Department Of Computer Science & Information Systems

EC-2: Final Project Report

Routing as a Cognitive Service in Software-Defined Networks

For the partial fulfillment of

CS F441 - Selected Topics in Computer Science

Submitted By

M Sharat Chandra (2014A7PS108P)

On

28th April, 2017

Birla Institute of Technology & Science

Pilani

Acknowledgement

I would like to express my special thanks of gratitude to Prof. Santanu Chaudhary and Prof. Rahul Banerjee for introducing this course that helped me gain working knowledge and understanding of cognitive systems. I would also like to thanks Dr. Kamlesh Tiwari, Dr. Kuldeep Kumar and Dr. Arun Chauhan for their constant support and guidance throughout this project. Their mentorship motivated me throughout the project and guided me in proper direction. I gained a lot of knowledge and deep understanding regarding the subject through this project. I also gained insights about the research topics and activities in the field of cognitive systems.

Table Of Contents

1. Introduction **5**

2. Literature Survey **5**

3. Background **6**

2.1 Software Defined Networking 6

2.2 Cognitive Systems 6

2.3 Q-Learning 7

2.4 Deep Q Learning 8

4. Problem Statement and Objectives **9**

5. Modelling the problem using Deep Reinforcement Relevance Networks **10**

5.1 State Representation 10

5.2 Action Representation 11

5.3 Reward Function 12

5.4 Learning Algorithm of DRRN 13

5.4.1 Intuitive Explanation of Training Process 13

5.5 Connecting all components: The big picture 14

8. Tasks with Timeline **17**

9. References **18**

1. Introduction

Routing is the process of selecting a path for the flow of traffic in a network or between networks. In a traditional networking setup, the routing decisions are based on distributed intelligence (local decisions) and in general follow shortest path routing metrics (Eg. Spanning Tree Protocol, Multi-Protocol Label Switching etc) which are not adaptive to the state of the network. Software defined networking introduces programmability into networks and thus the ability to capture and exploit the network state and its properties. In order to route traffic in SDNs routing applications which run on top of the SDN controller are made use of. These routing applications make routing decisions based on network policies and the state of the network. This project examines the design of cognitive routing service using novel reinforcement learning algorithms and aims to improve end-user experience by maximizing the network utilization.

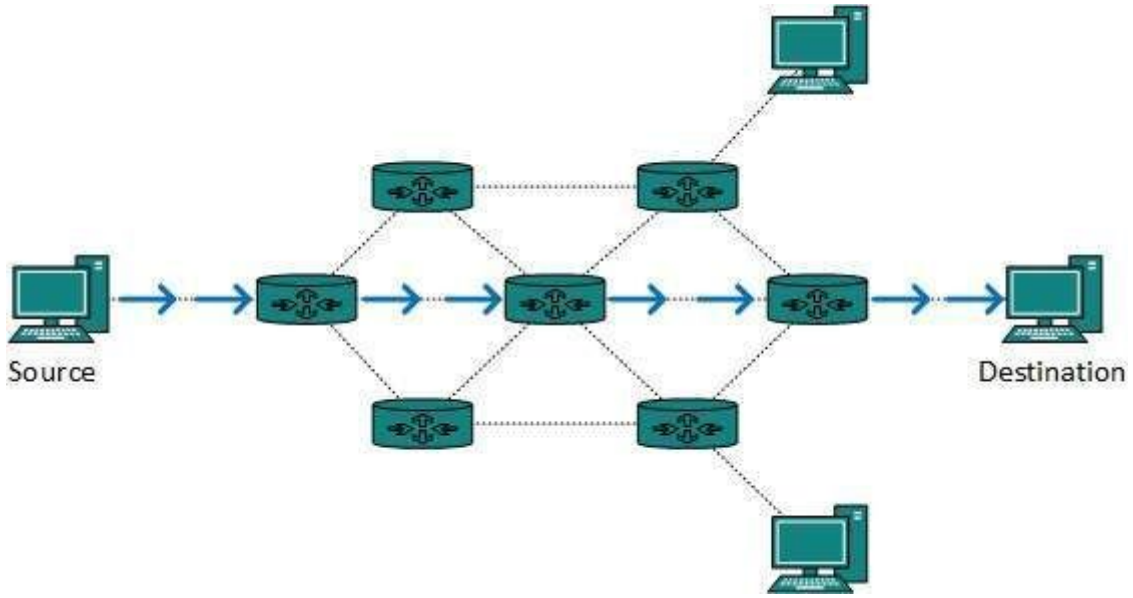

Fig 1: A naive representation of the routing process

2. Literature Survey

As per the current literature, the traffic engineering techniques [2] are moving from “stateless” traffic engineering to Network-State aware traffic engineering as this will provide greater flexibility and therefore greater and more optimized utilization of Network infrastructure. Most traffic engineering techniques used in SDN [3] take the view that SDN provides a global view of the current network state and topology in a logically centralized controller and therefore, optimization can be performed on the network by running different types of global traffic engineering algorithms.

Once the global topology of the network is available, the best path has to be selected which satisfies the specified QoS constraints. The problem of constraint based path finding [4] is

NP-Hard and cannot be used in the scenario under consideration as millions of decisions have to be taken in a very small time frame. Cognitive routing has been proposed in networks [1] which talks about using different types of packets to capture necessary information and distributed intelligence to make better routing decisions. This method has been tested and has shown performance improvements in traditional networks. With the advent of SDN and the separation of Control and Forwarding plane, such a service can be further improved upon by utilizing centralized control and network state.

Recent advances in Reinforcement Learning Algorithms such as Deep Q Networks [7] have opened the doors to approaching newer problem domains using reinforcement learning. Although deep Q learning incorporated an efficient technique to capture the infinite state space problem, the game solving which was intended to be solved by [8] had really small action spaces. Recent works such as [9] and [10] have suggested methods to tackle large action spaces as well.

One important contribution to reinforcement learning algorithms has been in [11] called the Deep Reinforcement Relevance Network. This has formed the basis for my work along with ideas and concepts learnt from the aforementioned research articles.

The following section throws light on certain prerequisites by giving a general background that the reader is expected to know before understanding the problem and its proposed solution.

3. Background

2.1 Software Defined Networking

Software Defined Networking is an upcoming architecture for networking that separates the control plane from the data/forwarding plane. This separation leads to better control of the network by introducing programmability to networks via open interfaces. It also provides an abstracted view of the underlying infrastructure to the applications and network services. Thus, through a greater support for dynamic network configuration this paradigm aims at making the network more manageable and adaptive.

Therefore, the advent of SDN has opened up the possibility of development of several new algorithms for optimizing the working of a network like routing algorithms, network assisted load balancing algorithms, etc.

2.2 Cognitive Systems

Cognitive computing is the simulation of human thought processes in a computerized model. Cognitive computing involves self-learning systems that use data mining, pattern recognition and natural language processing to mimic the way the human brain works. The goal of

cognitive computing is to create automated IT systems that are capable of solving problems without requiring human assistance.

Cognitive computing systems use machine learning algorithms. Such systems continually acquire knowledge from the data fed into them by mining data for information. The systems refine the way they look for patterns and as well as the way they process data so they become capable of anticipating new problems and modeling possible solutions.

2.3 Q-Learning

Q-learning is a model-free reinforcement learning technique. Specifically, Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). In simple words, a markov decision process (MDP) is a process from which the action depends only on the current state. Q learning works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. A policy is a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state.

Q-Learning is formulated by the famous bellman equation as follows.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Here,

1. The learning rate alpha determines how much of the difference (between actual value vs current Q(s,a) value) are we storing back into the function value, i.e, how quickly or slowly do you want to learn from the error.
2. The reward r_{t+1}, as we can see is the reward observed at time t+1. Once an action is chosen at time t using the state s_t, the reward is captured at time t+1.
3. The discount factor is how much into the future the current action at the current state matter. If this value is zero, the network will essentially learn how to predict the reward by looking at state and action only.
4. The learned value contains the estimate of the optimal future value (if we think recursively that will contain the reward in future + maximal Q value of it's future and so on..). In this way, when the Q values are finally learned, the Q value corresponding to a particular state and action will describe the quality/utility of the given action in a given state s.

One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment. It has been proven that for any finite MDP, Q-learning eventually finds an optimal policy, in the sense that the expected

value of the total reward return over all successive steps, starting from the current state, is the maximum achievable.

If the number of states and actions are small enough, the Q values can be calculated just by using a table with states as rows and actions as columns. This equation is unlike any other normal equation because it is recursively defined, i.e., the Q value for a particular state and action depends on the maximum Q value of next state over all possible actions. To go ahead with the table filling approach, we initialize the table with random values and then update the entries using the equation as and when we encounter a state and an action. An example in [13] explains how Q table is updated till convergence.

But even if one of them are infinitely many, it can clearly be seen that such a computation is infeasible. This is where Deep Q learning steps in.

2.4 Deep Q Learning

Neural networks are exceptionally good at coming up with good features for highly structured data. For example, when the state space becomes infinitely many, there are many states which are similar to each other. The Neural Networks generalize over this and consider them similar state and hence the term function approximation using neural network. We could represent our Q-function with a neural network, that takes the state and action as input and outputs the corresponding Q-value. Alternatively we could take states as input and output the Q-value for each possible action. This approach has the advantage, that if we want to perform a Q-value update or pick the action with the highest Q-value, we only have to do one forward pass through the network and have all Q-values for all actions available immediately. But this approach would deem infeasible, as the no. of actions increase. As will be discussed in the later parts of the reports, the generic action space is exponential with the no. of links present in the computer network and hence there is a need to use an architecture similar to the former one.

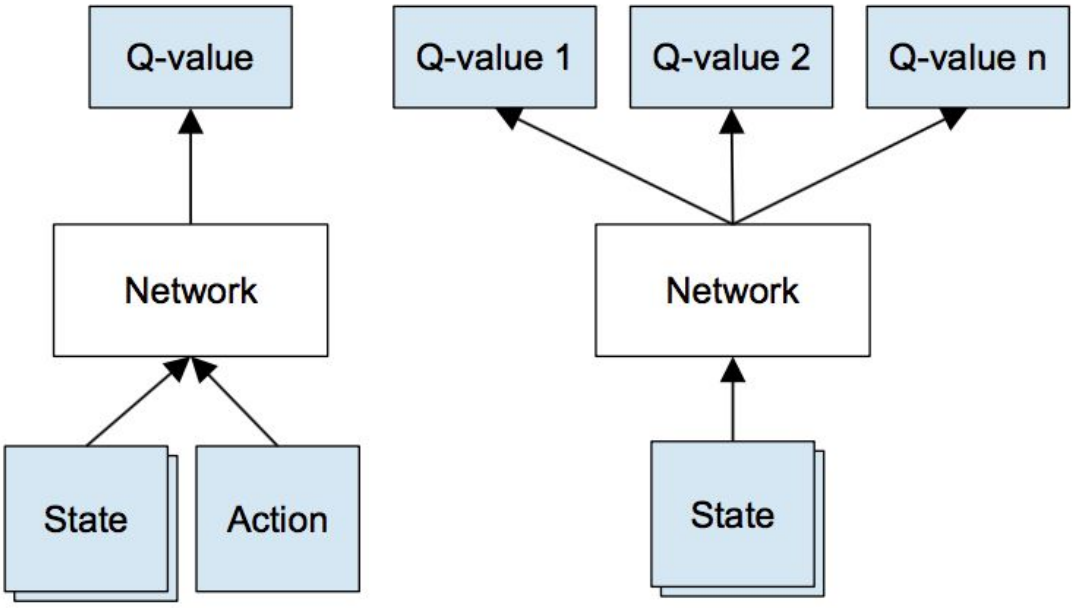

Fig2 : Possible Neural Network Architectures for Deep Q Learning

These neural networks are trained using back propagation on the loss function which is formed using the bellman equation as:

$$L = \frac{1}{2}[\underbrace{r + max_{a'}Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Eventually, the neural network will learn to approximate the Q-values just by using the **current state and action, instead of R + max future estimate** and this will then lead to optimal decisions being taken by selecting the action corresponding to the given state.

In a nutshell, this project aims at **creating a cognitive system** that seeks to draw inferences and **produce actionable outcomes** by **using the data** generated by exploiting the programmability offered by **Software Defined Networking** and powerful tools like **Deep Q learning**. The following section details the problem statement along with a list of objectives that the whole approach to this problem can be broken down into.

4. Problem Statement and Objectives

"To conceptualize, design and implement a cognitive traffic routing system for software-defined networks which utilizes the network state information and reinforcement learning algorithms to provide better QoS to end users"

Objectives:

- Study the existing research papers related to the given problem statements to identify the scope of improvements.
- Develop a system to collect appropriate network state data
- Design a cognitive system to make sense of the network data and produce actionable outcomes.
- Develop a cognitive routing service which aims to improve end-user experience by maximizing the network utilization.
- Integrate the developed service into the available network framework for testing and analysing the results produced.

5. Modelling the problem using Deep Reinforcement Relevance Networks

A novel architecture of deep Q networks has been proposed in [11]. This separates the state network from the action network and essentially seeks to find the relevance between a state and an action. The following figure depicts the architecture and comparisons with per-Action and Max Action network.

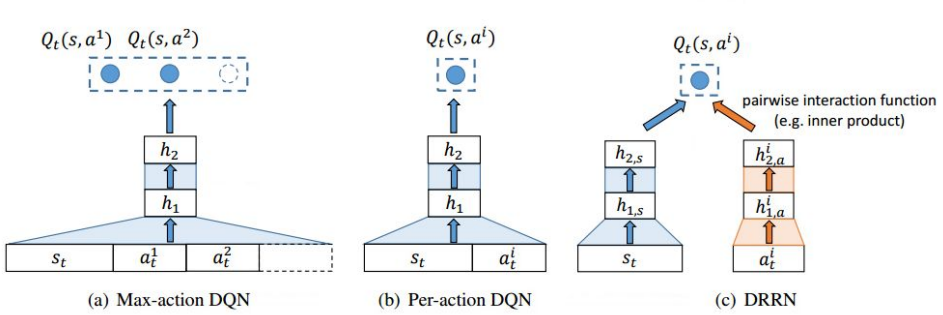

Fig 3: Possible Architectures of Deep Q networks

This neural network takes in State s_t and action a_t^i as inputs and produces the Q value using the dot product of hidden layers output of the respective neural networks. In the context of the problem, let us define the state and action.

5.1 State Representation

The state of the system is depicted using a 1-D vector that consists of link data in the computer network. This vector consists of both current bandwidth utilization and current delay of each and every link in the network.

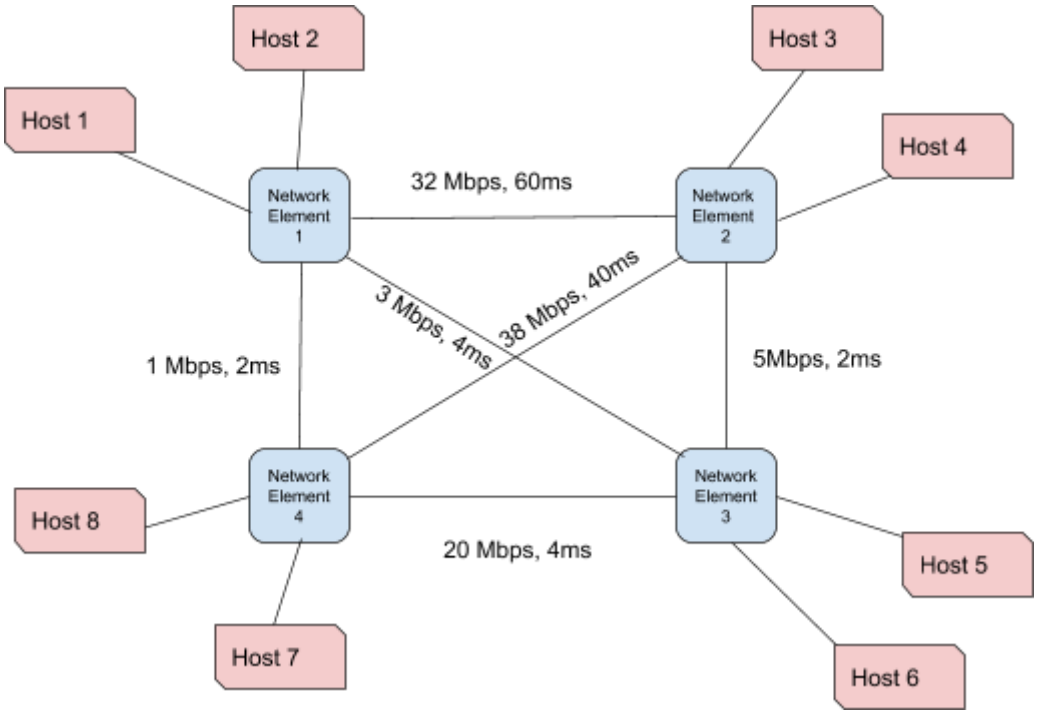

Fig 4: Network Representation

For the depiction in the above figure, (even though for pictorial simplicity only uni-directional weights have been shown, consider that the links are bidirectional have separate weights)
State-Vector :=
[32 3 1 23 5 38 6 8 20 2 43 12 0.06 0.004 0.002 0.01 0.002 0.04 0.008 0.04 0.006 0.08 0.009]

The first half of the 24 length vector has bandwidth information of 12 (=6*2 since bidirectional) links and second half contains their delay values.

5.2 Action Representation

Once we know the given state, the action that has to be taken is the selection of a path. Ideally, an optimal action would be the one which gives the best path in the current scenario for any request. As we can see that, the problem chosen is actually an MDP (without mathematical proof but using our intuition) since the action depends only on the current state and not any other previous state. The action representation in the above-mentioned scenario would be a 1-Dimensional Vector of length 12 an example would be:

Action vector := **[0 0 0 1 0 0 0 0 1 0 0 1]** <-- Path containing 3/12 links

Since, the action neural network is separate from the state network, we can pass the actions for feasible paths only and then use their q-values to determine the best path among the feasible paths.

5.3 Reward Function

An important part of Deep Q network is the reward structure, the reward for the actions i.e. the paths in the network must indicate how good the current path is given the state of the network. To calculate reward, we need to enforce the path (action) into the network and then observe the state of the network to know how well the action which was taken is performing. To compute the reward, both bandwidth utilization and delay minimization are considered just for the path being selected. The detailed code of the reward function and it's explanation are as follows:

```
def get_reward(state1,action,state2):
    MAX_BW = 50
    MIN_DEL = 0.002
    bw1 = state1[12:]
    delay2 = state2[:12]
    bw2 = state2[12:]
    residual_path_bw_utilized = np.dot((bw2 - bw1),action)/sum(action)
    path_delay = np.dot(delay2,action)

    if residual_path_bw_utilized > MAX_BW * (3.0/4):
        r = 1
    elif residual_path_bw_utilized > MAX_BW / 2.0:
        r = 0.75
    elif residual_path_bw_utilized > MAX_BW / 4.0:
        r = 0.50
    elif residual_path_bw_utilized > MAX_BW / 8.0:
        r = 0.25
    elif residual_path_bw_utilized > MAX_BW / 12.0:
        r = 0.125
    else:
        r = 1

    if path_delay < 0.005:
        r *= 1
    elif path_delay < 0.008:
        r*=0.75
    elif path_delay < 0.015:
        r*=0.50
    elif path_delay < 0.06:
        r*=0.25
    elif path_delay < 0.12:
        r*=0.125
    else:
        r*=0.05
```

The reward function `get_reward` computes the mean residual bandwidth utilized by the chosen path, if it is greater than a fraction of the maximum bandwidth of the link appropriate reward is given. To add the effect of delay, the reward is multiplied by a fraction that decreases as delay increases. With the reward value calculated, the next step is to know how to make the DRRN learn.

5.4 Learning Algorithm of DRRN

1. Initialize recent buffer *RB* and history buffer *HB*
2. Initialize DRRN with small random weights
3. Start network simulation and initiate host discovery
4. Initialize REST framework for enabling data push and query support
5. Read network graph and precompute possible optimal paths (actions)
6. Start network simulation with random requests with an initial random seed
7. For every request:

    Get network state s_t

    Compute $Q(s_t, a^i; \Theta)$ for the list of paths (a^i feasible for the given request

    Draw random sample probability p:

    If p>epsilon: select path that gives maximum q

    Else: select a path randomly

    Reply to request with the selected path

    Observe the new network state ns_t after selecting the path.

    Observe the reward *R* received and sample next probable request *req.*

    Store transition (st,ai,*R,*nst,*req)*

    If the recent buffer is filled:

        Load batch data from recent buffer RB

        For tuple k in batch data:

            Set y_k *to be* $R_k + \gamma \max_{a' \in A\ reqk} Q(nst_k, a^i; \Theta))$

        Perform a gradient descent step on mean squared error with respect to the network parameters Θ. Back-propagation is performed only for a^i *even though there are* $|Areq_k|$ *actions at time k.*

    If the history buffer is filled enough and with random probability 0.1:

        Load batch data from history buffer HB

        For tuple k in batch data:

            Set y_k *to be* $R_k + \gamma \max_{a' \in A\ reqk} Q(nst_k, a^i; \Theta))$

        Perform a gradient descent step on mean squared error with respect to the network parameters Θ. Back-propagation is performed only for a^i *even though there are* $|Areq_k|$ *actions at time k.*

5.4.1 Intuitive Explanation of Training Process

Initially, an untrained model guesses path to be good among the feasible paths. The feasible path includes the paths which actually contain the source and destination of the request. Once it performs this guess it receives a reward by looking at the state of the network which got affected due to its action (path choice). This reward is used to update the neural network to predict the Q-value using the loss function (defined in section 2.4 of this report). That is, the network tries to learn to predict the Q-value just by looking at state and action to be taken instead of always waiting for the reward. Initially, while selecting random paths, it gets to know and learn about the how the reward is being given and once trained it can predict

the Q-value i.e the utility of an action in a given state without knowing about the future state. Such learning will help the model look at the network state and choose a path that will have maximum utility.

This training is a continuous process and throughout the training process the value of epsilon reduces from 1 (Exploratory phase) to 0.1 (Exploitation Phase). In exploratory phase it tries to guess the paths but as it learns, it exploits it's learning at selects paths with maximum q values. Once we think that the model is trained enough, we can stop the training process and request the path just using the q values.

5.5 Connecting all components: The big picture

The following figure depicts how this neural network enabled cognitive model fits into the standard SDN architecture.

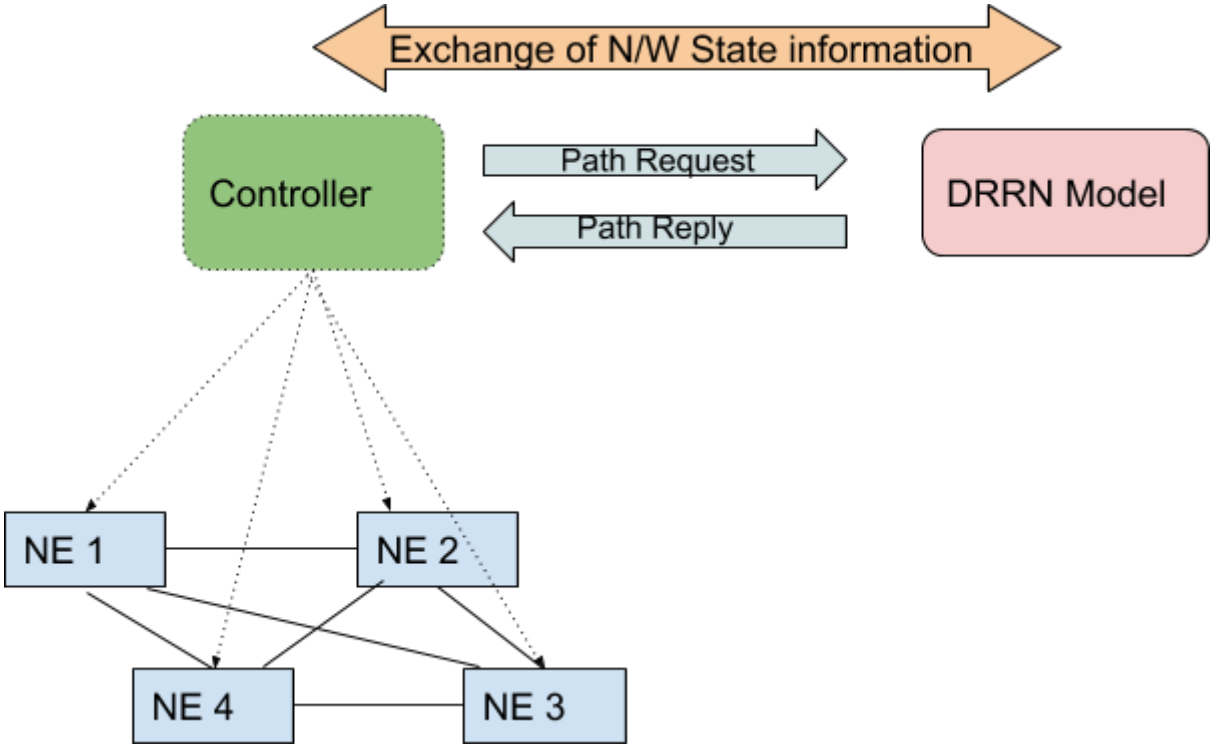

Fig 5: Connecting all components

As illustrated in the picture above, the controller is the brain of the network which manages all the networking elements (NEs). It collects the live state information and exchanges with the DRRN model. Whenever a new flow (series of packets) requests a path, the controller communicates to the DRRN Model through REST API and requests for a path. The DRRN model considers the current state and list of feasible paths and replies with the path that gives maximum Q value i.e. the math with maximum utility.

7. Results

The results section shows how the proposed scheme outperforms the shortest hop heuristic generally used by routing algorithms like OSPF and RIP.

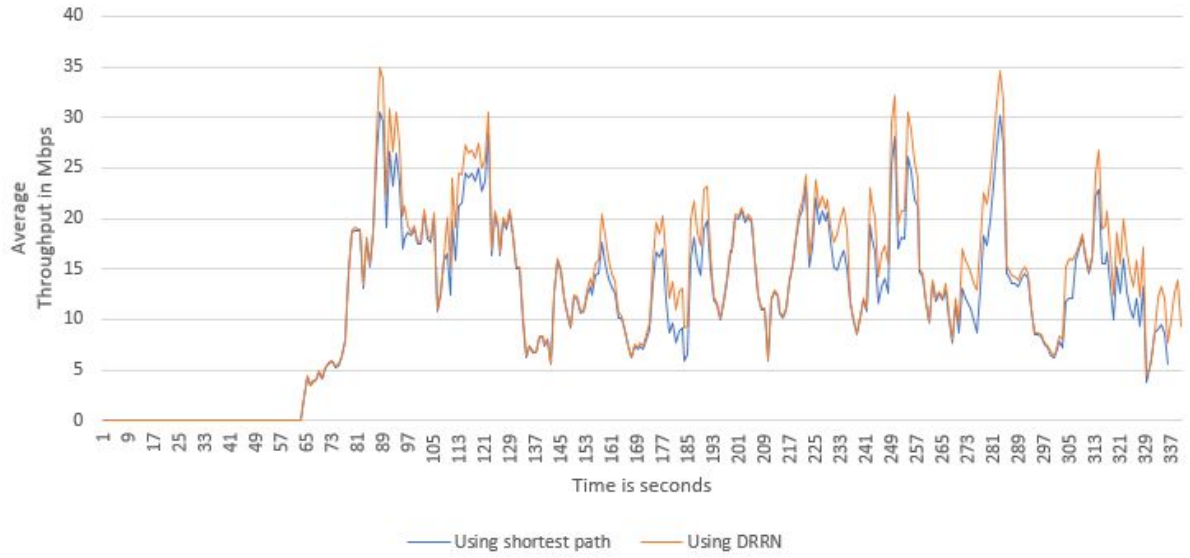

Fig 6a: This graph compares the average throughput of all the links for the same traffic scenario between shortest hop and proposed method

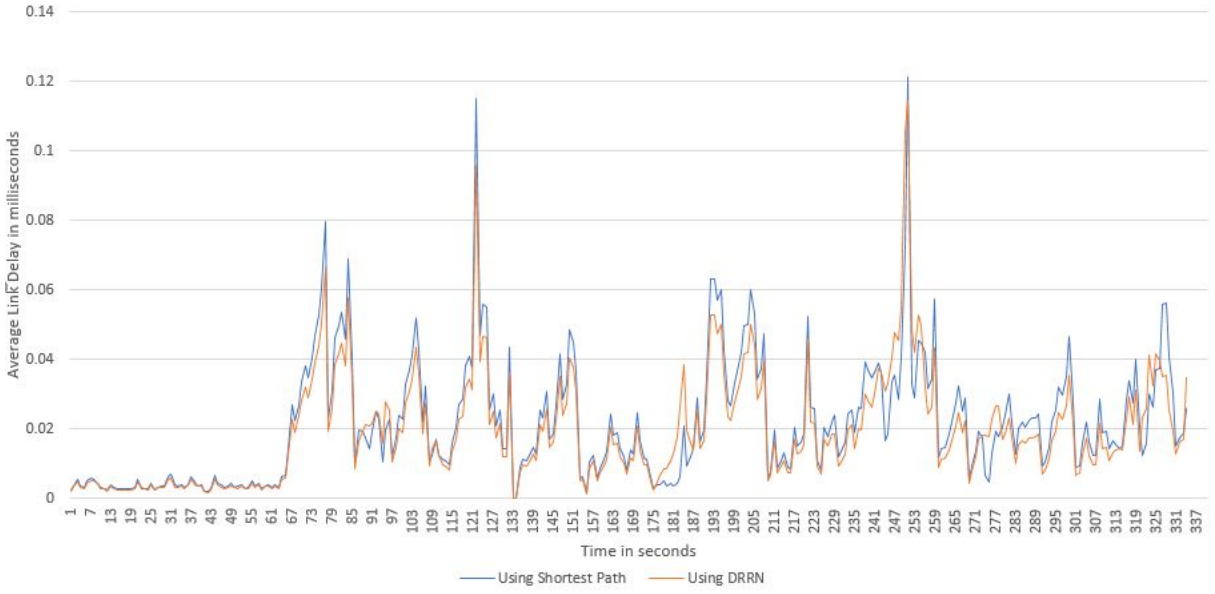

Fig 6b: This graph compares the average delay of all the links for the same traffic scenario between shortest hop and proposed method

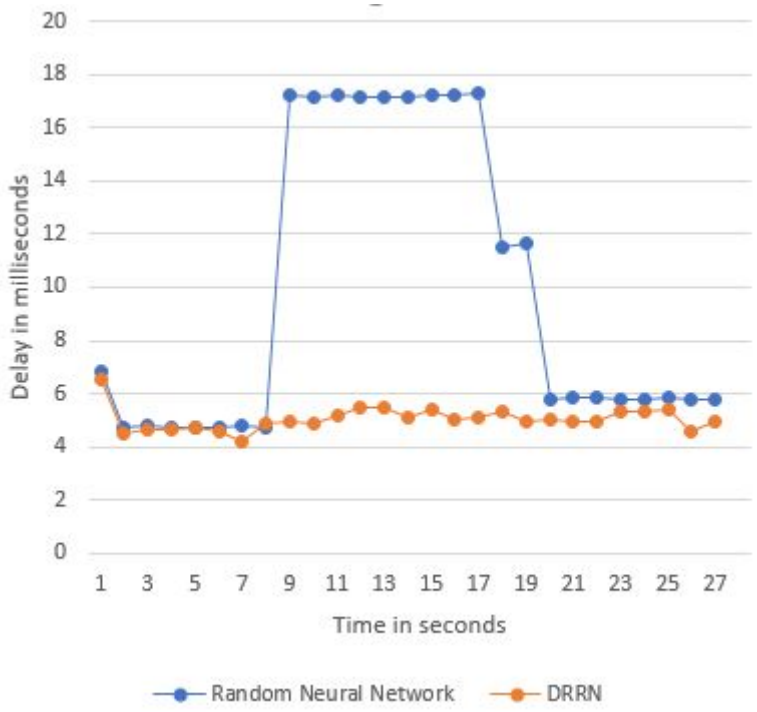

Fig 7: This graph depicts the comparison between RNN solution proposed in [12] and the proposed methodology of this project

Explanation: The results obtained in the comparison is due to the fact that the solution proposed in [12] assigns the shortest path to the request first and later updates the path and hence in that time observing a higher delay. Since the current algorithm looks at the state, noticing that it already has traffic assigns a new alternative path to the new request hence keeping the delay almost same for the initial flow.

The following graph shows the loss in terms of mean squared error that was plotted roughly after every 100 new samples have been added to the storage training buffer.

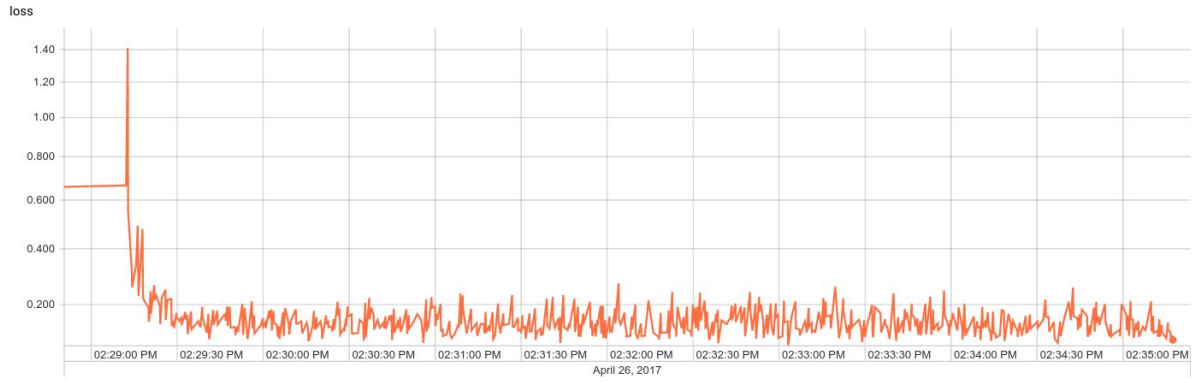

Fig 8: This graph depicts the loss measured in terms of mean squared error over training iterations

8. Conclusion

Through this project a novel routing mechanism is proposed using current reinforcement learning algorithms and exploiting the network state information gathered by employing the emerging paradigm of software defined networking. This work can be extended to larger networks by a following a hierarchical routing regime similar to the routing that happens in the internet. Moreover, the number of feasible paths can be reduced using bio-inspired algorithms or any other equivalent technique that can produce list of most feasible paths (say by ignoring paths of hop length > k etc.). Also, due the availability of powerful GPUs and their parallel processing capability the time taken to take the decision is independent of the number of paths as all the q-values are generated parally. Hence this makes the algorithm scalable and robust to larger network scenarios.

8. Tasks with Timeline

Timeline	Task
Week 1	Preliminary Literature Analysis
Week 2	Study of algorithms for pre-computation of possible optimal paths
Week 3	Developing the framework to monitor and gather network information
Week 4	Study of existing Reinforcement Algorithms (applications to routing)
Week 5	Design of Reinforcement Algorithm for constraint-based path selection
Week 6	Develop a framework to convert the paths found by the algorithm into the appropriate set of Openflow messages which create or update existing paths
Week 7	Testing and Analysis of results obtained

9. References

[1] Gelenbe, Erol, Ricardo Lent, and Zhiguang Xu. "Design and performance of cognitive packet networks." Performance Evaluation 46.2 (2001): 155-176.

[2] Basak, Debashis, Hema Tahilramani Kaur, and Shivkumar Kalyanaraman. Traffic engineering techniques and algorithms for the Internet. Tech report, Rensselear Polytechnic Inst, 2002.

[3] Akyildiz, Ian F., et al. "A roadmap for traffic engineering in SDN-OpenFlow networks." Computer Networks 71 (2014): 1-30.

[4] Kuipers, Fernando, et al. "Performance evaluation of constraint-based path selection algorithms." IEEE network 18.5 (2004): 16-23.

[5] Potvin, Jean-Yves. "A review of bio-inspired algorithms for vehicle routing." Bio-inspired algorithms for the vehicle routing problem. Springer Berlin Heidelberg, 2009. 1-34.

[7] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

[8] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.

[9] Dulac-Arnold, Gabriel, et al. "Deep Reinforcement Learning in Large Discrete Action Spaces." arXiv preprint arXiv:1512.07679 (2015).

[10] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[11] He, Ji, et al. "Deep Reinforcement Learning with an Action Space Defined by Natural Language." *arXiv preprint arXiv:1511.04636* (2015).

[12] Francois, Frederic, and Erol Gelenbe. "Towards a cognitive routing engine for software defined networks." Communications (ICC), 2016 IEEE International Conference on. IEEE, 2016.

[13] Path Finding using Q Learning
URL: http://mnemstudio.org/path-finding-q-learning-tutorial.htm