Application of Smell Detection Agent Based Algorithm for Optimal Path Identification by SDN Controllers

R. Ananthalakshmi Ammal[1(✉)], P.C. Sajimon[1], and S.S. Vinodchandra[2,3]

[1] Centre for Development of Advanced Computing, Thiruvananthapuram, India
{lakshmi, pcsaji}@cdac.in
[2] Computer Centre, University of Kerala, Thiruvananthapuram, India
[3] Department of Computational Biology, University of Kerala, Thiruvananthapuram, India
vinod@keralauniversity.ac.in

Abstract. Software Defined Networking separates the control plane and data plane with which the switches and routers become simply packet forwarding devices. The decision related to the path to be taken by the packet from the source to the destination is taken at the control plane. Thus the SDN controller has to identify the optimal path for the packets. Many of the SDN controllers use Dijkstra's algorithm for computing the shortest path and subsequently update the data plane devices. Many path computation algorithms including bio-inspired algorithms are published and are in use today in computer networks. In this paper, a novel bio inspired algorithm namely Smell Detection Agent based path computation algorithm is applied and studied for its performance in comparison with Dijkstra's algorithm, Extended Dijkstra's algorithm and the most commonly used bio inspired algorithm based on Ant Colony Optimisation. The Smell Detection Agent based algorithm inspired from the dog's smell detection capability for tracing and reaching a destination is found to be very useful and providing better results compared to the other algorithms.

Keywords: SDN · OpenFlow · Shortest path · Bio-inspired · Smell detection algorithm

1 Introduction

Software Defined Networking (SDN) is becoming popular in computer networking over the last few years and is presently a hot topic of research. In traditional networks, both the control plane and data plane reside in the routers and switches. The control plane in each device decides on the way the packets have to be forwarded. Whenever changes happen in network state or traffic patterns, there is lack of flexibility in accommodating those changes dynamically in the forwarding paths. With SDN, the data plane and control plane are separated [1]. There is a logically centralised SDN controller that configures the forwarding table in each data plane device, based on which the packets are forwarded in a network. The SDN controller is also responsible

Y. Tan et al. (Eds.): ICSI 2017, Part II, LNCS 10386, pp. 502–510, 2017.
DOI: 10.1007/978-3-319-61833-3_53

for network resource discovery and topology discovery. The controller gathers information about the state of the network including availability status, performance status including utilisation in a periodic manner. This enables the controller to identify the optimal path for the packets from source to destination and update the forwarding tables in the data plane devices in a dynamic manner. Thus SDN provides the required flexibility for efficiently managing traffic.

OpenFlow is one of the widely used protocols for communication between the controller and the data plane devices [2]. SDN controller uses OpenFlow Protocol for setting up flows for the switches, i.e. building up flow tables in switches. Whenever a new packet arrives at the switch port, resulting in a flow table miss, the switch can be programmed to send the packet to the controller which in turn receives the packet-in messages. The Controller intelligently sets the path to be traversed by the packet flows in the network. A controller has to be aware of the topology of the switch network and compute the packet traversal path. This gives rise to an optimisation challenge where the controller has to compute the optimal packet flow traversal path. Currently many of the SDN controllers use Dijkstra's algorithm for computing the shortest path and does not take into account the network link congestion, under-utilized state and network latency.

The problem is to compute optimal path from source to destination by the SDN Controller, given the network resource & topology discovery of the network are completed and the current status of the network resources such as device and link availability, bandwidth utilization are known to the Controller. To create path between source and destination within a network, some of the operational challenges include network link/resource failure condition, network link congestion condition, and network link under utilised condition. Dynamically finding the optimal path in shortest possible time based on network state changes, to reduce network latency is a major challenge.

In this paper, the application of bio inspired Smell Detection Agent based path computation algorithm [3] is applied and studied for optimal network path finding. The performance is compared with that of Dijkstra's [4] and Extended Dijkstra algorithms [5] and a variant of the most commonly used bio inspired algorithm based on Ant Colony Optimisation (ACO) [6].

2 Related Works

A few studies regarding the shortest or optimal path to be taken by the packet flows in Software Defined Networks have been done. The implementation issues in the modified Dijkstra's algorithm and the modified Floyd-Warshall shortest path algorithm in OpenFlow have been studied by Rus et al. [7]. Jehn-Ruey Jiang et al. have simulated the Extended Dijkstra's algorithm for SDN in a Mininet environment under the Abilene network topology [5]. Adnan Shahid et al. [8] have modified the SDN Controller to find out the highest bandwidth path instead of the shortest path.

Bio inspired algorithms for optimal path computations are in use for quite some time. The famous AntNet algorithm based on Ant Colony Optimisation [9] is used in many telecommunication Networks for routing purposes. Other bio inspired algorithms

used for path computation include Bee Colony Optimisation, Genetic Algorithms etc. Meta heuristics algorithms such as Particle Swarm Optimisation and its variants [10–12] are also used to solve hard optimisation problems. In SDN, study of application of bio-inspired algorithms is an emerging area. Ant Colony Optimisation approach to Quality of Experience (QoE) based flow routing [13] in SDN has been studied by Ognjen Dobrijevic et al. To the best of our knowledge, the application of Smell Detection Agent based model for optimal path finding by SDN Controllers is the first of its kind.

3 Smell Detection Agent (SDA) Based Algorithm

SDA based algorithm [3] is a novel bio inspired optimization algorithm based on the trained behavior of dogs in detecting smell trails. The olfactory mechanism of dogs can detect as well as memorize different smell signatures. Dogs also urinate in different spots to mark their territory as occupied. These two properties of dogs have been used to create the SDA to mark the path they undertake to reach the destination.

It is a known factor that different SDAs have varying olfactory capabilities and all the points within a territory cannot be traversed. The selected points within a territory are the smell spots which are visited by SDA. Each smell spot is characterized by two values, one is the signature related to the visit of an SDA to the smell spot and the other is the smell trail, which is value from destination. Each SDA is also characterized by two values, one is their signature value to be marked in smell spots and the other is the radius value indicating their olfactory capability. Thus the whole algorithm is based on a source, destination, smell spots and the smell radius for SDA to traverse. At any instance during the execution of the algorithm, the parameters related to the SDA include the signature value of SDA, total length traversed by the SDA and the current smell spot location of the SDA. Similarly for any smell spot, the parameters include the smell value and the signature of the visited SDA.

Application of SDA Based Algorithm in SDN Environment for Optimal Path Problem

The network domain under the control of the SDN Controller can be considered as the territory, a surface with smell trails and the agents inspired from dogs can be used to detect the optimal path. Each data plane device through which the packet flow has to traverse is considered as a smell spot. The smell value of the smell spot is assigned proportional to the number of interfaces available on the node. Thus the node having the highest number of interfaces gets maximum smell value and least one gets the minimal smell value. The SDAs start from the source node to the adjacent nodes and the visited nodes are marked with the signature of the SDA. Each SDA thus seek the path to the destination by traversing through the ‘not visited’ nodes. Since the visited nodes are marked, each SDA seeks disjoint path. Thus the execution the algorithm result in multiple possible paths to destination.

Algorithm: SDA based Algorithm for optimal path finding

```
Let
Nsda: Number of agents
Nss: Number of smell spots
1. Initialise Nsda=Number of interfaces of the source
   node
2. Initialise Nss=Number of data plane devices in the
   network
3. For i = 1 to Nsda
      Signature Value = i;
4. For i = 1 to Nss
      Smell value = number of node interfaces *(1 /
      highest interface count of a node in network)
5. For every agent 1 to Nsda,
      Update each agent to the next unvisited node with
      highest smell value
      Update the path including current node
      Update the total link weight value
6. Repeat step 5 until the SDAs reach the destination
```

The smell value of the nodes are decremented by the SDN Controller whenever there is a link failure, interface failure or high bandwidth utilization of the link indicating a possible congestion. The smell values are incremented when the interface become available or bandwidth utilization is below the threshold value. The optimal path has to be selected by comparing all the paths. To find the optimal path among the multiple identified paths, link weight is taken into consideration, apart from smell value of nodes. Each link in the network is assigned a link weight proportional to the interface bandwidth capacity, highest bandwidth is assigned the minimum link weight and the least bandwidth link is assigned the maximum link weight. The path with lowest link weight is chosen as the optimal path.

4 Test Bed Simulation and Results

Mininet

Mininet is an open source network emulation orchestration system for prototyping a large network on a single machine [14, 15]. It can create a network of virtual hosts, switches, links and support SDN Controller and OpenFlow protocol. Mininet runs on standard Linux system and uses virtualisation to emulate a complete network. The Mininet virtual hosts, switches and routers behave like real hardware, though they are created using software and we can send packets through Ethernet interfaces with specified link speed and delay. The advantage is that the same binary code and applications which we run on an actual network can be run in a Mininet network also.

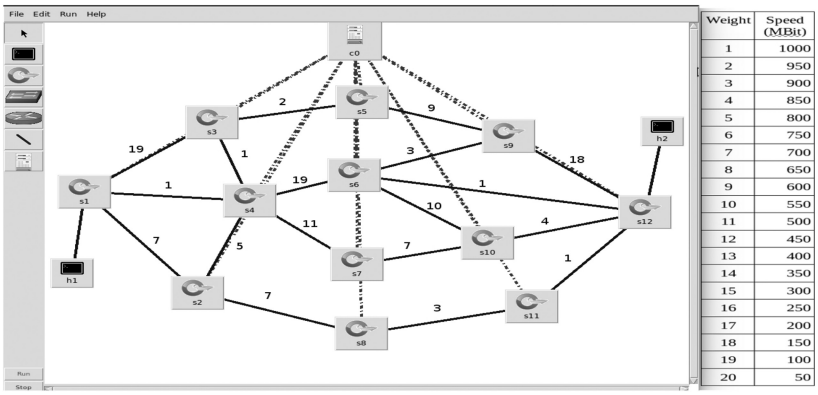
As part of our study, Mininet has been used to create a network test bed for testing the different algorithms with a basic set of parametrised topology, where a set of parameters are passed for a flexible topology.

Floodlight SDN Controller
Floodlight which is an Open Source Apache Licensed Java based SDN controller, is used for the simulations [16]. Being a module loading system, Floodlight SDN controller is easy to extend and enhance. We can consider Floodlight as a set of applications built over a Controller. Floodlight provides the core network services such as Device Manager, Link discovery, Topology Manager and application services such as forwarding, access control and firewall. By default, Floodlight SDN controller uses Dijkstra's algorithm to compute the path from source node to destination node. This algorithm has been replaced by different algorithms and the performance is compared to study the effectiveness of the bio inspired Smell Detection Agent based algorithm.

Test Bed Setup
Mininet is used to create a network topology for evaluation as shown in Fig. 1. The network consists of one SDN controller **C0** and 12 data plane devices from **s1** to **s12**. The source and destination nodes are two hosts **h1** and **h2** connected to data plane devices **s1** and **s12** respectively. Each data plane device is connected to the Controller. In the Floodlight SDN Controller the files related to topology such as *floodlightcontroller/topology/TopologyInstance.java, Topology Manager.java* were modified to incorporate the SDA based algorithm. The smell value of a node is assigned based on the number of interfaces in the node and the highest interface count of a node in network. The smell value ranges between 0 and 1. In the network topology shown in Fig. 1, the node **s4** has the highest number of interfaces on network, i.e. 5. Then **s4** has a smell value of 5/(1/5), equals to 1. Smell value of **s1** is 3/(1/5), equals to 0.6. The agents pass through the "not visited" nodes and the number of hops to reach the destination is an indicator of the distance to be traversed. In the present example shown in Fig. 1, the agents start from the node s1 and seek the unmarked next hops with the highest smell value and mark the smell spot as visited. This is iteratively done till the destination node of s12 is reached. Thus from s1, three paths are chosen to the destination:

Path 1: **s1** ↔ **s4** ↔ **s6** ↔ **s12** – Distance = 3 hops
Path 2: **s1** ↔ **s3** ↔ **s5** ↔ **s9** ↔ **s12** – Distance = 4 hops
Path 3: **s1** ↔ **s2** ↔ **s8** ↔ **s11** ↔ **s12** – Distance = 4 hops

To find the optimal path among the three identified paths, link weight is considered. The assigned link weights in the test bed topology for each link are marked in Fig. 1. Accordingly from the multiple paths that reach the destination, the one with the least weight is selected as the optimal path. The added link weight for the identified paths are 21, 48 and 18 for Path1, Path2 and Path3 respectively and the optimal path selected in this test bed is Path 3. This was verified by looking at the data plane devices flow table updates by the Controller with the help of modules for routing such as floodlightcontroller/routing/ForwardingBase.java. The smell value updates were done after getting the network monitoring statistics with the help of modules *floodlightcontroller/statistics/*



Weight	Speed (MBit)
1	1000
2	950
3	900
4	850
5	800
6	750
7	700
8	650
9	600
10	550
11	500
12	450
13	400
14	350
15	300
16	250
17	200
18	150
19	100
20	50

Fig. 1. Network topology for evaluation

StatisticsCollector.java. These modules provide the performance statistics related to bandwidth utilisation and port status.

5 Evaluation and Discussion

Benchmarks allow the evaluation of different algorithms for their performance. Three algorithms were executed on the same test bed after modifying the Floodlight SDN Controller. The algorithms include Dijkstra's algorithm, Extended Dijkstra's algorithm and a variant of ACO algorithm, which is a bio inspired one.

Dijkstra's Algorithm
The default algorithm for path computation in the Floodlight Controller is classical Dijkstra's Algorithm. In the original Dijkstra's algorithm, neither the nodes nor the links are associated with any weights. In other words, the algorithm does not take into consideration, either the bandwidth capacity of the links or the number of interfaces in the data plane node. Floodlight in SDN controller uses classical Dijkstra algorithm and it considers unit cost on all edges. Dijkstra's algorithm returns the shortest path from host **h1** to host **h2**, and is found to be **s1** ↔ **s4** ↔ **s6** ↔ **s12**. The distance is 3 hops.

Extended Dijkstra's Algorithm
In the Extended Dijkstra's algorithm, both link weight and node weight are taken into consideration. Here each link is assigned a weight proportional to the bandwidth capacity, as shown in Fig. 1. Each node is assigned a node weight based on the number of active interfaces among the total number of available interfaces. For the optimal path identification, the sum of link weights from source to destination is also considered. The link with the lowest weight is considered as the optimal path. Thus in the present example, the path chosen is **s1** ↔ **s4** ↔ **s3** ↔ **s5** ↔ **s9** ↔ **s6** ↔ **s12** with distance as 6 hops and the link weight as 17. The path chosen comprises the high bandwidth links though the number of hops are high.

Variant of ACO Algorithm

The ACO algorithm is modelled after ants which leave pheromone trails on their path in search of destination and their way back to source. The path with a stronger pheromone trail is the chosen path. The pheromone evaporates over a period of time, but in the chosen path the pheromone density will be strong as it is frequently traversed by more ants. In the present model, multiple ants of the same type are sent to the destination from the source in search of optimal path. The number of ants sent are equal to the number of connected interfaces in the source node. Each ant remembers the list of the nodes it has already visited on its way to destination. In this implementation of ACO algorithm, the links are assigned edge weights that is smell values, proportional to the bandwidth. The link with maximum bandwidth gets the maximum smell value and vice versa. The decay factor of smell is dependent on the distance from the destination and the number of levels of the topology tree as discovered by the SDN Controller. Thus as part of ACO algorithm implementation in SDN,

$$\text{Initial smell of link} = (\text{Weight of link/Maximum link weight on network}) * 100 \quad (1)$$

$$\text{Decay factor of smell} = \text{node distance} * (1/(\text{maximum level of tree})) \quad (2)$$

$$\text{Updated smell of link} = \text{link smell} + (\text{link smell} * \text{decay factor}) \quad (3)$$

In the present example, smell values of edges are assigned between *0 and 100* based on bandwidth of links. The smell is updated by considering decay factor on path. The highest smell path selected as the optimal path. Thus from the source host **h1** to destination host **h2**, the multiple paths, their smell values and decay values in the present example are given in Table 1. The path with highest smell value that is Path 2 is selected.

Table 1. Path computation using variant of ACO algorithm in SDN Controller

Path no:	Path nodes	Smell value	Decay factor	Total
Path 1	**s1** ↔ **s3** ↔ **s5** ↔ **s9** ↔ **s12**	150	72	78
Path 2	**s1** ↔ **s4** ↔ **s2** ↔ **s8** ↔ **s11** ↔ **s12**	415	251	164
Path 3	**s1** ↔ **s4** ↔ **s6** ↔ **s12**	195	78	117
Path 4	**s1** ↔ **s2** ↔ **s8** ↔ **s11** ↔ **s12**	310	166	144

We used *Iperf* network bandwidth measurement tool to test the TCP bandwidth performance. The testing time was set to 120 s. The four path finding algorithms were executed one after the other in a Floodlight Controller with the hosts **h1** and **h2** switching as client and server. The *Iperf* tool provided the bandwidth usage in all the four cases where the algorithms were executed by the Floodlight Controller. The results are shown in Fig. 2.

The maximum throughput of 681 Mbps was found to be for SDA based algorithm, followed by variant of ACO algorithm which was equal to 676 Mbps. The least

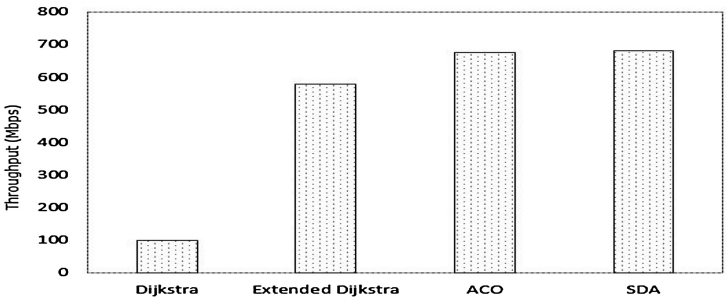


Fig. 2. Iperf bandwidth usage results

throughput of 99.4 Mbps was obtained for the unit weight classical Dijkstra's algorithm. The path selected by each of the algorithm for packet flows from **h1** to **h2** and back during the Iperf execution in the test bed is as given in Table 2. It is seen that the shortest path is selected by the unit weight Dijkstra's algorithm. The SDA based algorithm is the next shorter path with lesser number of hops compared to the other algorithms. When we compare both, the path taken and throughput, it is evident that SDA based algorithm performs better than all the other chosen algorithms.

Table 2. Details of selected paths

Sl No.	Algorithm	Path	Flows
1	Dijkstra	**h1 − > h2**	**s1, s4, s6, s12**
		h2 − > h1	**s12, s6, s4, s1**
2	Extended Dijkstra	**h1 − > h2**	**s1, s4, s3, s5, s9, s6, s12**
		h2 − > h1	**s12, s6, s9, s5, s3, s4, s1**
3	ACO variant	**h1 − > h2**	**s1, s4, s2, s8, s11, s12**
		h2 − > h1	**s12, s11, s8, s2, s4, s1**
4	SDA	**h1 − > h2**	**s1, s2, s8, s11, s12**
		h2 − > h1	**s12, s11, s8, s2, s1**

6 Conclusions

The simulation results clearly demonstrate the effectiveness of SDA based algorithm in finding the optimal path. Another point to be noted is that the bio inspired algorithms have shown better performance compared to the conventional algorithms. One of the limitations we faced in Mininet is that the experiments have to be conducted with slower links of 10–100 Mb/s. Here the packets were forwarded through OpenVSwitch rather than dedicated switching hardware. So the experiments have to be conducted with SDN Enabled switching platforms with high speed links of Gb/sec. Moreover realistic traffic load has also to be taken into consideration in the physical SDN test bed. In the present example only two attributes were considered in computing the optimal path which include the distance and the cost of edge in terms of link bandwidth. But in

practical network situations, other factors such as delay, reliability and packet loss have to be considered. Our research findings will be useful for researchers who work on SDN controller algorithms specifically for optimal path finding for packet flows.

References

1. Software-defined networking: the new norm for networks. In: Open Networking Foundation. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. Accessed 2012
2. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**, 69–74 (2008)
3. Vinod-Chandra, S.S.: Smell detection agent based optimization algorithm. J. Inst. Eng. India Ser. B **97**, 431–436 (2016)
4. Donald, J.: A note on Dijkstra's shortest path algorithm. J. ACM **20**(3), 385–388 (1973)
5. Jiang, J.R., Huang, H.-W., Liao, J.H., Chen, S.Y.: Extending Dijkstra's shortest path algorithm for software defined networking. In: Proceedings of the of IECiE APNOMS (2014)
6. Dorigo, M., Blum, C..: Ant colony optimization theory: a survey (2007)
7. Furculita, M.: Implementation issues for modified Dijkstra's and Floyd-Warshall algorithms in OpenFlow. In: Proceedings of RoEduNet International Conference on Networking in Education and Research (2013)
8. Shahid, A., Fiaidhi, J., Mohammed, S.: Implementing Innovative Routing Using Software Defined Networking (SDN). Int. J. Multimedia Ubiquit. Eng. **11**, 159–172 (2016)
9. Ducatelle, F., Di Caro, G.A., Gambardella, L.M.: Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. Swarm Intell. **4**, 173–198 (2010)
10. Qin, Q., Cheng, S., Zhang, Q., Li, L., Shi, Y.: Particle swarm optimization with interswarm interactive learning strategy. IEEE Trans. Cybern. **46**, 2238–2251 (2016)
11. Zhang, J., Lina, N.I., Chen, X.I.E., Ying, T.A.N., Zheng, T.A.N.G.: AMT-PSO: an adaptive magnification transformation based particle swarm optimizer. IEICE Trans. Inf. Syst. **94**, 786–797 (2011)
12. Solos, I.P., Tassopoulos, I.X., Beligiannis, G.N.: Optimizing shift scheduling for tank trucks using an effective stochastic variable neighbourhood approach. Int. J. Artif. Intell. **14**, 1–26 (2016)
13. Dobrijevic, O., Santl, M., Matijasevic, M.: Ant colony optimization for QoE-centric flow routing in software-defined networks. In: IFIP CNSM (2015)
14. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of ACM HotNets 2010 (2010)
15. Mininet. http://mininet.org
16. Floodlight controller. http://www.projectfloodlight.org/floodlight/