Towards Opportunistic Flow Management in OpenFlow

Seung Il Moon, Rossi Kamal, Choong Seon Hong and Sungwon Lee
Department of Computer Engineering
Kyung Hee University
Email:{moons85, rossi, cshong and drsungwon}@khu.ac.kr

***Abstract*—Traditional Internet is merely static, rather than flexible, which is demanded by Internet-Community. OpenFlow has opened the door of management of Internet in a more controlled way. In this context, we are motivated to manage OpenFlow in a more opportunistic way, which encourages flexible network-management by using experience, through trial and error-mechanisms. In this paper, we have proposed a Q-Learning(Q-L) based opportunistic flow management algorithm OpenFlow-QL, by which NOX(Network Operating System) Controller maintains the flow by considering QoS among OpenFlow switches. We have implemented OpenFlow-QL through a network architecture, where Agent visualizes the opportunistic flow, given by the NOX Controller through VN (Virtual Network) Manager.**

I. Introduction

Ongoing penetration of Internet-services demand flexible architecture, which can be controlled or programmed dynamically. OpenFlow[1] opens this door for developing new and flexible network technology, by moving from proprietary network to open network.

OpenFlow, living under the umbrella of SDN(Software Defined Network)[2] technology, separates Control Plane and Data Plane. OpenFlow is equipped with an open protocol to maintain dynamic/programmable flow-table[3] between different switches and routers.

We intend to manage optimal flow in OpenFlow-based network with an opportunistic way, which uses experience, by trial and errors, to avoid congested/erroneous path. We have adopted Q-learning algorithm as it does not use any environmental model and is capable to learn to achieve the goal from experience.

We first propose a Q-Learning[4] based centralized opportunistic flow management algorithm OpenFlow-QL. In OpenFlow-QL, a node learns to choose flow by considering the pay-off for QoS properties. Performance evaluation is presented to show how OpenFlow-QL outperforms link-state-routing.

We then apply OpenFlow-QL through our OpenFlow-based network architecture. At first, administrator requests for new path creation through Agent. Agent requests the VN Manager about optimal path. Then, VN Manager communicates with NOX Controller to get the Q-table. NOX Controller[5] collects QoS status from OpenFlow switches and executes OpenFlow-QL and then updates Q-table. Then, NOX Controller responses with optimal pathflow to the Agent, through VN Manager.

II. System Architecture

The system architecture consists of major modules, namely *Agent*, *VN(Virtual Network) Manager* and *NOX Controller* (Fig.1).

Agent consists of sub-modules, namely Path Management Program, View Network Information and Path Management, respectively. Path Management program makes HTTPS connection with the *VN Manager*. View Network Information provides the option to visualize topology, flow and link information. Path Management initiates the path creation/deletion request.

VN Manager consists of sub-modules, namely Connections, Request Msg Analyzer, VN-API and Network Information and Database. Connections can make HTTPS connection with *Agent* and NOX Connection with *NOX Controller*. Request Msg Analyzer analyzes the message received from the *Agent*. VN API, by using Path Flow Optimizer, extracts the Q-table information from NOX Controller. Network Information keeps the topology, flow and link information.

NOX Controller consists of sub-modules, namely NOX App, pyswitch, NOX and OF Secure Channel. Q-Manager in NOX App receives message from NOX API and responses with Q-table. Dispatcher is used as an interface, processor is used as execution unit and Q-table is database of reward and state-action pairs. Pyswitch notifies OpenFlow switches to be updated with the optimal path and also forwards the updated-information to *VN Manager*.

Administrator may request for path-creation or deletion or topology-visualization or link-state information to the *Agent*. Path Management Program at *Agent* forwards this request message to *VN Manager* through a HTTPS Connection. Request Message Analyzer at *VN Manager* receives it and calls VN(Virtual Network) API to forward to handle the request. VN API, at first, converts this message so that NOX application can recognize it and sends it through NOX Connection.

NOX application processes the request and then sends result to VN API. VN API converts the result in XML format and sends to Path Management Program at *Agent* through HTTPS Connection. Path Management Program, by using received XML, can visualize network information coming through. This interactive GUI environment shows the flow-management in the OpenFlow network.

NOX Controller implements OpenFlow-QL(described in the next section) by Q-Manager module. It has sub-modules,

978-3-901882-50-0

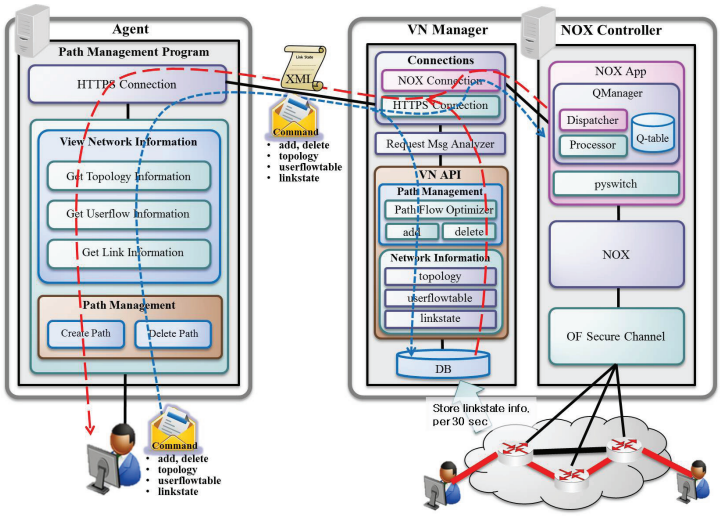

Fig. 1. System Architecture

namely dispatcher, processor and Q-table. Dispatcher receives request about Q-table from VN API and then collects current QoS status from the OpenFlow switches. Then, processor executes OpenFlow-QL algorithm and updates Q-table by using QoS status.Q-table contains the state, action and reward for different flow-choices.

III. Q-Learning-Based Opportunistic Flow Management for OpenFlow

In this section, we propose our Q-Learning based opportunistic flow management algorithm OpenFlow-QL(Algorithm 1). OpenFlow-QL is a centralized algorithm to be run by NOX controller. Agent requests to NOX-controller about optimal flow from source to destination OpenFlow switch. NOX-controller always runs OpenFlow-QL in itself and determines the requested optimal flow and then responses it to Agent. We have described state, action, reward etc of OpenFlow-QL in Table 1.

In this algorithm, at first stage(initialization), state transition probability and reward are initialized. State transition gives NOX controller the idea about how the OpenFlow switches will turn to a new QoS-status, by choosing path to its neighbors, from its original QoS-status. Reward relates to the cost paid to obtain the optimal flow with QoS-guarantee.

At second stage(processing), NOX controller calculates QoS-based reward for each OpenFlow switch, considering each flow to neighbors. For each OpenFlow switch i, this reward is calculated by summing the value and weight-factor of j QoS-parameters. Elimination factor is used so that we might either consider or ignore any QoS-parameter. Normalization factor is used for consistency among different QoS-parameters. Thus, NOX controller maintains a Q-table with QoS-based reward, action, state for each OpenFlow switches.

At third stage(activation), NOX controller chooses optimal flow/path for OpenFlow switches. It means that when any Agent requests optimal flow from source to destination OpenFlow switches, NOX controller immediately chooses path/flow by looking from its Q-table.

TABLE I
Description of OpenFlow-QL

Q-Learning	OpenFlow-QL
Policy (π)	Opportunistic Flow Management
State (s)	QoS-guaranteed/QoS-failed
Action (a)	Choose Optimal Path
Reward (r)	Optimal flow considering QoS-factors(bandwidth, delay, packet-delivery ratio etc.)
Cost (c)	Pay-off paid to guarantee each QoS factor)
State Transition ($s^{'}\|s, a$)	Probability of resulting in a certain QoS-state, by choosing an optimal path at present QoS-state)

At last stage(balancing), NOX Controller updates its Q-table by observing the immediate reward and future expected rewards. Immediate reward means how the corresponding OpenFlow switch is immediately benefited in terms of QoS. Future expected reward indicates by following this strategy, how the OpenFlow switch is expected to be benefited in terms of QoS in consequent flow management. Discount factor indicates how many future iteration we might consider in the future reward estimation. At last, the resulting QoS-status is turned to present QoS-status for the next iteration.

Iteration(Processing-Balancing) continues until NOX controller receives a request from Agent for flow selection among OpenFlow switches. Administrator, through NOX-controller, can initialize/redefine the state transition probabilities and rewards by observing how the OpenFlow-QL is performing by observing network for certain period of time.

IV. Simulation Results

In this section, we evaluate the performance of OpenFlow-QL in Matlab. First, we compare OpenFlow-QL with Link-State-Routing(LSR), considering flow management in terms

Algorithm 1 OpenFlow-QL

1. Initialization Initialize $p(s^{'}|s,a), R_{QoS}(s,a)$
2. **while** NOX-controller receives Optimal-Flow selection request from Agent **do**
3. Processing NOX-controller calculates QoS-based reward value(R_{QoS}) for each node i as follows $(R_{QoS})_i = E\sum_j m_j N(v_j)$, where
   v_j=cost value of j-th QoS parameter,
   m_j=weight value of j-th QoS parameter,
   E =elimination factor of network,
   N=normalization factor
4. Activation
   Choose path selection action a
   [Q table maintains actions for $(R_{QoS})_i$ entry of each i]
5. Balancing
   Upgrade $Q(s,a)$ for all nodes as follows,
   $Q(s,a) = r_{QoS}(s,a) + \gamma max_{'}\sum_a p(s^{'}|s,a)Q(s,a)$
   $s \leftarrow s$
6. **end while**

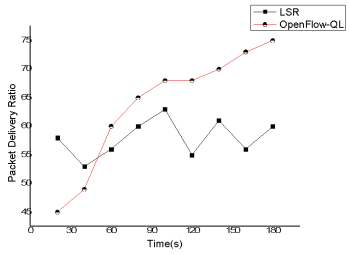

Fig. 2. Comparison of OpenFlow-QL with LSR

of one QoS-metric namely delivery-ratio(Fig. 2). OpenFlow-QL shows better packet delivery ratio than LSR, although at beginning its performance is worse. However, as time goes, OpenFlow-QL shows better delivery ratio, as NOX Controller learns from experience about which flow can yield better performance in future. Then, we perform simulation

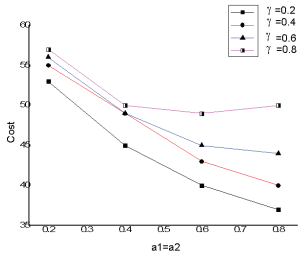

Fig. 3. Cost analysis for different discount factors

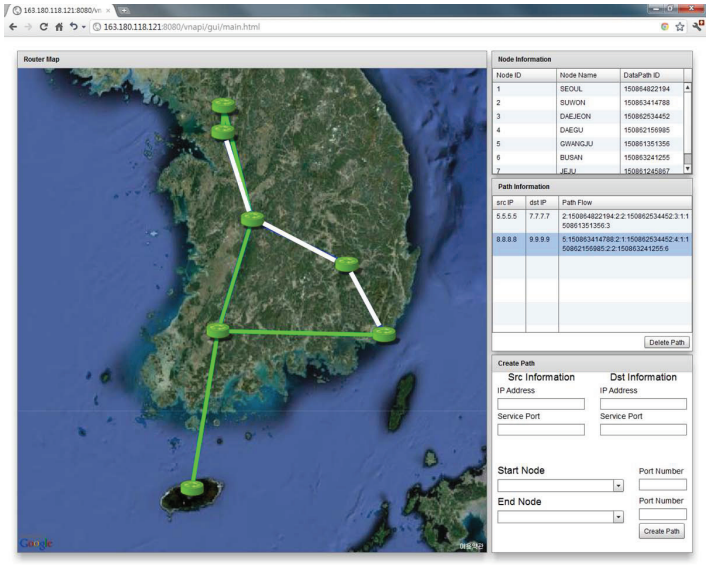

Fig. 4. Path Management Program

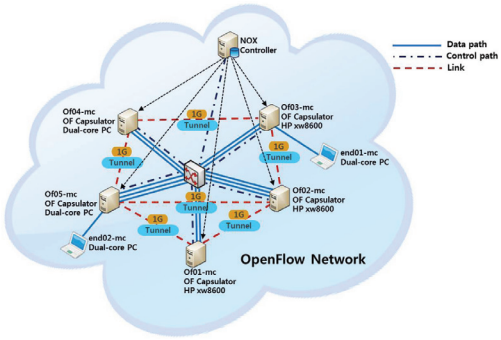

Fig. 6. Testbed Environment

to observe the cost function of OpenFlow-QL(Fig. 3), with respect to different value of discount factor. Let us consider cost, $r = a1b1 + a2b2$, where a1, a2 are QoS-values and b1, b2 are QoS-weight-vectors, of two example QoS parameters x,y, respectively. By keeping b1=b2 a constant value, we have observed that if the value of discount factor is increased, we get benefit in terms of cost. It means that if more future state transitions are considered(as pi increases), NOX Controllers reward for QoS management is increased as well. It means that NOX Controller gets more benefit by saving cost for flow-management.

V. Implementation

In this section, we describe how OpenFlow-QL is applied in path selection in the proposed system(Fig. 5). In this context, implementation status, including testbed environment(Fig. 6), application platform(Table II), Path Management program(Fig. 4) are presented.

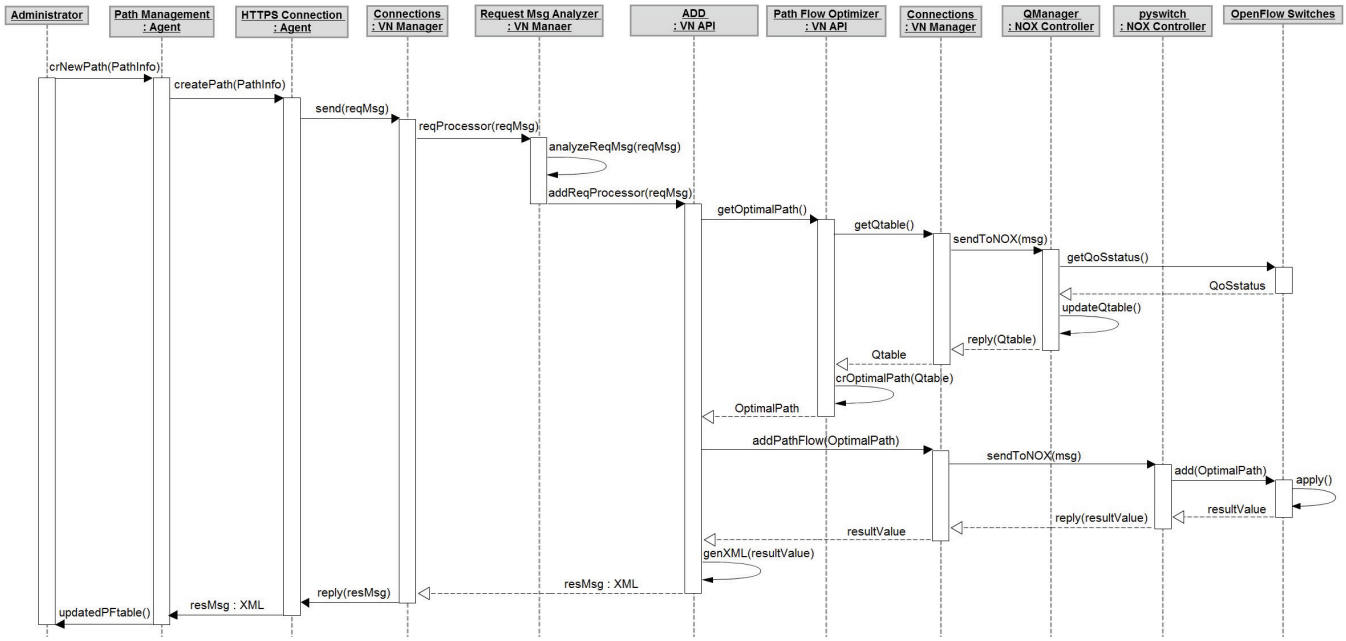

Fig. 5. System-workflow for OpenFlow-QL-based Path Creation

A. Application of OpenFlow-QL in Path Creation in the Proposed System

We describe the process of path flow creation using OpenFlow-QL in the proposed system(Fig. 4).

Administrator at first generates path-creation-request. Path Management program at Agent then forwards it to VN Manager through a HTTPS Connection. VN Manager analyzes the request and takes assistance of VN API through path flow optimizer. This optimizer collects the updated Q-table (of OpenFlow-QL) from NOX Controller. NOX Controller, in regular interval, updates Q-table with QoS status from OpenFlow switches. Then, VN API, after being updated with latest Q-table, notifies OpenFlow switches about optimal path through pyswitch.

B. Implementation Status

We have summarized application platform in Table II and Testbed Environment in Fig.6. Path Manager Program at Agent is developed in Java, having interactive Flex-made GUI(Fig. 4) that helps administrator to visualize opportunistic flow established by OpenFlow-QL.

TABLE II
APPLICATION PLATFORM

Ubuntu 10.4
NOX Controller version-zaku
Java v.1.6
Apache tomcat v.7.0
Python v.2.6
MySQL v.5.1
Adobe Flex 4.5 SDK

VI. CONCLUSION

In this paper, we have proposed and then applied a centralized Q-learning algorithm OpenFlow-QL, for opportunistic flow management in OpenFlow-based network. We have a plan to apply OpenFlow-QL in a distributed way among OpenFlow switches.

In this paper, our Q-Learning algorithm is based on one policy regarding QoS. With a view to develop more flexible and programmable network, we have plans to incorporate other policies related to security, safety in OpenFlow-QL. It will be very challenging, as there will be questions regarding conflicts among different policies.

ACKNOWLEDGEMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2012-0006421). Dr. CS Hong is the corresponding author.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] ONF, "Software-defined networking:the new norm for networks," *ONF(Open Network Foundation) White Paper*, 2012.

[3] H. Jin, D. Pan, J. Liu, and N. Pissinou, "Openflow based flow level bandwidth provisioning for cicq switches," in *INFOCOM, 2011 Proceedings IEEE*, april 2011, pp. 476 –480.

[4] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[5] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.