., k \end{aligned} \tag{9}$$

3.3 The Isolation Nodes Problem

The isolation nodes problem must be avoided during the load migration. The isolation nodes problem is the phenomenon that a domain is divided to two isolated parts because some key nodes in it are migrated to other domain. One example of the isolation nodes problem is shown as Fig. 2.

In the Fig. 2 there are two control domains managed by controller 1 and controller 2. In the Fig. 2(a) the load of control 2 is high. The migration plan decides to migrate the switch A from domain 2 to domain 1 in order to decrease the load of control 2. But this causes the isolation nodes problem. In the Fig. 2(b), The switch B can not connect to other domain 2's switches directly because the switch A is the key switch connects switch B. Assume that the host H1 needs to communicate with the host H2. Before migration, once the switch B requests the controller 2, the controller 2 will issue the flow rules for all switches along the path. But after migration, not only switch B needs to request the controller 2 for its flow table item, but also the switch A needs to request controller 1. It will cause burden to both controllers. Hence the performance decreases.

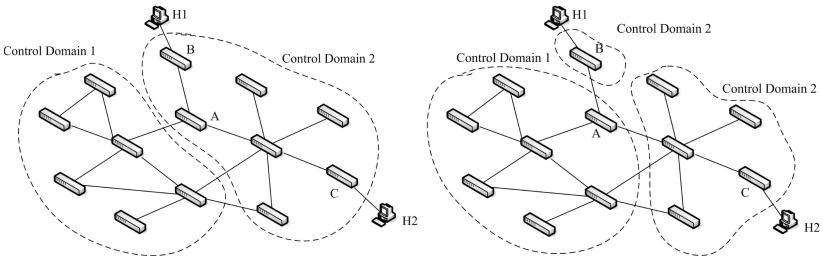


Fig. 2. The migration causes the isolation nodes.

If we decide to migrate switch A, the switch B should follow. The switch B is the migration dependent set of switch A. The migration algorithm needs to calculate the migration dependent set for each switch. If a switch is migrated, the switches in its migration dependent set need also be migrated.

4 Algorithm Designing

4.1 Algorithm for Calculating the Migration Dependent Set

For every node, we can identify a migration dependent set. If no other nodes depend on this node, the node's migration dependent set is empty. The main idea of the algorithm is dyeing to neighbors. The start node is the controller. The dyeing range is in the same domain. A queue is used in the algorithm. When there is no new nodes added to the queue, the algorithm ends. The detail of the algorithm is shown as Algorithm 1.

In the algorithm, each node enters the queue at most once. If the maximum degree of the graph is k, the time complexity of this algorithm is $O(kn)$.

Algorithm 1. Migration dependent set calculating algorithm

Input: graph G, the switch set S, node v to be migrated, controller $c1$
Output: migration dependent set H

$Q \leftarrow \phi$, $G' \leftarrow subGraph(G, S[1])$
all nodes colors in G' $\leftarrow$ black
v 's color $\leftarrow$ gray, $c1$'s color $\leftarrow$ white
$addQueue(Q, c1)$
while $Q \neq \phi$
    $u \leftarrow outQueue(Q)$, $N \leftarrow getNeighbors(G', u)$
    for each node i in N
        if $getColor(i)$ == black then $setColor(i$, white); $addQueue(Q, i)$
end while
$H \leftarrow$ remaining black nodes
return H

4.2 Algorithm for Minimum Cost Migration

The load of controller is changing continuously. To avoid oscillation, we define two levels, namely, safe level and warning level. The warning level is higher than the safe level. If a controller's mean service time is higher than the warning level, it will start the load migration. The load migration procedure will stop when the mean service time is lower than the safe level or there is no other controller to migrate.

Our algorithm uses a heuristic method to get the minimum cost solution. The migration includes many steps. In each step, there is only one switch gets migrated. Before we migrate each switch, the switch's migration dependent set is calculated. The switch and its migration dependent set will be considered as one big switch. Once a switch is migrated, a new switches-controller's mapping state is obtained. The algorithm constructs a states graph to trace the state transformation. For every new state there is a pointer indicates its old state. When a new state satisfies the goal, the reverse path from goal state to the initial state is a candidate plan. To get the minimum cost plan, we divide the cost of migration into two parts. One part is associated with the cost has been spent in current step. Another part is the cost to be spent for getting the goal. Let x denote a state. Define $f(x)$ as the total cost for the goal where x will be passed by. The $g(x)$ denotes the cost that has spent from initial state to current state x. The future cost need to spend is denoted by $h(x)$. Hence the cost $f(x)$ and the minimum cost $f^*(x)$ can be denoted as follow

$$f(x) = g(x) + h(x),\ f^*(x) = g^*(x) + h^*(x) \tag{10}$$

Because the future cost can not be measured, the heuristic function $h(x)$ is constructed to estimate the optimistic cost for the future steps. Assume that for a particular state x, the f_{min} is the new flows rate on switch which has the smallest new flows rate in remaining switches, f_{max} is the new flows rate for the switch which has the biggest new flows rate and m_{min} is the information size

for the switch which has the least information. If we migrates each switch, the load of controller will decrease f_{max}, we can calculate the value n that migrate at most n switches from current state the controller will achieve the safe level. We use these parameters to estimate the optimistic future cost. The optimistic $h(x)$ can be expressed as

$$h(x) = n(\alpha f_{min} + \beta m_{min}) \tag{11}$$

It is easy to see that the optimistic function $h(x)$ is no more than the real minimum future cost function $h^*(x)$.

$$h(x) \leq h^*(x) \tag{12}$$

In the algorithm we use the queue *OPEN* and *CLOSED* to record the visited states. Searching starts from the initial state *is*. The function *calcTime* calculates the mean service time for the state *x*. The function *getNeighborStates* finds the neighbor states of *x*. It moves one border switch from current state and calculates the switch's migration dependent set. If the migration dependent set is not empty, the switches and its dependent set will be seen as one big switch and moved together. The move of the switch is towards the neighbor domain which has least service time. If the move causes the distance from switch to controller greater than the threshold, this move must be abandoned.

Let the initial state is s and the set of goal state candidates is denoted by Γ. Let $r \in \Gamma$. Let P_{i-j} stand for the set of all paths from state i to state j. And let $P_{i-\Gamma}$ denote the set of paths going from state i to the set Γ. The $p_{i-r} \in P_{i-\Gamma}$ stands for any path from i to r. The set of cheapest paths from i to j is denoted

Algorithm 2. Minimum cost migration algorithm (MCMA)

Input: graph G, the set of switch set S, safe level ts, controller position $c1$
Output: migration plan P

$is \leftarrow S$, $OPEN \leftarrow \{is\}$, $CLOSED \leftarrow \{\}$
generate the state graph $SG \leftarrow \{is\}$
while true
    $x \leftarrow out\_queue(OPEN)$, $CLOSED \leftarrow CLOSED + \{x\}$
    if $calcTime(x) \leq t_s$ then out while
    $NS \leftarrow getNeighborStates(G, x, c1)$
    for each y in NS do
        calculate $f(y) \leftarrow g(y) + h(y)$
        If y is a new state, then
            add y into SG, let y point to x, append y into $OPEN$
        else if $newf(y) < oldf(y)$ then
            update y point to x, if y in $CLOSED$ then move y to $OPEN$
        sort $OPEN$ by their f values
end while
$RP \leftarrow the\ path\ from\ x\ to\ is$
$P \leftarrow reverse\ the\ RP$
return P

by P^*_{i-j}. Let the C^* stands for the cheapest cost of paths from s to Γ and let the Γ^* stand for the set of optimal goals.

Lemma 1. *At any time before the MCMA terminates, there exists an OPEN state x' on p^*_{s-r} with $f(x') \leq C^*$*

Proof. Consider any optimal path $p^*_{s-r} \in P^*_{s-\Gamma}$, $p^*_{s-r} = s, x1, x2, ..., x', ..., r$. The *MCMA* finds neighbor states from initial state. There must be one state in p^*_{s-r} is in *OPEN*. Let *x'* be a *OPEN* state on p^*_{s-r}. Since all parents of *x'* are in *CLOSED* and the path $s, x1, x2, ..., x', ..., r$ is optimal, the pointers assigned to *x'* are along $p^*_{s-x'}$, hence, $g(x') = g^*(x')$. Since $h(x) \leq h^*(x)$, we obtain:

$$f(x') = g^*(x') + h(x') \leq g^*(x') + h^*(x') = f^*(x') \tag{13}$$

And also since $x' \in p^*_{s-r}$, we have $f^*(x') = C^*$ and

$$f(x') \leq C^* \tag{14}$$

Theorem 1. *The algorithm MCMA returns the minimum cost migration plan.*

Proof. Suppose algorithm MCMA terminates with a goal node $t \in \Gamma$ for which $f(t) = g(t) \geq C^*$. The MCMA will order the *OPEN* and set the smallest cost state at the first. Hence when t was chosen for expansion, it satisfied:

$$f(t) \leq f(x) \qquad \forall x \in OPEN \tag{15}$$

This means that all states in *OPEN* satisfy $f(x) \geq C^*$. However, according to the Lemma 1 there was at least one OPEN state satisfies $f(x) \leq C^*$. Therefore the terminating t must have $g(t) = C^*$. Hence the algorithm MCMA returns the minimum cost migration plan.

5 Simulation

We perform numerical simulations to evaluate our approach. Our simulation program is developed in python. The python package NetworkX is used for modeling the networks. In the simulation we use two network topologies, one topology is AS-733 in 1st January, 1999, another topology is Oregon-1 in 31st March, 2001. Those topologies are collected from the Stanford Network Analysis Project (SNAP). The AS-733 has 531 nodes and the Oregon-1 has 10670 nodes. On these topologies, the high degree nodes are candidates to place controllers. We put 5 controllers on AS-733 and 30 controllers on Oregon-1. At the beginning, each controller manages almost equal number of switches.

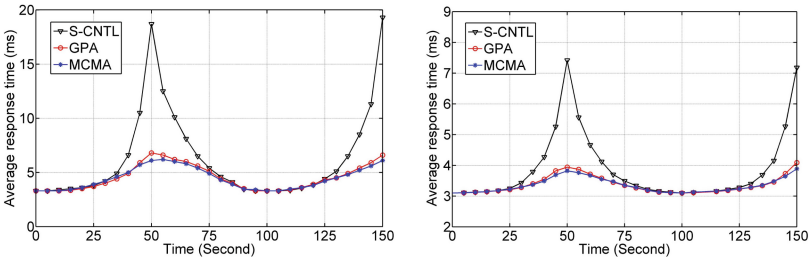
We set the average number of new flows rate for each switch as $f_i \in [30, 80]$. For each switch the simulation program generates new flows that follows the Poisson distribution. Each flow includes the switch address pair (s, t). The program checks whether the path crosses domains. If it does, the extra request will be added to the controller in every related domain. In the AS-733 topology, we

set the processing capacity of the controller as $u_j = 5000$ per second. In the Oregon-1 topology, the processing capacity of the controller is $u_j = 15000$ per second. The size of data for each switch in controller is $m_i \in [100, 5000]$ KB. On both topologies, the latency for every link is set as 1 ms.

To simulate the real environment, we random select some hot spots. If a switch is selected as a hot spot, its host's average new flow rate will become double. The hot spot will sustain for some time and then fade out. During the simulation, the percentage of the hot spots increases in the first 50 s, and then fade out in next 50 s, and then increases in next 50 s again. In the simulation, the warning level for the service time is set as 5 ms. The safe level is 0.5 ms. Once a switch is migrated, new flows on that switch will be suspended for one second, the response time of the new flow becomes 1000 ms and the average blocked data size for each flow is set as 500 KB.

We simulate the average response time for the switches while the request rate is changing. The static controllers (S-CNTL) scheme and the game playing algorithm (GPA) are used to compare with our approach. In S-CNTL scheme the controller-switch mapping is fixed. In GPA scheme, the overloaded controller select a most wanted switch to migrate and broadcast to other controllers. Those controllers discuss their difference of the average response time as the benefit. The controller which obtains the highest benefit will keep the switch. The response time is calculated according to the formula 3. The cost is calculated according to the formula 8. During the migration the allowed maximum path latency from switch to controller is 5 ms.

The Fig. 3 shows the average response time in AS-733 and Oregon-1. From the figures we can see when the load is increasing, the response time of S-CNTL scheme increases very fast as it does not support dynamic switch assignment. The GPA and MCMA support load migration, their performances are significant better than S-CNTL. As the MCMA chooses the plan which has less new flows be suspended and it supports isolation nodes avoidance. When the load is heavy, it has less response time than the GPA.

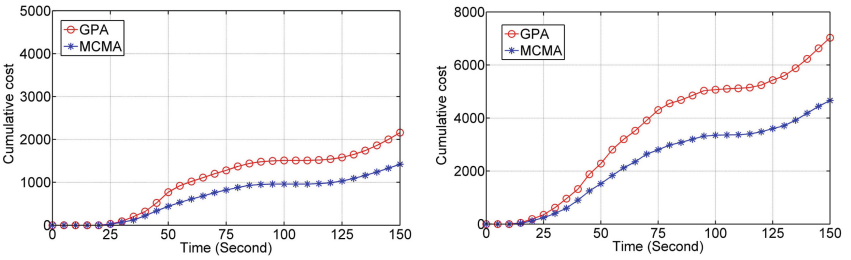


(a) AS-733 (b) Oregon-1

Fig. 3. Response time

The Fig. 4 show the cumulative costs of the load migration for GPA and MCMA. As the GPA migrates the switches according to maximizing the benefit

while the MCMA selects the minimum cost switches to migrate, the MCMA uses less cost than the GPA.



(a) AS-733 (b) Oregon-1

Fig. 4. Cost of migration

6 Conclusion

In large scale networks, multiple controllers are used for handling switches' requests. If the mapping between the controllers and switches is static, it is easy to cause controller overloaded. We studied the load migration of the controllers in SDN and introduced the isolation nodes problem. In this paper, a connected minimum cost heuristic algorithm is proposed for the load migration. The results of the simulation show that our approach can significant balance the loads between controllers with low cost. As the algorithm is still complicated, a more efficient load migration algorithm will be the future work.

Acknowledgments. This work is supported by the National Science Foundation of China (No. 61772385, No. 61373040, No. 61572370).

References

1. Farhady, H., Lee, H.Y., Nakao, A.: Software-defined networking: a survey. J. Comput. Netw. **81**, 79–95 (2015)
2. Mckeown, N., Anderson, T., et al.: OpenFlow: enabling innovation in campus networks. J. ACM SIGCOMM Comput. Commun. Rev. **38**, 69–74 (2008)
3. Dixit, A., Hao, F., Mukherjee, S., et al.: ElastiCon; an elastic distributed SDN controller. In: 2014 ACM/IEEE Symposium on ANCS, pp. 17–27. IEEE (2014)
4. Hu, Y., Wang, W., et al.: On the placement of controllers in software-defined networks. J. China Univ. Posts Telecommun. **19**, 92–97 (2012)
5. Chen, M., Ding, K., Hao, J., et al.: LCMSC: a lightweight collaborative mechanism for SDN controllers. J. Comput. Netw. **121**, 65–75 (2017)
6. Sufiev, H., Haddad, Y.: A dynamic load balancing architecture for SDN. In: Science of Electrical Engineering, pp. 1–3. IEEE (2016)
7. Zhou, Y., Zhu, M., Xiao, L., et al.: A load balancing strategy of SDN controller based on distributed decision. In: International Conference on Trust, Security and Privacy in Computing and Communications, pp. 851–856. IEEE (2014)

8. Yu, J., Wang, Y., Pei, K., et al.: A load balancing mechanism for multiple SDN controllers based on load informing strategy. In: Network Operations and Management Symposium, pp. 1–4. IEEE (2016)
9. Bari, M.F., Roy, A.R., Chowdhury, S.R., et al.: Dynamic controller provisioning in software defined networks. In: International Conference on Network and Service Management, pp. 18–25. IEEE (2013)
10. Fu, Y., Bi, J., Wu, J., Chen, Z., Wang, K., Luo, M.: A dormant multi-controller model for software defined networking. J. China Commun. **11**, 45–55 (2014)
11. Hock, D., Hartmann, M., Gebert, S., et al.: POCO-PLC: enabling dynamic pareto-optimal resilient controller placement in SDN networks. In: Computer Communications Workshops, pp. 115–116. IEEE (2014)
12. Cheng, G., Chen, H., Hu, H., et al.: Dynamic switch migration towards a scalable SDN control plane. Int. J. Commun. Syst. **29**, 1482–1499 (2016)
13. Wang, T., Liu, F., Guo, J., et al.: Dynamic SDN controller assignment in data center networks: stable matching with transfers. In: IEEE INFOCOM 2016, pp. 1–9. IEEE (2016)