Metaheuristic Solutions for Solving Controller Placement Problem in SDN-based WAN Architecture

Conference Paper · January 2017
DOI: 10.5220/0006483200150023

7 authors, including:

Kshira Sagar Sahoo
National Institute of Technology Rourkela

Anamay Sarkar
National Institute of Technology Rourkela

Sambit Kumar Mishra
National Institute of Technology Rourkela

Bibhudatta Sahoo
National Institute of Technology Rourkela

Metaheuristic Solutions for Solving Controller Placement Problem in SDN-based WAN Architecture

Kshira Sagar Sahoo[1], Anamay Sarkar[1], Sambit Kumar Mishra[1], Bibhudatta Sahoo[1], Deepak Puthal[2,†], Mohammad S. Obaidat[3,4,†,‡] and Balqies Sadun[5]

[1]*National Institute of Technology, Rourkela, India*

[2]*University of Technology Sydney, Australia*

[3]*Fordham University, U.S.A.*

[4]*University of Jordan, Jordan*

[5]*Al-Balqa Applied University, Jordan*

{*kshirasagar12, anamay.2009, ganesh.sambit, bibhudatta.sahoo*}*@gmail.com*,
Deepak.Puthal@uts.edu.au, m.s.obaidat@ieee.org

Keywords: SDN, Controller, CPP, FireFly, PSO.

Abstract: Software Defined Networks (SDN) is a popular paradigm in the modern networking systems that decouples the control logic from the underlying hardware devices. The control logic has implemented as a software component and residing in a server called controller. To increase the performance, deploying multiple controllers in a large- scale network is one of the key challenges of SDN. To solve this, authors have considered controller placement problem (CPP) as a multi-objective combinatorial optimization problem and used different heuristics. Such heuristics can be executed within a specific time-frame for small and medium sized topology, but out of scope for large scale instances like Wide Area Network (WAN). In order to obtain better results, we propose Particle Swarm Optimization (PSO) and Firefly two population-based meta-heuristic algorithms for optimal placement of the controllers, which take a particular set of objective functions and return the best possible position out of them. The problem has been defined, taking into consideration both controllers to switch and inter-controller latency as the objective functions. The performance of the algorithms evaluated on a set of publicly available network topologies in terms execution time. The results show that the FireFly algorithm performs better than PSO and random approach under various conditions.

1 INTRODUCTION

In recent years, the use of Software Defined Networking (SDN) has become an emerging technology that provides many advantages to the cloud data centers and network service providers. Both SDN and Network Function Virtualization (NFV) technology creates the new era of network innovation through the virtualization of network resources (Jarraya et al., 2014),(Jammal et al., 2014). The underlying principle of SDN is the abstraction of the control plane from the data plane. It introduces the logically centralized control plane also referred to as Network Operating System (NOS), that hides the network complexities to the application developer by introducing well known southbound APIs. The network intelligence moved onto one or more external servers from the hardware devices, called controllers, which has a global view of the entire network(Sahoo et al., 2016).

Although placing a single controller is economically advantageous, it suffers single point of failure. Hence, deploying multiple controllers in the network is an alternate solution. However, random placing of controllers affects the performance of the system and degrades end to end QoS. So, it requires a proper planning to find the optimal location of the controller to get satisfactory performance and a reliable SDN. The CPP is an optimization problem is similar to facility location problem; which has considered as a NP-hard problem. Authors in (Heller et al., 2012), shows the performance of the network by varying the position of the controller in the network. So, it is a big challenge to find the location as well as number of controllers to be deployed in the network. Dynamically changing

[†]Fellow of IEEE & Fellow of SCS
[‡]Corresponding author

Sahoo, K., Sarkar, A., Mishra, S., Sahoo, B., Puthal, D., Obaidat, M. and Sadun, B.
Metaheuristic Solutions for Solving Controller Placement Problem in SDN-based WAN Architecture.
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 1: DCNET*, pages 37-45
ISBN: 978-989-758-256-1

the network architecture especially for WAN is a challenging task for the network operator. To adapt to this dynamics; the optimized location needs to be calculated. The POCO (Hock et al., 2014) framework has capable of handling small and medium size topology which provide the solution within seconds.However, for the large-scale network, it requires a lot of time for exhaustive evaluation for finding the location which may not cope with the dynamics of the network. The summary of our contributions are as follows:

- The main objective of this work is to minimize controller to node and inter-controller latency in the average and worst case scenario.
- To best of our knowledge, we are the first to propose the population-based meta-heuristic techniques to solve CPP on a set of real topologies considering the above as the objective function.

The rest of this paper has organized as follows. The Section II introduced the related work; the mathematical formulation described in Section III. The considered meta-heuristics algorithms have discussed in Section IV. Then Section V exhibits the performance analysis during the experiment followed by conclusion in Section VI.

2 RELATED WORK

In a given network which comprises of certain nodes(node may be switch, router, firewall etc.) , how to place the controllers is an open question. At first, Heller et al.(Heller et al., 2012) examined both average and worst case latency between the switch and controller in the Internet2 topology. The latency and traffic load between switch and controller have emphasized by Yao (Yao et al., 2014). In another work, Bari (Bari et al., 2013), proposed a dynamical provisioning of controllers aims to reduce flow set-up time and reassignment time of a switch to another controller in case of an overloaded situation. The POCO framework has been formulated with different metrics. The trade-off between the different metrics with all possible placements is examined in POCO. Although the inter-controller latency has considered it has not discussed in-depth in their work. In his work Lange et la.(Lange et al., 2015) extends the POCO framework and deploy K controller in the dynamic topologies. In their work, they have used Pareto Simulated Annealing (PSA) a meta-heuristic to solve the CPP. Hu et la. (Hu et al., 2013) worked on the reliability of the controller. They have used a metric to quantify the reliability called expected percentage of control path loss but ignores inter-controller latency. The paper(Sallahi and St-Hilaire, 2015), solve the CPP with various multi-objective functions. To the best of our knowledge, the authors have not used any population-based meta-heuristic technique in their work.

3 PROBLEM FORMULATION

In this work, we represent the network as a graph $G(V,E)$, where V and E represent as the node set and edge set. The node set consists of the switch, router, and controllers. In SDN, the switches and controllers are the forwarding elements. Here, we assumed that the controller locations are some of the forwarding nodes. Let $d(v,c)$ is the shortest path distance from a forwarding node $v \in V$ to one of the controller $c \in C$. For a particular placement the number of controllers is fixed to k, i.e. $|C_i| = k$. The all possible controller placement set can be represented as $C = \{C_1, C_2, ..., C_m\}$. For finding out a location for the controller $c_i \in C_j$; can be set as an optimization problem where the evaluation metrics are optimized. In our work, we have used latency as the evaluation metric. Latency refers to the time taken to reach a packet from source node to the destination node. But in case of SDN, the propagation latency between node and controller is proportional to the distance between a node to controller.

3.1 Controller to Switch Latency

It is the most common metric used in CPP. It is the longest distance between a node (v_i) and a controller (c_j) i.e. max $d(v_i, c_j)$. This is considered as the worst-case switch to controller latency. The objective of this worst case latency is to minimize the longest distance.

$$\pi^{worstlat}(C) = min_{v \in V} \quad max_{c \in C_i} \quad d(v,c) \qquad (1)$$

For the average latency case, the average distance between the placed controllers and remaining nodes assigned to them is calculated. In order tocompute it, the following equation has used.

$$\pi^{avglat}(C) = \frac{1}{|V|} \sum_{v \in V} min_{c \in C_i} \quad d(v,c) \qquad (2)$$

3.2 Inter-controller Latency

The latency between the individual controllers has a major significance because communications between these controllers are required to achieve proper synchronization of the network state. In order to minimize the controller to controller communication cost, they should deploy near to each other. To compute this metric, the d' must be calculated. The new $d'(c_i,c_j)$ is produced from $d(i,j)$ that signifies the shortest distance between controller c_i to controller c_j. The matrix d' removes the node that does not hold as a controller. The following equation is used to calculate the above metric.

$$\pi^{avgiclat}(C) = \frac{1}{|k|} \sum_{c_i,c_j \in C} min \quad d(c_i,c_j) \tag{3}$$

For worst-case inter-controller latency the maximum distance that separates controllers is computed. The considered metric can be formulated as:

$$\pi^{worsticlat}(C) = max_{c_i,c_j \in C} \quad d(c_i,c_j) \tag{4}$$

4 META-HEURISTIC TECHNIQUES FOR CPP PROBLEM

With a given the number of controllers k to be deployed in the network, the goal of our work is to find a placement C_i from the set C, such that the considered metrics must be minimized.

The single state meta-heuristic techniques like simulated annealing (SA) has a single candidate solution. This solution is used to compare with possible new solutions. Population-based algorithms like Firefly, PSO store many candidate solutions and the solution set compare them against each other. For exploring large and continuous space regions, single state algorithms are not suitable for large search space because of the possibility of stuck in local optima and lack of comparisons between candidate solutions.

4.1 PSO Algorithm for CPP

Particle Swarm Optimization (PSO) is a population-based stochastic technique. This algorithm searches the optimum solution from a population called the swarm. Based on two types of learning i.e. cognitive and social learning; it finds the best solution out of many. At the initial phase cognitive factor required for determining the best position. After searching the local best, social factor helps to find the global best position.

Let there are n particles, that represent the potential solutions and each particle represented as a d-dimension vector. Let, the current position of the particle is $\chi_i = [x_{i1}, x_{i2}, ..., x_{id}]$, where $i \in 1...n$. The current velocity is $V_i = v_{i1}, v_{i2}, ..., v_{id}$. The best position of the particle P_i is $p_{i1}, p_{i2}, ..., p_{id}$. The P_{gb} is the global best position vector.

At the time $t+1$, it updates the position (χ_{id}^{t+1}) and velocity (V_{id}^{t+1}) of the individual particle is defined as follows:

$$V_{id}^{t+1} = \omega V_{id}^{t} + c1'r1(P_{id} - \chi_{id}^{t}) + c2'r2(P_{gbd} - \chi_{id}^{t}) \tag{5}$$

$$\chi_{id}^{t+1} = \chi_{id}^{t} + V_{id} + t \tag{6}$$

Where ω represents initial weight,$r1$ and $r2$ denotes the random sequence: $r1, r2 \in \{0,1\}$. The positive constant $c1'$ and $c2'$ adjusts the cognitive part and social part respectively. The value of ω is used for improving the performance of the algorithm. The adaptive adjustment of ω helps to improve the local as well as global search ability, which can be represented as:

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{I_{max}} \times I \tag{7}$$

In the Equation 7, I_{max} represents the maximum number of iteration it can hold and I represents the current iteration. The ω_{max} and ω_{min} represents the maximum weight and minimum weight respectively.

4.2 Algorithm for CPP_PSO

The *CPP_PSO* Algorithm takes the co-ordinates of the nodes (*latlng*), required number of controllers(*nCont*), and the matrix containing the shortest path from each node to all other nodes (*srtPathMtrx*)as the inputs.The algorithm produces the semi-optimized value (*bstCst*) of the cost function and the final location of the Controllers (*fnlPosPSO*). The algorithm begins by setting the number of Nodes (*numNod*), x and y-co-ordinates of *latlng* as *latlngX* and *latlngY*, initializing the number of maximum iterations (*maxIt*), the population of the swarm (*nPop*), inertia coefficient (ω), damping ratio for inertia coefficient (ω*damp*), individual ($c1$) and social ($c2$) acceleration coefficients and sets Global Best Cost (*glBest.Cst*) to infinity. It initializes each particle's position (*par*(*i*).*Pos*) as a random permutation of *nCont* out of *numNode*. Each particle's velocity in x (*par*(*i*).*Vel.x*) and y (*par*(*i*).*Vel.y*) components set to zeros. Also it initializes each particle's cost (*par*(*i*).*Cst*) using the objective function. It also initializes best position of each particle (*par*(*i*).*Bst.Pos*) as the current position and best cost(*par*(*i*).*Bst.Cst*) as the current cost and sets Global Best Cost(*glBest.Cst*) as the optimum value among all the particles and Global Best Position (*fnl_pos_PSO*) as the position of the particle giving the optimal value. For each iteration and for the each particle the Algorithm 2 updates the particle's x and y component of velocity and the position. With the current position of the particle, x and y component of velocity and *latlng* are given as input and updates the

Algorithm 1 *CPP_PSO*

```
Input: latlng, nCont, srtPathMtrx
Output: optimized cost (bstCst), Final position of controllers (fnlPosPSO)
  Initialization :
  numNod ← Size of latlng
  latlngX, latlngY ← x- and y- co-ordinates of latlng
  maxIt ← maximum number of iterations
  nPop ← Swarm Size
  w ← inertia coefficient
  wdamp ← damping ratio
  c1 ← personal acceleration coefficient
  c2 ←social acceleration coefficient
  glBest.Cst ←Set Global Best Cost to ∞
  for i = 1 to nPop do
      par(i).Pos ←Randomly select nCont positions from numNodes
      par(i).Vel.x ←Initialize x-component of velocity with 0
      par(i).Vel.y ←Initialize y-component of velocity with 0
      par(i).Cst ← Pass par(i).Pos, latlngX, srtPathMtrx to CST_NC
      par(i).Bst.Pos ← par(i).Pos
      par(i).Bst.Cst ← par(i).Cst
      if (par(i).Bst.Cst < glBest.Cst) then
          glBest.Cst ← par(i).Bst.Cst
          fnl_pos_PSO ← par(i).Pos
      end if
  end for
  BstCost ←Initialize the Best Cost with 0
  Optimization Loop
  for it = 1 to maxIt do
      for i = 1 to nPop do
          par(i).Vel.x ←Update x component using Equation 5
          par(i).Vel.y ← Update y component of velocity similar to x
          component of velocity
          par(i).Pos ← Pass
          par(i).Pos, par(i).Vel.x, par(i).Vel.y, latlngX, latlngY to
          updPos_PSO
          par(i).Cst ← Pass par(i).Pos, latlngX, srtPathMtrx to CST_NC
          if (par(i).Cst < par(i).Bst.Cst) then
              par(i).Bst.Pos ← par(i).Pos
              par(i).Bst.Cst ← par(i).Cst
              if (par(i).Bst.Cst < glBest.Cst) then
                  glBest.Cst ← par(i).Bst.Cst
                  fnlPosPSO ← par(i).Pos
              end if
          end if
      end for
      BstCost(it) ← glBest.Cst
      w ← w × wdamp
  end for
  z ← BstCost from last iteration
  return z, fnlPosPSO
```

cost of the particle using the considered objective function. If the new cost is better than the previous cost of the particle, update the particle's current position and cost. If the updated cost is better than the previous one then, the global best cost and global best positions are updated to the particle's cost and position respectively.

The *updPos_CPP_PSO* Algorithm takes the current position of the particle (x), x-component of velocity (vx), y-component of velocity (vy) , x-coordinates of all nodes ($latlngX$) and y-coordinates of all nodes ($latlngY$) as input and produces the updated position (z). In order to obtains the x-component of particle's i^{th} controller position ($xPos(i)$), this algorithm get the sum of x-coordinate of i^{th} controller ($latlngX(x(i))$) and x-component of velocity of i^{th} particle ($vx(i)$). This process is similar for the y-component of particle's i^{th} controller position ($yPos(i)$). For each controller location, it initializes minimum distance between final position and any actual node co-ordinate (mini) to infinity and initializes the index ($mini\_index$) of such a node to 0. Then, for each controller location, it checks all other nodes and updates $mini\_index$ as the index of the node which gives minimum distance.

Algorithm 2 *updPos_CPP_PSO*

```
Input: x, vx, vy, latlngX, latlngY
Output: updated position (z)
  Initialization :
  sze ←size of x
  numNode ← size of latlngX
  xPos ←Initialize the x component of position with zero
  yPos ←Initialize the y component of position with zero
  dist←Initialize the final location with 0s
  Main Body
  for i = 1 to sze do
      xPos(i) ← latlngX(x(i)) + vx(i)
      yPos(i) ← latlngY(x(i)) + vy(i)
  end for
  for i = 1 to sze do
      mini ← ∞
      mini_index ← 0
      for j = 1 to numNode do
          if (Distance between i^th position and j^th node is less than mini)
          then
              mini← Distance between i^th position and j^th node
              mini_index ← j
          end if
      end for
      dist for i^th iteration ← mini_index
  end for
  z← dist
  return z
```

4.3 Firefly Algorithm (FFA)

This Algorithm 3 is based on the flashing characteristics of the fireflies. The biochemical process of illuminating light from firefly is called bio-luminescence has a significance role. The basic objective of this algorithm is to attract the opponent through rhythmic flashes. The inverse squared law inferred that the intensity of the light(I) decreases as the distance r increases. So, in a simple form the FFA based on the equation 8.

$$I \propto \frac{1}{r} \tag{8}$$

The light intensity $(I)$is having a fixed absorption coefficient γ, which has given in the equation 9.

$$I = I_0 e^{-r\gamma} \quad (9)$$

Where, I_0 is the initial light intensity. For simplicity and to avoid singularity at $r = 0$, we can combine both the equations.

$$I = I_0 e^{-r^2\gamma} \quad (10)$$

The attractiveness (β) is directly proportional to the intensity of the light, that are coming from nearby fireflies, which has given in the below equation.

$$\beta = \beta_0 e^{-r^2\gamma} \quad (11)$$

In the Equation 11 β_0 is the attractiveness at $r = 0$. For faster calculation we normalize the equation as a monotonically decreasing function.

$$\beta = \beta_0 e^{-r^m\gamma}; (m \geq 1) \quad (12)$$

The movement of one butterfly(i) to another butterfly (j) is determined by the given equation 13

$$x_i = x_i + \beta_0 e^{r_{ij}^2\gamma}(x_j - x_i) + \alpha\varepsilon_i \quad (13)$$

The third term is used for randomization where, α is the randomized parameter and ε_i is the vector of random numbers.

4.4 Algorithm for CPP_FFA

The Algorithm 3 takes the co-ordinates of the nodes, number of controllers and the matrix containing the shortest path from each node to all other nodes as input. Similar to *CPP_PSO* it produces the optimized value of the cost function and the final location of the controllers.

The Algorithm 4 takes all the fireflies (par), light intensity of the firefly (*Lightn*), randomness coefficient (α), absorption coefficient (γ) , x co-ordinates of nodes (latlngX) and y co-ordinates of the nodes (latlngY) as inputs. It returns the updated position of all the fireflies (par). The algorithm begins by initializing the number of nodes (*numNode*), number of particles (*numPar*) and the number of controllers (*numCon*) as the size of any particle. For each controller (*numCon*) the following steps are performed. It sets the x-coordinate (*xo*) and y-coordinate (*yo*) of all fireflies for all controllers to zeros. Then the x-coordinate ($xo(i)$) and y-coordinate of each particle ($yo(i)$) are updated. For each particle, set r as the distance between i^{th} particle and j^{th} particle. If the light intensity of i^{th} particle ($Lightn(i)$) is greater than the light intensity of j^{th} particle ($Lighto(j)$), then set β as the negative exponent of product of γ and r. The parameter $xn(i)$ set as in per the equation 13. Finally the algorithm sets dist as the updated position of the fireflies, in terms of the indices of the corresponding controller location and then it sets each firefly's corresponding controller location $(par(g).Pos(k))$.

Algorithm 3: *CPP_FFA*.

Input: *latlng, nCont, avlPathMtrx, srtPathMtrx*
Output: *bstCst, fnlPosFire*
Initialization :
numNode ← *size of latlng*
latlngX ← *x co − ordinates of latlng*
latlngY ← *y co − ordinates of latlng*
nPop ←*Initialize number of fireflies*
maxGen ←*Initialize number of iterations*
α ←*Set the randomness coefficient*
γ ←*Set the absorption coefficient*
δ ←*Set the randomness reduction coefficient*
minCst ←*Initialize the minimum cost of objective function to infinity*
Lightn ←*Initialize the light intensity of all the fireflies to zeros*
for $i = 1$ to *nPop* **do**
  $par(i).Pos$ ←*Randomly select nCont from numNode*
end for
Optimization Loop
for $i = 1$ to *maxGen* **do**
  zn ←*Initialize the cost for all the fireflies with zeros*
  for $j = 1$ to *nPop* **do**
    $zn(j)$ ←*Pass* $par(j).Pos, latlngX, srtPathMtrx$ *to Objective function*
  end for
  [*Lightn*] ← *Return sorted zn in Lightn*
  Lighto ← *Make a copy of Lightn*
  par ← *Pass* $par, Lightn, Lighto, \alpha, \gamma, latlngX, latlngY$ *to* $updPos\_Fire()$
  $\alpha \leftarrow \alpha * \delta$
  if ($minCst > Lighto(1)$) **then**
    $mincost \leftarrow Lighto(1)$
    fnlPosFire ←*Set fnlPosFire as the position of firefly that gave the minimum Cost*
  end if
end for
bstCst ← *minCst*;
return *bstCst, fnlPosFire*

4.5 Algorithm for the Cost Functions

In the main body of the algorithm, the average controller latency is added with average inter-controller latency. Average node latency and average controller latency are added to obtain the objective function (z), which is returned by the Algorithm 6.

5 EXPERIMENT

To evaluate of this work, we have taken various topologies from the TopologyZoo website. As per our knowledge since, no one has used any population-based meta-heuristic technique to solve CPP problem. In this work, we have used two metrics, which have been already discussed in the previous section.

Algorithm 4: *updPos_FireFly.*

Input: $par, Lightn, Lighto, \alpha, \gamma, latlngX, latlngY$
Output: par
Initialization :
$numNode \leftarrow size\ of\ latlng$
$numPar \leftarrow size\ of\ par$
$numCon \leftarrow$ *Set number of controllers as size of any particle*
Main Body
for $k = 1$ to $numCon$ **do**
  $xo \leftarrow$ *Initialize x-coordinates of the fireflies by zeros*
  $yo \leftarrow$ *Initialize y-coordinates of the fireflies by zeros*
  for $i = 1$ to $numPar$ **do**
    $xo(i) \leftarrow$ *Set the x-coordinate of* ith *particle's* kth *controller location from* $latlngX$
    $yo(i) \leftarrow$ *Set the y-coordinate of* ith *particle's* kth *controller location from* $latlngY$
  end for
  $xn \leftarrow xo$
  $yn \leftarrow yo$
  for $i = 1$ to $numPar$ **do**
    for $j = 1$ to $numPar$ **do**
      $r \leftarrow$ *Distance between* ith *and* jth *particle*
      if $(Lightn(i) > Lighto(j))$ **then**
        $\beta \leftarrow$ *Set as the negative exponent of product of* γ *and* r
        $xn(i) \leftarrow$ *Set as the sum of (product of previous* $xn(i)$ *and* $1-\beta$*) and (product of previous* $xo(j)$ *and* β*) and (product of* α *and a random number)*
        $yn(i) \leftarrow$ *Set as the sum of (product of previous* $yn(i)$ *and* $1-\beta$*) and (product of previous yo(j) and beta) and (product of* α *and a random number)*
      end if
    end for
  end for
  $dist \leftarrow$ *Initialize the final location of* k^{th} *controller of all fireflies with 0s*
  for $i = 1$ to $numPar$ **do**
    $mini \leftarrow$ *Initialize with* $infinity$
    $mini\_index \leftarrow$ *Initialize with* $0s$
    for $j = 1$ to $numNode$ **do**
      if (*Distance between* i^{th} *particle's* k^{th} *controller location and* j^{th} *node is less than mini*) **then**
        $mini \leftarrow$ *Distance between* i^{th} *particle's* k^{th} *controller location and* j^{th} *node*
        $mini\_index \leftarrow j$
      end if
    end for
    $dist\ for\ i^{th}\ iteration \leftarrow mini\_index$
  end for
  for $g = 1$ to $numPar$ **do**
    $par(g).Pos(k) \leftarrow$ *Set the* g^{th} *particle's* k^{th} *controller location as* g^{th} *entry of dist*
  end for
end for
return par

All the algorithms are written in Matlab version R2014a and runs on a system having Intel Core i5 with 4-Core processors and 8 GB RAM with Ubuntu 13.10 of 64-bit Operating System. The results are obtained by running the algorithms 30 times on the graph. Among 261 networks in the Topology-Zoo, we have considered 20 topologies for our work. The network sizes range from 50 to 150 nodes and among them the total number of controllers considered are between 5 to 20. The TataNld topology has been chosen as an example, which contains 144 nodes and 141 edges. When more than one metric are optimized, it is usually difficult to find an exact solution that satisfies all at the same time. So, there is a trade-off that is required between these two or more metrics. Thus to justify the necessity, the Pareto frontier is used.

Algorithm 5: *CST_NC.*

Input: $x, latlngX, srtPathMtrx$
Output: z
Initialization :
$sze \leftarrow$Set number of Controllers as the size of x
$numNode \leftarrow$Set number of Nodes as the size of $latlngX$
$contLat \leftarrow$Initialize average controller latency with 0
$nodeLat \leftarrow$ Initialize average node latency with 0
Main Body
for $i = 1$ to sze **do**
  for $j = 1$ to sze **do**
    $contLat \leftarrow contLat + srtPathMtrx(x(i), x(j))$
  end for
end for
$contLat \leftarrow contLat/(sze * sze)$
for $i = 1$ to $numNode$ **do**
  $mini \leftarrow \infty$
  for $j = 1$ to sze **do**
    $mini \leftarrow min(mini, srtPathMtrx(i, x(j)))$
  end for
  $nodeLat \leftarrow nodeLat + mini$
end for
$z \leftarrow nodeLat + contLat$
return z

Algorithm 6: *CST_NC.*

Input: $x, latlngX, srtPathMtrx$
Output: z
Initialization :
$sze \leftarrow$Set number of Controllers as the size of x
$numNode \leftarrow$Set number of Nodes as the size of $latlngX$
$contLat \leftarrow$Initialize average controller latency with 0
$nodeLat \leftarrow$ Initialize average node latency with 0
Main Body
for $i = 1$ to sze **do**
  for $j = 1$ to sze **do**
    $contLat \leftarrow contLat + srtPathMtrx(x(i), x(j))$
  end for
end for
$contLat \leftarrow contLat/(sze * sze)$
for $i = 1$ to $numNode$ **do**
  $mini \leftarrow \infty$
  for $j = 1$ to sze **do**
    $mini \leftarrow min(mini, srtPathMtrx(i, x(j)))$
  end for
  $nodeLat \leftarrow nodeLat + mini$
end for
$z \leftarrow nodeLat + contLat$
return z

5.1 How Worthwhile to Optimize the Controller Position?

The Fig.1 and 2 shows the ratio of random deployment of the controller to our proposed algorithms on the same topology. It has clearly observed that the average latency of the proposed algorithms is about 1.7 times greater than random deployment. For the larger value of $K(18)$, the cost of the random placement is almost 70% higher than that of other placements. For the worst case latency, this cost is even larger for both the algorithms. The ratio starts from 1.5 for a single controller, but for the higher value of K, the CPP_FFA increases to $2.1x$, whereas CPP_PSO increases to $1.9x$. Hence, it is important to optimize the placement of controllers in a given topology.

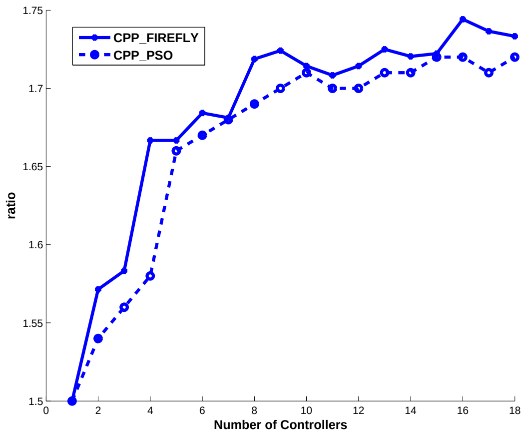

Figure 1: Ratio of random deployment to CPP_PSO and CPP_FFA (Average-case).

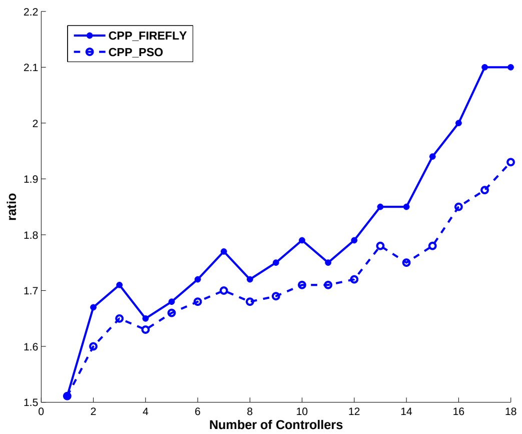

Figure 2: Ratio of random deployment to CPP_PSO and CPP_FFA (Worst-case).

5.2 Performance of the Algorithms

The Fig. 3 and Fig. 5 analyzes the impact of average and maximum latency on increasing the number of the controller, k ranges from 1 to 20, under the same network topology. The experimental result shows that by increasing the number of controllers for each placement, the both the latencies (cost) is decreased as expected. The average latency is calculated using both Equation 2 and Equation 3. From the plot it can be observed that, when $k > 15$ both average and worst case latency are almost linear,so we restrict the number of controllers to 15 for the topology. Similar to average case the worst case inter-controller latency is computed using Equations 1 and 4. The average delay of both meta-heuristic algorithms is usually relatively more stable than random algorithm. When K =16, the cost within an acceptable range for both the algorithms,

5.3 Computational Time of the Proposed Algorithms

With the increasing number of controller it can be seen from the Fig. 4 and Fig. 6 that CPP_FFA algorithm takes the shortest time. The processing time of both CPP_FFA and CPP_PSO is much faster than random algorithm. Due to the randomness and non-uniformity nature, the random algorithm behaves like it. But, the execution time taken by the two meta-heuristic techniques is almost close to each other in both the cases. For the larger network like TataNld, the CPP_FFA algorithm converges to the optimal solution slightly faster than CPP_PSO.

5.4 Optimal Positions of the Controllers

Now, we compare the geographic locations of the controllers with respect to the different objective functions. The Fig. 7 and Fig. 8 shows the final outcome of the simulation with 15 controllers on TataNld topology. The black circle represents the controller position and the white circle represents the other than controller. The Fig. 7 depicts the locations of the controller that minimizes the average latency using $CPP\_FireFly$ algorithm, and the Fig. 8 represents the same using CPP_PSO algorithm.

6 CONCLUSIONS

In this paper, we discuss the controller placement problem for SDN-based WAN architecture. The greedy and brute force methods are well suited for

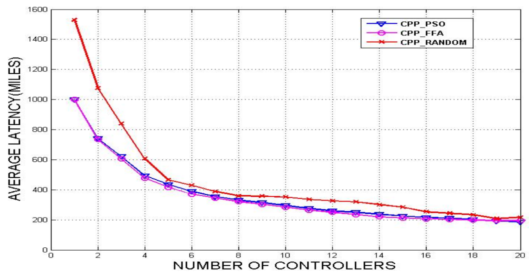

Figure 3: Impact of average case latency on deploying controllers.

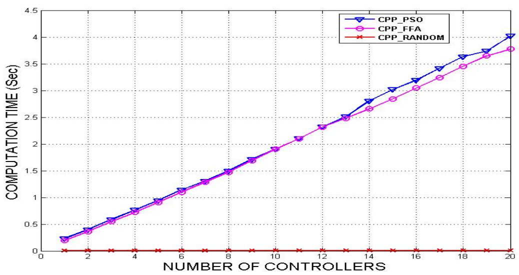

Figure 4: In average case the computational time of the algorithms.

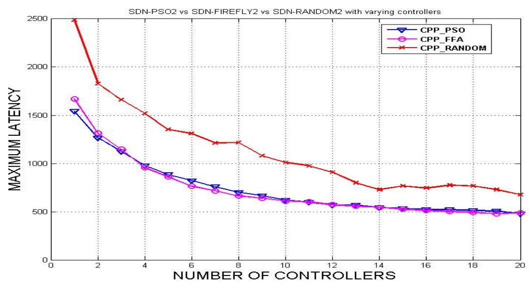

Figure 5: Impact of worst-case latency on deploying controllers.

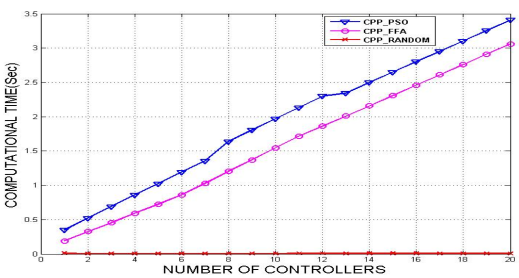

Figure 6: In worst case the computational time of the algorithms.

small and medium-sized problem instances, but for large scale problem instances a heuristic mechanism is a wise choice. Hence, we propose two meta-heuristic algorithms named as *CPP_PSO* and

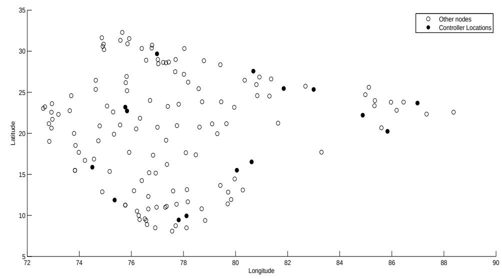

Figure 7: Optimal placement using CPP_FFA in the TataNld (Average-case latency).

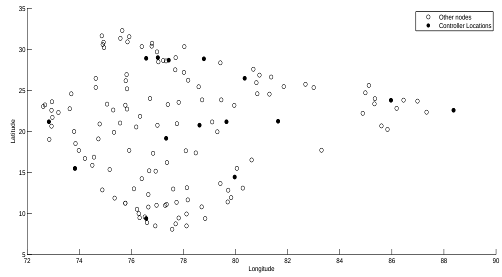

Figure 8: Optimal placement using CPP_PSO in the TataNld (Average-case latency).

CPP_FFA for evaluating CPP. In particular, such algorithms optimize a set of objectives that find the optimal number and location of the controller. The objective of this work is to minimize the latency between controllers as well as from switch to controllers. Experimental results show that for average case and worst case latency, the *CPP_FFA* has better performance than *CPP_PSO*. The time taken by both *CPP_PSO* and *CPP_FFA* are in almost close level. When network size is larger, *CPP_FFA* converges to the optimal solution slightly faster than *CPP_PSO* as it is the inherent property of these Meta-heuristic techniques. Finally, the scope of this paper is not limited to only this two objectives. As future work, we plan to incorporate other objective functions like load balancing and multi-path connectivity.

REFERENCES

Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., and Boutaba, R. (2013). Dynamic controller provisioning in software defined networks. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 18–25. IEEE.

Heller, B., Sherwood, R., and McKeown, N. (2012). The controller placement problem. In *Proceedings of the* *first workshop on Hot topics in software defined networks*, pages 7–12. ACM.

Hock, D., Gebert, S., Hartmann, M., Zinner, T., and Tran-Gia, P. (2014). Poco-framework for pareto-optimal resilient controller placement in sdn-based core networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–2. IEEE.

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

Jammal, M., Singh, T., Shami, A., Asal, R., and Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks*, 72:74–98.

Jarraya, Y., Madi, T., and Debbabi, M. (2014). A survey and a layered taxonomy of software-defined networking. *IEEE Communications Surveys & Tutorials*, 16(4):1955–1980.

Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., Hock, D., Jarschel, M., and Hoffmann, M. (2015). Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17.

Sahoo, K. S., Mohanty, S., Tiwary, M., Mishra, B. K., and Sahoo, B. (2016). A comprehensive tutorial on software defined network: The driving force for the future internet technology. In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, page 114. ACM.

Sallahi, A. and St-Hilaire, M. (2015). Optimal model for the controller placement problem in software defined networks. *IEEE Communications Letters*, 19(1):30–33.

Yao, G., Bi, J., Li, Y., and Guo, L. (2014). On the capacitated controller placement problem in software defined networks. *IEEE Communications Letters*, 18(8):1339–1342.