Optimal Controller Placement in Software Defined Networks (SDN) using a Non-Zero-Sum Game

Hemant Kumar Rath, Vishvesh Revoori, SM Nadaf, and Anantha Simha

TCS Networks Lab, Bangalore, India
Email:{hemant.rath, vishvesh.revoori, sm.nadaf, anantha.simha}@tcs.com

***Abstract*—In this paper, we discuss optimal controller placement for Software Defined Networks (SDN) and propose a non-zero-sum game based distributed technique. Our proposed technique is a simple and low-complexity solution which runs in real-time. This can be implemented as an optimization engine at each SDN controller. The optimization engine at each controller computes a payoff function and compares its own payoff value with that of neighbors and takes appropriate decisions such that either new controllers should be added, or existing controllers should be deleted or offloading should be performed between controllers dynamically. We have conducted extensive simulations and verified the usability of the proposed scheme. We also propose a deployment framework which can be implemented using OpenFlow enabled platforms. Use of this technique not only can improve Quality of Services (QoS - minimum packet drops and delay) but also can save cost of deployment and operation.**

I. Introduction

Software Defined Networks (SDN) provide a vendor independent solution in which unlike traditional networks, control and data planes are de-coupled; control plane is managed by a set of controllers while data plane is managed by a set of forwarding devices or switches. The SDN controllers can run either on virtual machines (VM-based controller) or on physical machines (physical controller). A typical controller has an upper limit on the number of flows (traffic) it can handle at any point of time; hence there is a limit on the number of switches that can be managed by a controller. A single controller may be enough to manage the switches on a small scale network. However, as in data centers and enterprise networks etc., with increase in load on the network and size, the number of controllers required also increases. Moreover, the allocation of controllers to switches should be performed dynamically. Since the number of controllers used in the network directly impacts the capital and operational expenditures (CAPEX and OPEX), it becomes necessary to obtain optimal number of controllers and then map them to the switches (i.e., place the controllers optimally). The goal of the controller placement algorithm should be such that (i) load on the controllers is uniform, (ii) maximum utilization is achieved, and (iii) latency is reduced. This will not only reduce the expenditure, but also will make a positive impact on the environment.

Optimal controller placement can be made possible by dynamic addition or deletion of controllers as and when needed, i.e., add a controller by invoking a new VM and delete a controller by switching off a VM. However, selecting a particular VM to switch off and re-allocating the switches of that controller to other surviving controllers, is non-trivial. This becomes further complicated when the network is managed by physical controllers, as frequent switching on/off consumes more time and power. Therefore, instead of switching off, physical controllers may be put into idle/sleep mode; switching off can also be made possible, but must be decided using some optimization techniques. In this direction, we propose an optimal and dynamic controller placement algorithm in SDN. To make the controller placement problem distributed and scalable, controllers need to communicate among themselves such that load/delay information, routing updates, controller allocation information, etc., can be shared among themselves. This can be achieved by the inter-controller communication mechanism defined by SDNi [1], which is out of the scope of this paper. Our proposed solution is topology independent and is supported with appropriate theoretical analysis.

The rest of the paper is organized as follows. Section II brings the brief overview on the related work in the area of controller placement. In Section III, we formulate the problem of optimal controller placement and describe a novel solution technique in Section IV. In Section V, analytical and simulation studies of the proposed methodology are discussed. In Section VI, we provide a concluding remark.

II. Related Works

There have been several ongoing research projects [2]–[6] centered on placement of controllers in Software Defined Networks. Most of the solutions proposed in the literature are based on a centralized algorithm, in which a central controller can decide the number of controllers required and their placement. Moreover, they are topology dependent and as the network grows, they become non-scalable. Heller et al. [2] motivates the controller placement problem and advocates its relevance. It examines the impacts of the controller placement on average and the worst-case propagation latencies for real world topologies. However, [2] does not talk about the dynamic sharing of load between the controllers to address changing traffic. Dixi et al. [3] proposes “Elasticon“, an architecture that provides the ability to dynamically adapt controller resource pool. Though it shows encouraging results which are obtained by migrating switches from one controller to others, it does not discuss anything on the load adaptation algorithms, which our paper addresses. Zou et al. [4] proposes an Application Layer Traffic Optimization (ALTO) based mechanism for controller placement in SDN networks. It is a centralized scheme in which aggregate network information received from the SDN nodes (switches) are used to obtain forwarding and optimization decisions centrally, contrary to our requirement. Hu et el. [6] presents four controller placement algorithms to maximize the reliability of controller placement. However, these algorithms only work for initial placement of controllers, hence can only be used for static cases. Bari et al. [5] attempts to solve the dynamic controller provisioning problem through Integer Linear Programming (ILP) and proposes algorithms which minimizes flow setup times by dynamically changing the number of controllers and their locations. However, our work focuses on using game theory for dynamically mapping the switches to controllers for reducing the average controller-switch latencies and balancing load on controllers. In this process, we use the concept of global-individual optimization policy as defined in [7], [8].

III. Problem Formulation

Let there be M SDN capable switches connected to form a network representing one or multiple logical/physical domain. For simplicity, we assume uniformity among switches, though it is not a constraint for this formulation. Controllers are referred by the switches when new flows arrive at them so that forwarding rules or Forwarding Information Base (FIBs) can be updated. The controllers need to update the FIBs of the switches regularly and ensure QoS in the networks.

A. Optimal Number of Controllers and Placement

Let the M switches receive new flows randomly forming a non-uniform load scenario in the network at any time instant T. Assuming switch i receives l_i number of new flows and average load that can be handled by one controller is C, the minimum number of controllers required is:

$$\lceil k \rceil = \frac{\sum_{i=1}^{M} l_i}{C}. \tag{1}$$

The above argument is justified, if the load on the network/switches is known a-priori which is not possible in practice. Moreover, with the dynamic changes of load, value of k also changes dynamically. Our aim in this paper is to obtain the optimal value of k dynamically and optimally map k controllers (place) to M SDN switches.

We assume that every controller can run either in master[1] mode, slave mode or both (master for one set of switches and/or slave for another set of switches). In the slave mode, the controller is capable of listening to the switch referrals without any action. Both master and slave controllers can communicate with each other using an inter-SDN communication protocol. As the load on the network increases, one or more new controllers can be added to handle the load, resulting in change of placement of the existing controllers and change of state master to slave or vice versa. However, if the load on the network decreases, one active (master/slave) controller can be deleted resulting in change of placement of the surviving controllers and change of state from master to slave and vice versa. This process of addition/deletion of controller is captured through the following optimization problem:

$$\begin{gathered} \min f(k, c), \\ \text{s.t.}, \Delta_i \leq \Delta_{th}, \forall i \in I, \\ U_\alpha \leq U_i \leq U_{th}, \forall i \in I, \end{gathered} \tag{2}$$

where f is a non-linear function of the number of controllers k and cost c (both CAPEX and OPEX) associated with each controller. Note that, k and c are inter-related, and c could be a function of k. U_i is the utilization index (processor or memory or flow utilization), Δ_i is the delay (combination of processing and path delay) associated with i^{th} controller at time T, and I is the set of all active controllers operating in the network. Delay and utilization constraints are such that the delay associated with the controller should be less than a pre-defined threshold value (to support QoS) and the utilization should be within minimum and maximum thresholds (U_α and U_{th}); minimum threshold to reduce cost and maximum threshold to cater to sudden rise in network traffic respectively. Eqn. (2) is a global optimization problem in which the objective and constraints are conflicting with each other. Solution of Eqn. (2) should be such that the number of the controllers (k) which can be used is unique and optimum. Apart from this, the mapping of switches to controllers should ensure delay and utilization requirements. However, as load changes, obtaining an unique k is not possible, which can be used for all load conditions. Moreover, a centralized solution is not advisable due to scalability, controllability and fault-tolerance issues. We therefore attempt to solve Eqn. (2) using distributed individual optimization as follows (c_i is the cost associated with i^{th} controller):

$$\begin{gathered} \min c_i, \\ \text{s.t.}, \Delta_i \leq \Delta_{th}, \\ U_\alpha \leq U_i \leq U_{th} \end{gathered} \tag{3}$$

B. Solution Methods

The set of individual and global optimizations are equivalent, if the solution is unique and optimization solution exists for all possible load conditions. However, as mentioned before the load changes dynamically and hence obtaining a unique solution is not viable. Therefore, we argue that for an initial condition, we can solve the global optimization with a reasonable accuracy (level of accuracy can be user defined) or load variation and obtain k - optimal controller number for a particular traffic condition and then start the network with k controllers. As time progresses and load in the network changes, we solve Eqn. (3), such that new controllers can be added or an existing controller can be deleted to an initial solution obtained from Eqn. (2). Therefore, it is a joint global-individual optimization method of solving a multi-objective

[1]To ensure load balancing and fault tolerance in the network, we assume that each switch must have at least one master and one slave controller; all the neighboring controllers of a master are considered as the slaves for all the switches controlled by the master.

and multi-constraint optimization. In this paper, our focus is on the solution of the individual optimization problem, which we solve using a non-zero-sum game. We use a non-zero-sum game to obtain the best possible solution of Eqn. (3), as an optimal solution is not viable all the time. Hence, neither there is a single optimal strategy that is suitable to all others, nor a predictable outcome. Note that, non-zero-sum games are also non-strictly competitive, rather they are cooperative for a common cause.

We now discuss the non-zero-sum game which we propose to solve the above individual optimization problem. In this, each active controller is assumed to be a player and plays according to a set of rules defined in Algorithm 1. Each controller computes a payoff (f_x), which is a function of the existing utilization and delay as defined:

$$f_x = \lambda_x * [U_{th} - U_x] + \delta_x * [\Delta_{th} - \Delta_x], \quad (4)$$

where U_x is the current utilization and Δ_x is the delay associated with controller x. λ_x represents a non-linear function or a constant related to the usage of the controller and δ_x represents a non-linear function or a constant for delay payoff computation for x^{th} controller. Note that, each controller computes its payoff function f_x and takes its decision independently. Moreover, decisions are taken at different controllers at different time instants, i.e., an asynchronous mode of operation. As the load in the network changes, utilization and delay of the active controllers also change and hence the payoff.

Using Algorithm 1, i.e., by playing the game, each controller attempts to maximize its utilization and minimize the delay. In this process, based on the payoff value, one or more neighbor controller(s) can share the traffic of the controller 'X' (Step 7) and/or controller 'X' can trigger controller deletion message(s) (Step 9), such that they can be deleted (Step 4 to Step 12). This can be obtained using the Under Utilization Algorithm (Algorithm 2). Similarly, as load increases, any controller which is not satisfied with the current payoff it receives can either offload to a neighbor controller (Step 13 to Step 20) or trigger the controller addition message (Step 22). This can be obtained using the Over Utilization Algorithm (Algorithm 3). Post that, inter-SDN controller communication mechanism needs to be used to share the traffic profile, such as payoff matrix, neighbor list, switch list, utilization threshold and delay threshold, with the neighbors (Step 26). Note that, payoff is always greater than zero in both over utilization as well as under utilization cases. When payoff is very high ($f_x \geq \beta$), then the controller 'X' should be deleted resulting in overall optimization of the network. In the other case, i.e., when the payoff is very low then the controller 'X' will not be interested to take further load, rather will ask for possible offloading, resulting in load-balance in the network.

In the case of under utilization, using Algorithm 2, controller 'X' computes the possible change in payoff that can result at each of its neighbor j due to offloading of its switch i using Step 3, where ΔU_{ij} and Δ_{ij} are the change in utilization and delay at controller j due to offloading of switch i. After payoff change computation, controller 'X' checks for optimal offloading of each of the switches with its neighboring controllers, i.e., slaves (Step 5 to Step 23). Once the solution to offload all its switches is obtained, controller 'X' triggers for self deletion.

In the over utilization case (Algorithm 3), controller 'X' should find a group of switches connected to it which needs to be offloaded to a neighboring controller. This requires partitioning the set of switches connected to 'X' into multiple connected groups or solution groups (Step 3) using simple graph theoretic concept. In that process, controller 'X' has two options: offload the best solution group to its neighbors (Step 5 to Step 17) or trigger for a new controller (Step 21 to Step 33). First, the algorithm checks whether the offloading solution obtained using Step 5 to Step 17 is acceptable or not (Step 18). If not, controller 'X' triggers for a new controller. To check for best possible offloading solutions, it computes distributive payoff g_x which is a measure of possible impact on payoff on 'X' as well as on its neighbors. Controller 'X' then requests for offloading of the group which has less impact on its neighbors or equivalently with a best g_x (Step19, Step 33).

Algorithm 1 : Non-Zero Sum Game at Controller 'X'

```
Require: {Input data}
    Switch Set of '$X$' $S = \{S_i\}, i = 1 \to m$
    Slave/Neighbor set of '$X$' $N = \{N_j\}, j = 1 \to n$
    $\theta$ and $\beta$: min and max payoff thresholds respectively
    $U_{th}$, $\Delta_{th}$: max limits for utilization, delay of $X$
Ensure: Controller '$X$' is optimal in both utilization and delay
 1: while true do
 2:   if Change in traffic profile of '$X$' then
 3:     Compute payoff $f_x = \lambda_x * [U_{th} - U_x] + \delta_x * [\Delta_{th} - \Delta_x]$
 4:     if $f_x \geq \beta$ then
 5:       ($Sol_1$,$Solfound_1$) ← Algorithm 2($X$, $N$, $S$)
 6:       if $Solfound_1$ =1 then
 7:         $OffloadACK_1$ ← Ack from neighbors $N_t$, ($N_t \subset$
            $N$) in response to offload requests $Sol_1$.
 8:         if $OffloadACK_1$= 1 then
 9:           Offload traffic to $N_t$ and deallocate X.
10:         end if
11:       end if
12:     end if
13:     if $f_x \leq \theta$ then
14:       ($Sol_2$,$Solfound_2$, $type$) ← Algorithm 3(X, N, S).
15:       if $Solfound_2$ =1 and $type$ =1 then
16:         $OffloadACK_2$ ← Ack from neighbors $N_t$, ($N_t \subset$
            N) in response to offload requests $Sol_2$.
17:         if $OffloadACK_2$ = 1 then
18:           Offload traffic to $N_t$.
19:         end if
20:       end if
21:       if $Solfound_2$ =1 and $type$ =2 then
22:         Initiate a new controller and offload traffic using $Sol_2$
23:       end if
24:     end if
25:   end if
26:   Update neighbor information of $X$ and share $X$'s traffic profile
      with the neighbors.
27: end while
```

$$g_{xl} = (\sum_{i=1}^{n} (w_i/W) * f_{N_i}) + (w_x/W) * f_x, \qquad (5)$$

where l: group number n: no of neighbors affected due to offloading. N_i: i^{th} affected neighbor, W: total no of switches of 'X' and its affected neighbors based on offloading solution. w_i: no of switches mapped to i, f_{N_i}: payoff of neighbor i, post offloading.

Algorithm 2 : Under Utilization Algorithm at 'X'

```
Require: {Input data}
    (Controller under consideration $X$, Neighbors $N$, Switches: $S$)
Ensure: Every switch of $X$ is optimally mapped to its neighbors $N$
 1: n, m ← number of neighbors, switches.
 2: New controller mapping for switches $NCM \leftarrow [0\ 0 \cdots 0]_{1xm}$
 3: $f_{ij} \leftarrow \lambda*[U_{th_j}$ - $\Delta U_{ij}]$ + $\delta*[\Delta_{th_j}$ - $\Delta_{ij}]$ $\forall i = 1 : m, j = 1 : n$
 4: $changed \leftarrow 1$; $AtleastOneSwitchLeft \leftarrow 1$
 5: while $changed$ = 1 and $AtleastOneSwitchLeft$ = 1 do
 6:   $changed \leftarrow 0$; $AtleastOneSwitchLeft \leftarrow 0$
 7:   for $i = 1 \to m$, $j = 1 \to n$ do
 8:     if $NCM(i)$ = 0 then
 9:       $k \leftarrow j^{th}\ max_{index}\ \{f_{i1}, f_{i2}, f_{i3}, \cdots, f_{ij}\}$
10:       if $S_i$ is connected to $N_k$ then
11:         Map $S_i$ to $N_k$ and compute $f_{N_k}$.
12:         if $f_{N_k} \geq \theta$ then
13:           $NCM(i) \leftarrow j$; $changed \leftarrow 1$
14:         else
15:           Map $S_i$ back to $X$ and continue.
16:         end if
17:       end if
18:     end if
19:   end for
20:   if At-least one element of $NCM$ is 0 then
21:     $AtleastOneSwitchLeft \leftarrow 1$
22:   end if
23: end while
24: if $AtleastOneSwitchLeft = 0$ then
25:   $Solfound \leftarrow 1$; $Sol \leftarrow NCM$
26: else
27:   $Solfound \leftarrow 0$
28: end if
```

We argue that the updation at the active controllers happens only at discrete time intervals of t_{update}, a time period fixed for routing updation as used by OSPF [9] or BGP [10] or inter-SDN protocols [1]. Each controller independently updates the routing table (FIB) and decides whether to continue in the previous mode of operation or change; resulting in asynchronous updation at different controllers. For the algorithms to converge, each active controller once receives peer or neighbor information from other controllers, should be able to take a decision at least before the next peer information is received. We argue that, the non-zero-sum game we play here ensures the time convergence, which we verified through simulations. Amount of offloading is obtained such that utilization as well as delay constraints are met.

IV. Deployment Framework

As discussed before, this is a distributed decision technique, in which each controller runs a non-zero-sum game to solve

Algorithm 3 : Over Utilization Algorithm at 'X'

```
Require: {Input data}
    (Controller under consideration $X$, Neighbors $N$, Switches $S$)
Ensure: X's extra traffic is optimally offloaded to its neighbors
 1: ($V_x$,$E_x$) = (Switches, Links) of controller '$X$'.
 2: Potential Solution Groups $PSG \leftarrow \{(V_{i1},E_{i1}), (V_{i2},E_{i2})\}, i = 1 \to$ k. S.T ($V_{i1} \cup V_{i2} = V_x$ & $V_{i1} \cap V_{i2} = \emptyset$ & both ($V_{i1}$,$E_{i1}$), ($V_{i2}$,$E_{i2}$) are connected graphs)
 3: Values of Solution Group $VSG \leftarrow [-1, -1, \cdots, -1]_{1\times k}$
 4: Optimal Solution Group $OSG \leftarrow \{\ \}$
 5: for $i = 1 \to k$ do
 6:   ($Sol_1$, $SolFound_1$) ← Algorithm 2($X, N, V_{i2}$).
 7:   if $SolFound_1 = 1$ then Compute $g_{x_1}$ with $N_t \leftarrow Sol_1$, $X \leftarrow V_{i1}$
 8:   ($Sol_2$, $SolFound_2$) ← Algorithm 2($X, N, V_{i1}$).
 9:   if $SolFound_2 = 1$ then Compute $g_{x_2}$ with $N_t \leftarrow Sol_2$, $X \leftarrow V_{i2}$.
10:   if $Solfound_1 = 1$ or $Solfound_2 = 1$ then
11:     if $g_{x_1} > g_{x_2}$ then
12:       $VSG(i) \leftarrow g_{x_1}$; $OSG(i) \leftarrow \{Sol_1, V_{i1}\}$
13:     else
14:       $VSG(i) \leftarrow g_{x_2}$; $OSG(i) \leftarrow \{Sol_2, V_{i2}\}$
15:     end if
16:   end if
17: end for
18: if maximum($VSG$) $\geq \theta$ then
19:   $Solfound \leftarrow 1$; $type \leftarrow 1$; $Sol \leftarrow OSG(max_{index}(VSG))$
20: else
21:   Pool of Available Controllers $PAC \leftarrow \{C_l\}$, $i = 1 \to L$
22:   for $i = 1 \to k$ do
23:     $OC_1$ ← optimal Controller for $V_{i2}$ from $PAC$
24:     Compute $g_{x_3}$ with $OC_1$ as Neighbor, $X \leftarrow V_{i1}$
25:     $OC_2$ ← optimal Controller for $V_{i1}$ from $PAC$
26:     Compute $g_{x_4}$ with $OC_1$ as Neighbor, $X \leftarrow V_{i2}$
27:     if $g_{x_3} > g_{x_4}$ then
28:       $VSG(i) \leftarrow g_{x_3}$; $OSG(i) \leftarrow \{OC_1, V_{i1}\}$
29:     else
30:       $VSG(i) \leftarrow g_{x_4}$; $OSG(i) \leftarrow \{OC_2, V_{i2}\}$
31:     end if
32:   end for
33:   $Solfound \leftarrow 1$; $type \leftarrow 2$; $Sol \leftarrow OSG(max_{index}(VSG))$
34: end if
```

the above optimization problem (Algorithm 1- 3) and triggers a process to add/delete (switch off or idle) a controller. Appropriate measures can be taken to decide whether to switch off a physical controller or to put it into sleep mode. If virtual controllers are involved, then the process of addition and deletion becomes easy. Note that, the algorithms presented here are real-time in nature and run independently in each active controller.

The function of each active controller can be broadly divided into three categories: (i) inter-SDN controller communication, (ii) control function and (ii) controller optimization engine as shown in Fig. 1. The optimization engine performs the non-zero-sum game and decides who (which switch) should be offloaded. It also computes the self payoff value. The control function module computes the FIB and then updates the FIB with the inter-SDN control module, which is out of the scope of this paper. The inter-SDN control module communicates with the peers (peer controllers for routing updates, payoff value, offloading messages and state change messages etc.). It also provides an interface with the switches for master/slave mode of communication.

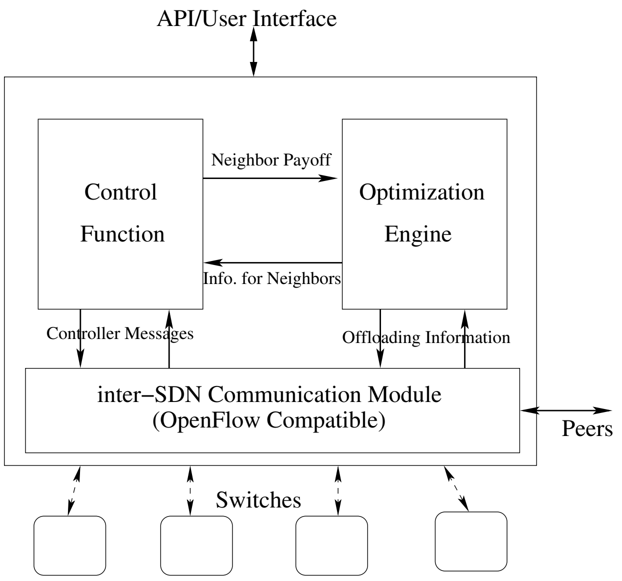

Fig. 1: Proposed Modules of the SDN Controller

V. Performance Evaluation

A. Bound on the Optimal Number of Controllers

Theorem 1: For a variable load condition, the optimal controller number is random with Gaussian distribution.

Proof We argue that at any instant, instead of fixed k controllers, we need $k_1 \leq k \leq k_2$ controllers, where k_1 and k_2 are minimum and maximum bounds. Though $k_1 = 1$ and $k_2 = M$ as the worst case condition, we need to estimate the bound $\lceil k_1, k_2 \rceil$, such that it is practical. The value of k_1 and k_2 can be estimated using statistical techniques as follows. As discussed before, we can deploy virtual machines running controllers and can invoke any number of controllers required at any instant of time. Since the load on each switch is random over time, load on total network is also random over time. Assuming large M and using Law of Large Numbers, we argue that total load on M switches at any time instant follows Gaussian distribution. Over a long duration, the load on the network is Gaussian in 3-D, with mean m and variance σ. Therefore, for a stable system $\lceil k \rceil = \frac{m}{C}$ and $\lceil k_1, k_2 \rceil$ should be within $\frac{\sigma}{C}$. This argument holds good for physical controllers also.

B. Simulations

We have conducted Matlab simulations by implementing the above discussed algorithms at each controller. We consider a random network of 28 switches/forwarding devices and assume traffic arrival at those switches to follow Poisson distribution. We start the scenario by placing one controller initially, and later based on the real-time changes in traffic we add and remove controllers as required. The simulation parameters used are as mentioned in Table I.

TABLE I: Operating Parameters

Operating Parameter	Value
No. of SDN Switches	28
Maximum Payoff Value β	0.9
Minimum Payoff Value θ	0.1
Utilization Threshold U_{th}	0.9
Latency threshold Δ_{th}	2 sec
Weightage λ_i	0.7
Weightage δ_i	0.3

To evaluate the proposed scheme, we consider the following three cases:

1) Case 1: In this case, we observe a scenario when there is a decrease in traffic (flow requests). In such a scenario, the number of active controllers can possibly come down. This specific case is captured through simulations and is shown using Fig. 2 and Fig. 3. In Fig. 2, initially there are 7 controllers serving 363 flow requests. With decrease in flow requests from 363 to 295 (Fig. 3), controllers C1 and C3 could be turned off without compromising on QoS.

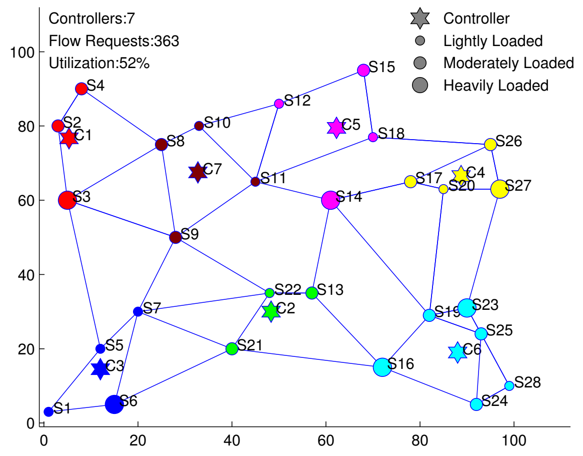

Fig. 2: Case 1: Original Setting

2) Case 2: In this case, we observe a scenario when there is an increase in incoming traffic at a controller (controller C7, Fig. 4), while there is not much change in traffic at other controllers. In such a scenario, the overloaded controller (C7) attempts to offload some of its traffic to its neighboring controllers using the over utilization algorithm. Controller C7 which is overloaded now offloads its incoming traffic of switches S10 and S11 to controller C5 (Fig. 4).

3) Case 3: In an other case, we observe a scenario when there is an increase in incoming traffic at several controllers. In such a scenario, where the total traffic cannot be handled even by the neighboring controllers, a new optimal controller should be initiated. In Fig. 5, controller C7 is overloaded and attempts to offload some of its traffic to neighbor controllers. Since, all its neighbor controllers are also overloaded, C7's extra

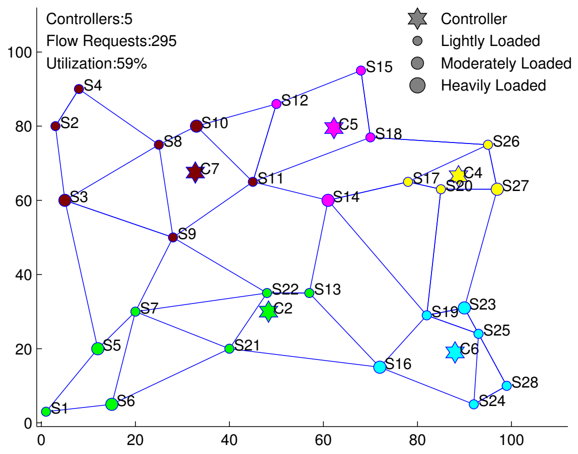

Fig. 3: Case 1: After Deletion of Controllers

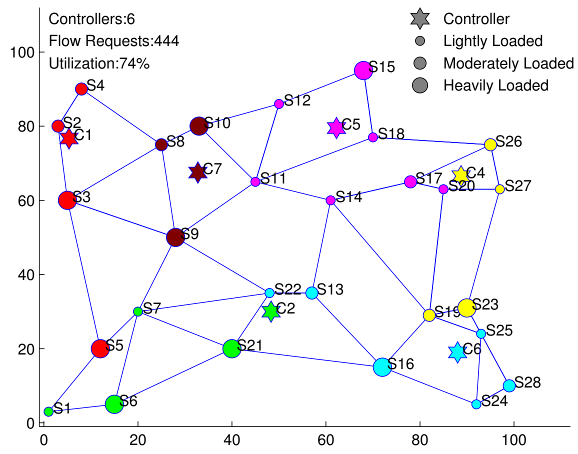

Fig. 5: Case 2: After Addition of Controllers

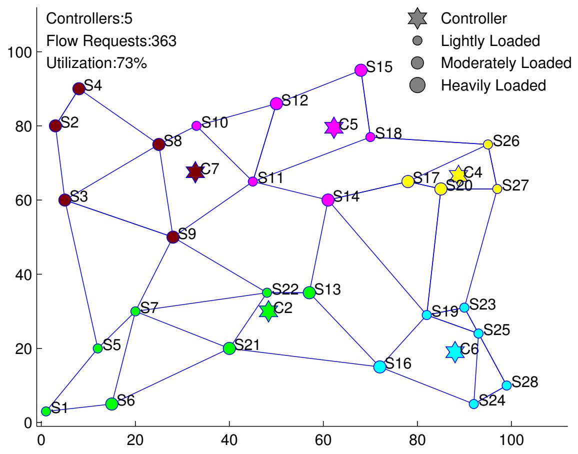

Fig. 4: Case 3: After Offloading

traffic cannot be accommodated and hence a new controller C3 is initiated as shown in Fig.5. Using these simulations we verify that packet drops which could have resulted due to overloading of controllers are avoided through the addition of new controllers and/or offloading and overall energy efficiency is achieved.

VI. Conclusions

In this paper, we propose a non-zero-sum based game theoretic scheme which can be used in a distributed manner at each active SDN controller. This scheme can ensure optimal number of controllers and their mapping to SDN switches, resulting in cost savings and QoS improvement. To deploy the non-zero-sum game, we also propose a set of algorithms which are simple, operate in real-time and do not require complex processing. All the algorithms proposed here have been simulated using Matlab and usability of these algorithms has been demonstrated through simulation results. Apart from this, we also propose a modular design of an SDN controller such that the above algorithms can be implemented in practice. To extend this proposal to real SDN networks, we plan to implement this in an emulation platform comprising of OpenFlow [11] enabled controllers and switches. We also plan to integrate inter-SDN controller communication technique such that appropriate evaluation can be executed. Post that, we plan to further extend this concept to a cloud network.

References

[1] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains," *Internet Draft, work in progress*, June 2012.

[2] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," in *Proc. of ACM HotSDN'12*, pp. 7–12, August 2012.

[3] A. Dixi, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," in *Proc. of ACM HotSDN'13*, pp. 7–12, August 2013.

[4] T. Zou, H. Xie, and H. Yin, "Supporting software defined networking with application layer traffic optimization," 09 2013.

[5] M. Bari, A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*, (New York, NY, USA), pp. 18–25, ACM, Oct 2013.

[6] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 672–675, May 2013.

[7] H. K. Rath, A. Sahoo, and A. Karandikar, "Cross Layer Congestion Control Algorithm in Wireless Networks for TCP Reno-2," in *Proc. of NCC, IIT-Delhi*, January 2006.

[8] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 2nd ed., 1999.

[9] J. Moy, "OSPF Version 2," 1998.

[10] C. K, "Border Gateway Protocol (BGP) and Traceroute Data Workshop Report," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, pp. 28–31, Jul 2012.

[11] "OpenFlow Enabled Mobile and Wireless Networks," September 2013. [Online] Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-wireless-mobile.pdf.