An Agent-Based Autonomic Network Management Architecture

By

DANIEL BRYAN MARCONETT
B.S. (California State University, Sacramento) 2006
M.S. (University of California, Davis) 2008

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA, DAVIS

Approved:

________________________________________
Dr. S. J. Ben Yoo, Chair

________________________________________
Dr. V. Rao Vemuri

________________________________________
Dr. S. Felix Wu

Committee in Charge

2014
Abstract

Today’s network technology environment is varied and diverse. Disparate technologies, such as Optical Fiber, DSL, 802.11 Wireless LANs and Wi-Max, are deployed in varying localities, sometimes simultaneously, and expected to work seamlessly through the use of the TCP/IP protocol stack. While TCP/IP has allowed for substantial interoperability among such technologies, there remain limitations. TCP’s incorrect inference of link congestion as the sole source of packet losses in wireless networks is one such example [1, 2]. Network management of diverse network deployment scenarios, be they multi-hop wireless, wire-lined or heterogeneous environments, point to the need for fully automated network management to comprehend the underlying technologies used, their inherent limitations, so as to fully maximize the optimizations performed in terms of policy-based management decisions. Such a feat is by no means trivial to engineer. However, this dissertation endeavors to illustrate a conceptual framework, coupled with experimental design and analysis, in order to convey the feasibility and utility of this approach, namely the approach of taking the human out of the decision-making loop in network management systems.

In this dissertation, we study cognitive agent-based optimization of network processes in three scenarios: a) choosing ideal cluster-head nodes in purely multi-hop wireless network
deployment based upon stability metrics, b) dynamically tuning the update interval in the AODV routing protocol to lessen the overhead and increase overall packet delivery rates, and c) introducing multi-domain intelligent resource brokering, leveraging machine learning techniques to improve the decision making abilities of the agents. And in all three cases we illustrate the ability of these network agents to observe network state, leverage machine learning to extract understanding from those state observations, and determine configuration decisions in an optimized and autonomous fashion.
Acknowledgements

I would like to thank my advisor and committee chair Prof. S. J. Ben Yoo for his constant guidance and infinite patience over the past several years. If it were not for his steadfast support of my research, my graduate career would not have been possible. Prof. Yoo was always available for meaningful discussions regarding the various research problems I worked on, and leveraged his vast research expertise and technical background to help me become and far more focused and disciplined researcher and engineer. Words cannot express my gratitude to Prof. Yoo as both an advisor and a mentor.

I would also like to thank Prof. V. Rao Vemuri and Prof. S. Felix Wu for their willingness to serve on my committee and provide valuable feedback and advice regarding my research.

I would also like to thank my wife Tessa, without whom higher education would have been far more difficult, if not impossible, to obtain. Her love and patience, along with that of Mason, Shirin and Saeed, have not only encouraged me to pursue my research, but have made it possible to have the time, support, and patience necessary to finish my degree, and be a better husband, father, and son. This is for my family.
Contents

Chapter 1 Introduction ............................................................................................................. 1
  1.1 Autonomics and Cognitive Network Systems ................................................................. 1
  1.2 Organization .................................................................................................................. 5

Chapter 2 Case Study: Stability-Based Clustering in Multi-hop Wireless Networks ....... 7
  2.1 Introduction .................................................................................................................... 7
  2.2 Existing Clustering Techniques ..................................................................................... 9
  2.3 Reinforcement Clustering ............................................................................................ 13
     2.3.1 Reinforcement Learning ......................................................................................... 13
     2.3.2 Reinforcement Learning-based Clustering ............................................................ 15
  2.4 Methodology and Results ............................................................................................. 20
     2.4.1 Clusterhead Change ............................................................................................... 20
     2.4.2 Cluster Count ......................................................................................................... 22
     2.4.3 Implications of Transmission Range and Nodal Density ....................................... 24
  2.5 Conclusion .................................................................................................................... 25

Chapter 3 Self-Adapting Protocol Tuning for Heterogeneous Networks Using Q-Learning 26
  3.1 Introduction ................................................................................................................... 26
  3.2 Related Work ................................................................................................................ 29
     3.2.1 Cross-Layer Approaches for Intelligent Network Management in Wireless Networks 29
     3.2.2 Challenges in Multi-Hop Wireless Networks ........................................................ 30
     3.2.3 Existing Protocol Parameter Tuning Solutions ....................................................... 34
  3.3 Self-Configuration for AODV ....................................................................................... 38
     3.3.1 Self-Configuration Parameters for AODV ............................................................ 38
     3.3.2 Q-Learning in Multi-hop Wireless Routing ......................................................... 39
     3.3.3 Cross-Layer Autonomic Network Management Architecture ............................... 40
  3.4 Q-Learning Based Self-Configuration (QLS) for AODV ............................................ 44
     3.4.1 Q-Learning ............................................................................................................ 44
     3.4.2 Q-Learning Based Self-Configuration ................................................................. 45
  3.5 Simulation Environment ............................................................................................... 48
  3.6 Performance Evaluation ............................................................................................... 50
     3.6.1 Responsiveness ..................................................................................................... 50
     3.6.2 Routing Overhead .................................................................................................. 53
     3.6.3 Packet Delivery Ratio ............................................................................................ 55
     3.6.4 Impact of Learning Rates ....................................................................................... 57
  3.7 Conclusion .................................................................................................................... 61

Chapter 4 FlowBroker: A Software-Defined Network Controller Architecture for Multi-Domain Brokering and Reputation ................................................................. 62
  4.1 Introduction .................................................................................................................... 62
  4.2 Background and Related Work .................................................................................... 65
     4.2.1 Distributed SDN Controller Implementations ...................................................... 66
4.2.2 Issues in Multi-Domain SDN ................................................................. 70
4.3 FlowBroker ........................................................................................................ 73
  4.3.1 FlowBroker Architecture ............................................................................. 74
  4.3.2 Load-Balancing and Forwarding with FlowBroker ..................................... 78
4.4 Broker Reputation ............................................................................................ 83
  4.4.1 Broker Performance Reputation .................................................................. 84
  4.4.2 Linear Discriminant Analysis ...................................................................... 85
  4.4.3 LDA-based Learning Agents ........................................................................ 89
  4.4.4 Market-Driven Brokering with Reputation ................................................ 91
4.5 Simulation Environment ................................................................................... 95
4.6 Performance Evaluation .................................................................................. 100
  4.6.1 Two Domain, Single Broker Scenario ....................................................... 101
  4.6.2 Multi-Domain Multi-Broker Scenario ....................................................... 103
  4.6.3 Broker Reputation Detection ...................................................................... 109
  4.6.4 Performance Impact of Broker Reliability ................................................ 112
  4.6.5 Market-Driven Brokering ............................................................................ 115
  4.6.6 FlowBroker Overhead Analysis ................................................................. 118
  4.6.7 Heterogeneous Market-Driven Brokering ................................................ 121
4.7 Conclusion .................................................................................................... 125

Chapter 5 Conclusion and Future Work .............................................................. 127

Bibliography ...................................................................................................... 130
Chapter 1  Introduction

1.1  Autonomics and Cognitive Network Systems

Today’s physical network deployments are varied and diverse, involving various physical
technologies, such as Optical Fiber, Ethernet and Wi-Fi. Network control and management
systems must therefore comprehend the complexity of these environments and aid the operators
in optimization of policies with respect to various concerns, such as Fault Tolerance,
Configuration, Accounting, Performance management, and Security monitoring (FCAPS) [3]. As
networked systems increase in complexity, it is clear that human actors are not able to monitor
and formulate optimal policy-based management decisions within these systems in real-time
with the speed of a distributed system of intelligent software agents [4-7]. As such, there clearly
exists a need for fully automated network management software systems, which are capable of
Observing network state, Analyzing the relationship between previous policy decisions and the
current network state, and Formulating and applying new policy decisions to further improve the
network state. Such a workflow, or closed-loop approach, is typically referred to as an
Autonomic (or Cognitive) System [8-11], which self-monitors, self-analyzes, and self-optimizes,
without human intervention.

The premise behind autonomic or cognitive networking is network state awareness [12].
Cognitive network element(s) follow an operational workflow, or feedback loop, which entails
observation, policy modification, enacting such policies, and further observing the effects of
these modifications [13]. In the case of a cognitive network layer, policies address fundamental
questions, such as, “To which outgoing link should node i forward the current data packet, and
what is the result of this forwarding decision upon network QoS?” High-level goals, such as
throughput maximization, are formulated into policy, which in turn dictates specific actions on the part of the network management entities, which then observe the effects of these actions in the form of quantifying network performance metrics, such as latency and packet loss rates. The uncertain, dynamic environment of a hybrid mesh network, where certain nodes are mobile and existing link quality is constantly in a state of flux, is an ideal motivation for the development of a cognitive network management architecture. In general, cognitive networks involve cross-layer adaptability based on network performance, and are ultimately driven by end-to-end QoS demands [14]. The components required for this closed-loop architectural approach should include:

1.) a network measurement system for observing the various dimensions of current network state
2.) a knowledge discovery system for refining network state into system knowledge
3.) a network management system for translating high-level goals to specific management policies and orchestrating the coordination of knowledge to various distributed management agents
4.) a knowledge propagation mechanism to facilitate the efficient transfer of knowledge to various network distributed management agents
5.) and network elements which are capable of enacting the management decisions (actuations) driven by the agent-based reconciliation between policy and knowledge discovery.
Using these components, the *observing, knowledge forming, reasoning, knowledge propagating*, and *actuation* phases will be simplified into an “observe-analyze-act” loop [15] in the networking system, as illustrated in Figure 1.

![Autonomic Network Management Workflow](image)

**Figure 1 - Autonomic Network Management Workflow**

In this dissertation, we study cognitive agent-based optimization of network processes in three scenarios: a) choosing ideal cluster-head nodes in purely multi-hop wireless network deployment based upon stability metrics, b) dynamically tuning the update interval in the Ad-Hoc On-Demand Distance Vector (AODV) routing protocol to lessen the overhead and increase overall packet delivery rates, and c) introducing multi-domain intelligent resource brokering to software-defined networks to improve network-wide load balancing and traffic delivery. In all three cases we leverage machine-learning techniques to improve the decision-making abilities of
the agents embedded within the network elements themselves. And in all three cases we illustrate the ability for machine-learning enhanced network agents to observe network state, leverage machine learning to extract understanding from those state observations, and determine configuration decisions in an optimized and autonomous fashion.
1.2 Organization

The rest of this dissertation is organized as follows. Chapter 2 studies how Wireless Mesh Networks allow for dynamic reconfiguration in the face of unstable topologies, as well as a decreased reliance on traditional infrastructure models based-upon more static networking architectures. This work studies how leveraging observed nodal characteristics in this chaotic environment can be used for optimal clustering structures, which allows for more reliable transport and management, despite the fluctuating nature of network connectivity. We show that by using machine learning techniques to observe nodal stability, the average time a particular node is associated with a given cluster is lengthened, over more traditional approaches such as standard weighted clustering. This result leads to a decrease in the network overhead needed to maintain network structure, as well as a better understanding of a given network’s behavior.

Chapter 3 discusses how the addition of reinforcement learning-based network management agents can improve Quality of Service (QoS) by reconfiguring the network layer protocol parameters in response to observed network performance conditions. This chapter presents a closed-loop approach to tuning the layer three protocol in multi-hop wireless networks based upon current and previous network state observations, specifically the HELLO Interval and Active Route Timeout parameters of the AODV routing protocol (AODV-Q). Simulation results demonstrate that the self-configuration method proposed here demonstrably improves the performance of the original AODV protocol, reducing protocol overhead by 43% and end-to-end delay 29% while increasing the packet delivery ratio by up to 11% depending upon the traffic model.
Chapter 4 develops a modified distributed network control architecture for Software Defined Networking (SDN). As the reference implementation for the SDN architecture is reliant upon a single controller to push flow rules to all SDN-enabled switches in the network, this creates a performance bottleneck and single-point of failure in large networks. To provide a scalable yet efficient solution to distributed SDN network management, we propose FlowBroker, which is a hierarchical brokering agent layer to manage and coordinate among distributed SDN controllers, where each controller is charged with the flow-rule maintenance of the switches in its own managed domain. Moreover, we introduce distributed machine learning agents to allow controllers to evaluate which brokers are more advantageous than others, from a performance-based reputation perspective. Simulation results show that the FlowBroker architecture, with broker-based collaborative load balancing and controller-based distributed reputation, can significantly increase the network performance of a multi-domain software-defined network. FlowBroker yields significant reductions in traffic loss, end-to-end delay and maximum link utilization when cooperative brokering and reputation are utilized. Finally, we introduce market-based competition among brokers for controller customers and study the effects upon overall network performance.

Chapter 5 concludes this dissertation.
Chapter 2  Case Study: Stability-Based Clustering in Multi-hop Wireless Networks

2.1  Introduction

A key challenge in leveraging the full potential of Multi-Hop Wireless Networking technology is the heterogeneous behavior of the nodes that make up the network, and how to manage the network in light of their typically unstable and unpredictable behaviors. Furthermore, the ability to localize certain behavioral patterns as well as aggregate similar nodes to more accurately identify emergent regional or network-wide behavior would give the management mechanism a more powerful means by which to formulate efficient policies. Additionally, it is important that such a management scheme scales with the number of nodes in the network.

A key facilitator of scalability in large mesh network deployments is clustering [16-18]. Clustering is needed to leverage temporal and spatial relationships which arise from nodal dynamics, such as mobility and transmission range. These dynamics dictate the state of the network in terms of topology and overall reachability, which in turn directly impact the performance of transport and management tasks taking place on the network. By more accurately characterizing network dynamics at node-level granularity, as well as higher-level emergent network behavior, and being cognizant of past network state, we are better able to predict more optimal clustering associations, by choosing better clusterheads and reducing the overall number of clusters, as well as the overhead associated with maintaining these nodal aggregations.
In the ensuing section, we discuss the topic of network state analysis. The state of the network can be viewed in several fashions, depending upon the metric of interest. For example, network state can be thought of as the reachability of all nodes in the network relative to each other, or the speed with which each node is moving, or a snapshot of the current topology. An autonomic network management system must be able to reconcile several perspectives and create an emergent, cohesive view of what is actually happening in this environment, so as to allow for effective policy management, whether it be load balancing, fault handling, or any other management task [19].

We discuss how virtual topological clustering can be used to scale network management decisions and allow for state information to be aggregated, such that the most ‘stable’ nodes end up being the ones to formulate management policy. Such nodes are indeed the ones promoted to the top of the hierarchy and are therefore responsible for high-level management decisions. Clustering allows for management decisions to be scaled, observational state to be aggregated temporally and spatially, and optimal policy to be fed back down to low-level actuating nodes that make the actual decisions in the network. We begin by discussing varying approaches to clustering in multi-hop wireless networks, followed by a description of reinforcement learning [20] and its applicability to optimal clustering. Finally, we conclude with a discussion of how virtual hierarchical clustering aids in autonomic management of highly dynamic and mobile network environments.
2.2 Existing Clustering Techniques

In nodal clustering schemes for multi-hop wireless networks there are typically three types of nodes; clusterheads, cluster gateways, and ordinary cluster member nodes. Clusterheads are charged with the maintenance of their given cluster, ordinary nodes are adjacent to clusterheads, and as such are members of a clusterhead’s cluster. Cluster gateways are nodes which are members of two or more clusters. Generally, clustering techniques typically fall into one of six categories [21]:

- Dominating Set-based Clustering,
- Reduced Maintenance Clustering,
- Mobility-Aware Clustering,
- Energy-Efficient Clustering,
- Load-Balancing-based Clustering, and
- Combined Metrics-based Clustering.

The following discussion will provide a brief overview of these clustering techniques with respect to the key metrics of interest and how they are used to form clusters in multi-hop wireless networks.

The Dominating Set-based approach to clusterhead election is based upon the need to identify a minimal set of key nodes upon which to optimize route formation and maintenance operations, thereby effectively forming a virtual backbone across the multi-hop network [22].
Furthermore, the calculation of this minimal node set allows for less frequent routing updates, so long as the dominating set chosen is also fairly stable.

Reduced Maintenance Clustering recognizes that since the cost of cluster reformation is far higher than mere re-affiliation, determining clusterheads based upon decreasing the frequency and cost of reformation is a promising approach to cluster formation. Specifically, the Passive Clustering algorithm [23] is a clustering technique which does not require the generation of explicit signaling packets for state awareness and cluster formation, but rather clusters are constructed on the fly, in response to user-driven data traffic. By piggybacking nodal status on the data packets that a source node sends, it indicates its readiness to serve as a clusterhead. Thereafter, the neighbor nodes decide whether to stay members of their current clusters, or join with the newly announced clusterhead to form a new cluster. Moreover, for the process of selecting gateway nodes (intermediate nodes which belong to two or more clusters), a minimal set of such nodes is chosen to reduce the need for reformation due to topological changes.

Mobility-based clustering highlights the fact that mobility is the main factor which causes topological flux within a network. By taking mobility metrics into account, such as speed and direction, more reliable clustering can take place with fewer instances of reformation and re-affiliation. A large body of work concerning mobility-based clustering focuses on mobility prediction as a temporal means of efficient nodal aggregation [24, 25]. However, this cannot always hold true since current mobility does not necessarily translate into future mobility patterns of a given node.

The motivation behind energy-consumption aware clustering is the fact that a clusterhead tends to expend more energy than standard nodes due to the increased frequency of data
transmission inherent in its role. By monitoring the amount of energy consumption which takes place at each individual node, better clusterhead elections can take place to increase the average residual power available on a node-to-node basis and across the network. This in turn will increase the lifetime of a network which may be severely limited by the lack of available energy, especially in the case of mobile nodes powered by limited battery capacity. Several existing schemes attempt to address this performance concern by limiting the duration a node can spend as a clusterhead, based upon the energy it has consumed thus far by minimizing the size of the dominating set upon which the clustering scheme is determined [26].

Load-Balancing oriented clustering is typically concerned with limiting the number nodes in a cluster to minimize the traffic load on clusterheads [21]. When clusters are too large, the clusterhead can impede transport performance of the network itself. Conversely, when the cluster size is too small, the number of clusters network-wide increases, along with the overhead of maintaining the clustering structure.

Combined metrics clustering is typified by the Distributed Clustering Algorithm (DCA) [26], in which a weighted linear combination of a subset of the above mentioned clustering metrics is used determine the overall suitability of a node to act as a clusterhead. The various parameters are normalized and weighted according to performance,

\[ s_i = w_1c_1 + w_2c_2 + ... + w_nc_n \]

where \( i \) is the node ID, and \( n \) is the number of metrics under consideration for the nodal fitness calculation.
However useful it may be to incorporate one or more of these metrics into the general clusterhead-determining calculation, it remains that the afore-mentioned techniques rely on static snapshots to perform spatial aggregation, or mere mobility prediction to address temporal concerns [27]. There is clearly a need to tune these calculations with the temporal state of nodes so as to better anticipate which candidates will in fact make the most ideal clusterheads in terms of long-term stability. We posit that reinforcement learning is a promising technique to achieve this goal.
2.3 Reinforcement Clustering

While clustering is typically a technique used to scale routing in a large multi-hop wireless network, our goal is to leverage this technique for the management of dynamic networks. By creating multi-level hierarchy and clustering based upon stability metrics, distributed network management is facilitated and varying views of emergent network behavior are apparent at varying levels of granularity, depending upon how the network state is observed and aggregated. Our particular clustering algorithm uses a network-distributed form of Reinforcement Learning to take note of a node’s previous state and factor that knowledge into the current clustering determination.

2.3.1 Reinforcement Learning

Reinforcement Learning is a form of Machine Learning [28], characterized by the formulation of policy to achieve a particular set of goals. Reinforcement Learning problems are typically modeled by means of Markov Decision Processes (MDPs) [29]. The model is comprised of the set of potential environment states, S; the set of possible actions, A; the designated reward function, R: S × A → R’; and the policy which determines state transition, P: S × A → π(S). The set π(S) represents the set of functions over set S, which specify the actions required to transition by means of action a ∈ A from state s to state s’. The instantaneous reward (S, A) is a result of taking action a while in a particular state [30].
The specific form of Reinforcement Learning that our clustering algorithm employs is referred to as $Q$-Learning [30]. As a model-free reinforcement learning technique, $Q$-Learning is ideally suited for optimization in dynamic network environments. Model-free techniques do not require any explicit representation of the environment in which policy updates are formulated, and as such, can solely address parametric optimization to maximize long-term reward. This is a particularly important benefit of $Q$-Learning, since the problem addressed, namely the management of dynamic wireless mesh networks, is multi-dimensional problem with several state variables involved, which leads to a large state space.

$Q$-Learning employs a weighted moving average calculation, to not only take note of recent policy success/failure as feedback, but to also take into account the weighted average of past values observed, referred to as ‘$Q$-values.’ $Q$-value computation is performed via the following equation
\[ Q(s, a) = (1 - \alpha)Q(s, a) + \alpha \cdot Q(s', a') \]

\(Q(s, a)\) refers to the \(Q\)-value computed based upon state ‘s’ and action ‘a.’ In the current network environment considered, state is the observed behavior of nodes within the network, whether considering mobility, connectivity, nodal distance, or a combination of parameters. Actions in this setting correspond to the decision of whether or not to employ certain nodes as clusterheads and include the one-hop neighbors of such nodes in the newly formed cluster. Variable ‘\(r\)’ represents the instantaneous reward value, derived from a measurement of the current environment based upon the current policy. The variable ‘\(\alpha\)’ is referred to as the learning rate, or the weight assigned to the current observation, between 0 and 1. In our implementation the learning rate also determines the weight coefficient of the previously observed \(Q\)-values, namely the ‘\((1-\alpha)\).’ \(Q(s', a')\) is the new \(Q\)-value computed, which corresponds to the current action \(a'\) and the newly resultant state, \(s'\). By measuring \(Q\)-values over time, the relative fitness of a particular policy, or in our case, a choice in determining cluster formation, may be ascertained, and obligatory iterative modifications to this policy can be performed [31].

### 2.3.2 Reinforcement Learning-based Clustering

Clustering techniques, which incorporate previous network state into the decision of node fitness for clusterhead election, mainly focus on mobility prediction [27, 32] as a means of calculating nodal fitness. However, our RL-based clustering scheme leverages the weighted parameterization of the Weighted Clustering Algorithm (WCA) [33] algorithm, by combining node metrics which are relevant to measuring the stability of the node, and uses Reinforcement Learning as a means optimize this calculation by taking into account past stability values. We
dynamically tune the learning rate based upon relative mobility, by normalizing the speed of the node based on the max speed of all other nodes for a given iteration. This in turn becomes the current learning rate for this particular node, thereby facilitating a distributed learning mechanism which is both specific to that given node, but also takes into account network-wide parameters of mobility for the normalization process. The goal is to allow the clustering management mechanism to leverage the emergent stability which exists within the network at any given time. Further, by taking into account recent nodal behavior through the use of a weighted moving average, we are able to temper short term reaction to network dynamics with a long term perspective of overall optimization and long-term reward.

The adaptive learning rate updating is a function of the relative stability which may exist in the network. Stability can mean several things in this sense; degree of connectivity, residual power levels, relative mobility, etc. Since we are primarily interested in the physical stability of the network to facilitate adaptive nodal aggregation through clustering, we chose to use three metrics for the stability measure, and weight them according to their perceived importance; namely:

1.) $c_i$: the difference between ideal cluster member count and number of one-hop neighbors for each candidate node,

2.) $d_i$: the average distance of a candidate clusterhead with all of its one-hop neighbors

3.) $s_i$: the speed of each node

These metrics are then combined to form the following node fitness equation,

$$f_i = w_1 c_i + w_2 d_i + w_3 s_i$$
where \( f_i \) is the fitness index for node \( i \), and weights \( w_1, w_2, \) and \( w_3 \) are determined before-hand (for simulative purposes, we set each equal to 1/3 to reflect the uncertainty of the relative importance of each measurement).

Our algorithm then takes a weighted moving average of these fitness values on a per node basis, relative to the state of the current node with respect to its mobility. In particular, the learning rate ‘\( \alpha \)’ of this calculation is determined by how fast the node is currently moving, divided by the fastest node speed in the network.

\[
f_i = (1 - \alpha) f_i + \alpha \cdot f'_i
\]
By leveraging the above behavioral metrics in a weighted fashion for periodic clusterhead election, as shown in weighted equation above, we are able to take a more comprehensive view of network stability. Further, we optimize our reinforcement calculation by taking into account recent stability indices for each node, weighted by the instantaneous mobility of the node itself. The fitness value, becomes, in effect, the penalty values for the Q-Learning calculation, in that the lower the fitness value, the higher the chance a node will become a clusterhead. Clusters are
then formed by comparing the relative fitness values, and choosing the lowest value as a clusterhead, as well as its one hop neighbors to form a new cluster (see Figure 3 for illustrative algorithmic workflow). The clustering algorithm executes as follows:

1. for each iteration of the algorithm, if a node is already chosen as a clusterhead, or covered by a clusterhead, then it is no longer considered to be elected a clusterhead,

2. otherwise, each node’s fitness metric is updated relative to the moving average of its past stability metrics,

3. from the remaining candidate clusterheads, the node with the lowest fitness value is chosen as a clusterhead, and its one-hop neighbors as cluster members, and

4. this algorithm persists until all nodes are either clusterheads or are covered by a clusterhead, at which time the election process ends its current iteration.

By taking past node behavior into account, we prevent the clustering mechanism from making hasty decisions about the fitness of a node based upon instantaneous fitness measurements, thereby decreasing the overhead needed to maintain our clustering scheme.
2.4 Methodology and Results

We use the stability of the clustering structure itself to determine the overall performance of the clustering scheme, with respect to such metrics as number of clusterhead changes and total number of clusters, as a function of maximum nodal speed, nodal density, and transmission range. To simulate the environment, we use a Matlab model, which employs random waypoint mobility for changing node mobility vectors. The speed and direction are uniformly chosen with a max pause time of 5 seconds and max speeds of 5, 10, 20, 30 and 50 m/s for each simulation. The simulation duration is set at 200 seconds to give sufficient time for topological flux to take place due to mobility. The physical area of the network is a typical 1000m × 1000m, with node counts of 50 vs. 100 nodes. Node counts are varied for the same geographic area to observe the effect of nodal density on clustering stability and decision-making. Further, transmission range changes from 100m to 200m for the first and second set of simulations respectively. This too is done to observe the effects of cluster expansion upon the stability of the cluster formation and maintenance phases.

2.4.1 Clusterhead Change

The results obtained with respect to cluster change are promising, as shown in Figure 4. Notice a decrease in the number of cluster reformations of approximately 15-20% for Reinforcement Clustering over the WCA clustering implementation. This result proves intuitive, in that, our algorithm is better able to adjust the rate at which network stability is taken into account when determining nodal fitness values. Rather than simply calculating the connected dominating set at a particular time instant, Reinforcement Clustering weights the current state
observation of nodal stability by the current mobility of that node, tuning the learning rate accordingly.

By using this relative stability calculation, we effectively reduce the number of cluster reformations which take place. This in turn leads to more efficient management of the network, by decreasing the need for explicit cluster management messages, decreasing nodal energy consumption, and increasing the accuracy of network state measurement. The distributed network management agents may then take advantage of this more accurate state representation to make more optimal policy decisions, in terms of routing, resource provisioning, and topology management.

Figure 4 - Degree of Clusterhead Change per Clustering Technique
(50 Nodes)
2.4.2 Cluster Count

Though there is a fairly consistent reduction in the number of clusters required when using Reinforcement Clustering over simple WCA, as shown in Figure 6, the decrease is too minimal to be considered significant. We attribute this result to the fact that both algorithms use similar methods of cluster election. Furthermore, while Reinforcement Clustering takes past behavior into account, it does not necessarily work towards lowering the overall cluster count, but rather creating the most stable cluster configuration, as evidenced by the decrease in instances of cluster reformation, discussed in the previous section.

Figure 5 - Degree of Clusterhead Change per Clustering Technique
(100 Nodes)
Figure 6 - Average Cluster Count per Clustering Technique (50 Nodes)

![Figure 6](image)

Figure 7 - Average Cluster Count per Clustering Technique (100 Nodes)

![Figure 7](image)
2.4.3 Implications of Transmission Range and Nodal Density

By doubling the number of nodes for a given geographic area, we effectively increase the nodal density of the network. We witness a more consistent decrease in the number of cluster reformations over the lower density 50 node count. The increased transmission range will usually improve cluster stability due to the increased spatial locality. Moreover, increased nodal density appears to also have a beneficial effect on clustering performance and stability. Regardless of which metric is considered, our Reinforcement Clustering technique yields a decrease of cluster reformation, but the amount of improvement appears to decrease with the increase in node density and decrease in node movement [34].
2.5 Conclusion

By taking advantage of emergent stability trends within a given deployment of nodes, we display that when using reinforcement learning, more effective temporal and spatial aggregation of nodes into virtual clusters can take place. Further, by dynamically tuning the learning rate of the RL clustering algorithm, with respect to the stability of an individual node relative to its peers, we are better able to assess the fitness of that node, in terms of whether or not it should carry out the duties of a clusterhead.

While many challenges remain insofar as reinforcement-learning-based clustering is concerned, our preliminary results yield promising indications that policy-based reinforcement learning can indeed be used in mesh environments with varying degrees of dynamicity. Future work will look at the tuning of weighted stability metrics for fitness value computation, depending upon the state characterization of the network at any given time. Additionally, using hierarchical management for varying levels of policy formation will be an important step to introduce scaling in large deployments. This in-turn will allow for a more efficient means to manage such dynamic networks through building a management hierarchy, and decreasing the overhead necessary for maintaining this management structure.
Chapter 3  Self-Adapting Protocol Tuning for Heterogeneous Networks Using Q-Learning

3.1 Introduction

While the strict layering architecture of the Open Systems Interconnection (OSI) stack is conceptually useful, it is not as effective for wireless networks when time-varying traffic is served over a channel with limited throughput. Efficiently utilizing the resources with QoS provisioning requires a cross-layer optimization approach. As a result, better performance can be expected from information exchange across the protocol layers [35, 36]. The purpose of this chapter is to address these issues by exploring the concept of intelligent network management for globally optimum performance in a dynamic wireless network deployment.

In typical network deployment scenarios, networks elements are limited in their abilities to adapt to changing application demands and topology characteristics, lacking the ability to take the context of these changes into account. In the case of routing in multi-hop wireless networks, battery-powered devices create challenging problems in terms of prolonging the autonomous lifetime of the network. In designing intelligent routing protocols, the various features of sensor networks lead to a set of optimization problems in routing path length, load balancing, consistent link management, and aggregation [37]. In real scenarios however, these factors are usually in conflict with one another, and influence the routing performance in a complex way. This in turn, leads to the need for a more sophisticated routing scheme that makes ideal trade-offs between
multiple factors. Clearly, solving the optimization goals separately does not lead to a globally optimal solution; rather, all metrics should be addressed with respect to one another.

A solution for addressing these multi-variant optimization problems in network management lies in the vision of cognitive networks [38]. The hallmark of Cognitive Networks is their ability to continuously adapt to changing environmental conditions and/or user needs by iteratively optimizing the bandwidth access and communication links. Typically, machine learning techniques, such as Q-learning [39] help implement the adaptation methods of self-configuration and self-management in the autonomic computing paradigm. Recent work further reinforces the efficacy of leveraging machine learning for network management task optimization [40-43].

The self-configuration of network systems has cross-layer ramifications for the protocol stack, from the physical (PHY), Medium Access Control (MAC), network, and transport layers to the application layer. Therefore, cross-layer design [44-47] approaches are critical for the efficient utilization of limited resources, to enable QoS guarantees, in future wireless and heterogeneous networks. In this chapter, we present the concept of self-configuration in a cross-layer context, which can overcome the current limitations of network management in heterogeneous wireless networks, by allowing networks to observe, analyze and act [48] in order to optimize performance. Our approach is to augment the routing strategy of the AODV routing protocol with Q-learning, to ensure that the packet delivery ratio can be increased, while at the same time minimizing management overhead.

Toward the above stated goals, we present a new architecture [49, 50] of reconfigurable ad hoc routing management with Q-learning, namely, Q-Learning based Self-configuration.
QLS management and the AODV-Q protocol. The QLS management architecture enables nodes to efficiently learn optimal routing strategies, thereby enhancing the packet delivery ratio, end-to-end delay, and other QoS performance metrics. We present NS-2 [51] simulation results showing that our cross-layer, self-configuration approach successfully improves the scalability of the AODV routing protocol in a heterogeneous network environment. The remainder of the chapter is organized as follows: Section 3.2 presents a belief survey of related work. Section 3.3 gives an overview of our network architecture with reinforcement learning techniques for autonomic self-management. Section 3.4 describes in detail AODV-Q. Section 3.5 explains the NS-2 simulation scenario. Section 3.6 presents NS-2 simulation results. Finally, Section 3.7 concludes by projecting future research directions.
3.2 Related Work

3.2.1 Cross-Layer Approaches for Intelligent Network Management in Wireless Networks

The realm of network management covers a vast collection of issues, such as IP configuration, security and network monitoring [52]. While these concerns are not unique to Mobile Ad hoc Networks (MANETs), they do become more difficult to optimize when nodal mobility, dynamic network membership, and unstable links are introduced into the network [53]. Depending on the speed of the Mobile Nodes (MNs), mobility can be classified into three categories: static, low mobility, and high mobility. The management layer of such a network should be able to take into account any of these three cases or combination thereof. In the case of low mobility, the steady-state performance should be optimized since incidental updates (e.g., for route discovery) can consume excessive bandwidth resources. For high mobility networks, resource consumption, and delay due to route maintenance are important limiting factors [54].

Centralized network management architectures fail to provide effective scalability in MANETs. In the last few years, a distributed decision making scheme [55] has been introduced to address these concerns. In this proposed scheme, nodes may only be aware of their own neighbors and have no understanding of the size and extent of the network. Finding a mechanism that can deal with particular challenges associated with distributed decision-making in ad hoc networks is certainly non-trivial. To cope with these demands, management solutions based on cross-layer
design [44-47] are necessary for efficient utilization of the limited resources in future wireless networks.

### 3.2.2 Challenges in Multi-Hop Wireless Networks

Several papers have classified MANET routing protocols in terms of their behavioral characteristics and applicability. In this chapter, we largely adhere to the standard convention of classification, namely *flat*, *hybrid*, and *geographically-oriented* protocols. Routing protocols which are not organized in any hierarchical fashion are commonly referred to as flat routing protocols [60]. Flat routing schemes have three main classifications: *proactive* (table-driven, e.g. Optimized Link State Routing Protocol (OLSR) [61]), *reactive* (demand-driven, e.g. Dynamic Source Routing protocol (DSR) [62], Ad-Hoc On-Demand Distance Vector protocol (AODV) [63]), and *hybrid* (e.g. Zone Routing Protocol (ZRP) [64]).

Dynamic Source Routing protocol (DSR) is a reactive protocol which uses source routing as a central mechanism [58]. When a route request (RREQ) is made by a particular node, it uses the destination route stored in its local route cache to send the data packet. Nodes along the path aggressively cache the path from the source node’s cache (which is embedded in the packet itself). However, if the node does not have the required route information cached, the route discovery process is initiated by flooding the network with route request packets. The request packets propagate throughout the network until they reach the destination node, or a node which has a cached path to the destination. The end node then sends a route reply with the newly discovered route source information back to the source node, which then caches the path for
future source routing. Additionally, destination nodes respond to all route request packets, thereby increasing the amount of aggressive caching taking place throughout the network.

The Ad-Hoc On-Demand Distance Vector (AODV) [63] routing protocol is another routing protocol for multi-hop wireless networks, similar in nature to DSR. AODV shares DSR’s on-demand characteristics in that it also discovers routes on an as-needed basis via a similar route discovery process. However, AODV adopts a very different mechanism to maintain routing information. There is only one table entry per destination in any particular node’s routing table. AODV uses sequence numbers to determine the “freshness” of routes in the various routing tables. Without source routing, AODV relies on routing table entries to propagate the route reply (RREP) back to the source and, subsequently, to route data packets to the destination. An important feature of AODV is the maintenance of timer-based states in each node (e.g. active route timeout, hello interval, etc) with parameters regarding the utilization of individual routing table entries. A routing table entry is expired when not used recently. A set of predecessor nodes is maintained for each routing table entry, indicating the set of neighboring nodes which use that entry to route data packets. These nodes are notified by route error (RERR) packets when the next-hop link breaks. Each predecessor node, in turn, forwards the RERR to its own set of predecessors, thus effectively erasing all routes containing the broken link.
Figure 8 - Overall cognitive network architecture for distributed optimization in heterogeneous networks.
However effective AODV may be [65], it suffers from the following drawbacks in a mobile network environment:

(a) it does not frequently update the route to the destination,

(b) due to the large Hello Timer values, there appears to be a periodicity in the route request generation which, in turn, can be attributed to poor link failure detection, and

(c) it determines the ‘best effort’ shortest path, i.e. the shortest successful path.

In the case of proactive protocols, such as OLSR, there are sufficient exchanges of routing information to result in near-optimal routes. Therefore, OLSR is more resistant to packet drops at the MAC layer. However, one of the drawbacks of OLSR is that it generates routing traffic independent of application traffic [66]. Due to the higher degree of routing overhead in proactive routing protocols, we chose the reactive routing protocol, AODV, in our cross-layer approach and focus on enhancing the protocol performance with a self-configuration mechanism.

To identify the trade-off issues when using reinforcement learning, it is crucial to study the impact factors of routing protocols, traffic load and mobility, and their impact on service delivery. A statistical design of experiments could be beneficial to identify both main effects and interactions of factors that best explain the response variables [67]. However in this chapter the focus of our investigation is on reconfiguring the critical timers, namely, hello interval and active route timeout (ART), to enhance network performance by dynamic context exchanges in heterogeneous networks.
3.2.3 Existing Protocol Parameter Tuning Solutions

Parametric tuning of routing protocols, and AODV in particular, has been of increasing interest in recent years [49, 68-73]. In [73], Vadde and Syrotiuk explore the sensitivity of AODV protocol parameter tuning in conjunction with network performance metrics. The authors illustrate that nodal mobility is the major contributing factor to delay, due to frequent route re-establishing processes. Additionally, they explore the fact that the packet arrival rate is the main contributing factor to affecting throughput, and that the timer interactions of network events and timers, such as the ACTIVE_ROUTE_TIMEOUT, directly affect the generation of performance-degrading protocol overhead packets.

Other works have proposed solutions on how to concretely modify these protocol parameters to improve network performance. In [70], Xing et. al. propose a modification to AODV, DA-AODV (Dynamically Adjusting AODV), which measures network scale to limit the scope of network max hop count. Leveraging the RREQ and RREP packets to carry this information, the max hop count indicates the number of max hops between source and destination pairs. Network max hop count is calculated on a per-node basis, indicating the max hops on a path for a particular source/destination pair. The authors add a new routing table parameter, Net_Diameter, to denote the max hop count value for each node’s routing table entry. When the routing table entry changes, Net_Diameter is compared against every table entry to ensure that it is set equal to the max hop value. When a node wishes to send a RREQ message to a particular destination, the source node first compares its max hop count with the Net_Diameter value, setting either of the two values to the greater of the two values, thereby allowing the
HELLO packet to be broadcasted over the whole known network. By increasing the range of network discovery, the authors show a reasonable reduction in end-to-end delay and route error packets, due to the enhanced routed discovery mechanisms.

In [69], Li and Han propose a multi-hop wireless protocol tuning approach which uses nodal mobility characteristics to determine changes in settings in AODV. Specifically, the authors chose to tune the SEND_HELLO_INTERVAL based upon feedback of reply and acknowledge packets. The algorithm used is as follows:

Procedure Recv(P)

begin

p_type = P.type();

if p_type == REP or ACK

intr = Calculate_LAST_REP_ACK(P)

if intr decrease

SEND_HELLO_INTERVAL += ∆t

end;

end;

The intr value represents the time elapsed between the current RECV or ACK packet being analyzed and the last time a RECV or ACK packet was analyzed. The ∆t value is added to the SEND_HELLO_INTERVAL based upon whether or not the intr value decreased, which denotes an increase in frequency of REP or ACK packets. When combining this modification with the calculation of average neighbor’s speed (sending more RREQ packets if neighbor speed
increases), the authors were able to increase the lifetime of the network significantly over when using standard AODV routing.

In [71], Tan and Seah propose a solution whereby nodal mobility is used to tune the frequency of HELLO messages. Before a HELLO message packet is transmitted, the nodal mobility is inferred by comparing the current neighbor table of the node to the previous neighbor table of the node when the last HELLO transmission occurred. If the change count (new neighbors + neighbors left count) is less than one, a so-called Deviation_Fraction value is set to 1.25, if its greater than 5, Deviation_Fraction is set to 0.75, and if the change count is in between 1 and 5 exclusive, the Deviation Fraction is set to 1. The HELLO interval is then set equal to HELLO_INTERVAL*Deviation_Fraction. This process continues for each interval expiration. The proposed solution results in a 20% reduction in Hello packet overhead, and increase in packet delivery ratio due to longer lasting stable routes.

In [49, 68] we propose a general framework for autonomic network management in heterogeneous network environments. Specifically, in [68] we proposed using Q-learning to load-balance OSPF traffic to avoid link congestion, which was achieved by having network agents observe and track queue length of nodes in various routes. The reinforcement learning agent would then compute and track these queue lengths over time to determine the optimal routes to facilitate network-wide load balancing, and resulting in dramatic decreases in packet loss. In [49] we provided initial experimental results of leveraging reinforcement learning to improve AODV routing protocol performance over standard AODV by tuning the HELLO Interval. This work was a cursory study of how sensitive AODV would be to parameter modification in a heterogeneous environment. However, this work did not compare these modifications of AODV to any existing AODV parametric tuning solutions, nor was the learning
rate adaptation explored. Moreover, the approach we use in this chapter for measuring application performance is more tightly coupled to the actual realities of the network behavior. By using the max observed end-to-end delay instead of a predefined max allowable end-to-end delay, as part of the feedback mechanism to determine which protocol parameters to tune, we now observe better results.

For the purposes of this work, we chose to compare the performance of AODV-Q to the modified protocols Modified-AODV (Mod-AODV) [71] and Optimized-AODV (Opt-AODV) [35], as well as standard AODV. Since both solutions dynamically update the HELLO interval, the modifications proposed in [71] and [69] are more closely related than other existing proposed protocol enhancements. In the next section, we cover the details regarding our proposed protocol tuning enhancement, aided by autonomic/cognitive management principles, leveraging Machine Learning for better optimization and performance. After presenting comparative performance results, we then give the results of an initial investigation into dynamic learning rate tuning, changing the learning rate in response to the degree of node mobility.
3.3 Self-Configuration for AODV

3.3.1 Self-Configuration Parameters for AODV

In AODV, the hello interval and ART values are important parameters to cope with link failures caused by network dynamics. However, these timers are typically set in a trial-and-error manner or as a constant value, which can lead to great inefficiencies with respect to performance [74]. In our approach, we apply the Q-Learning technique, leveraging cross-layer performance, to identify any possible performance implications involving these timers.

AODV uses hello messages, periodic local broadcasts by a node to inform each mobile node in its neighborhood [75]. The hello messages may list other nodes from which a mobile has heard, thereby yielding a broader knowledge of network connectivity. Setting the optimal hello interval is a crucial aspect of maintaining network connectivity in AODV-based multi-hop wireless networks.

The route discovery process of AODV allows the intermediate nodes to store a route’s state between the endpoints [76]. Each node keeps this state for a length of time given by the ART parameter. Every time the route is used, the timer is reset back to the ART value. The ART is a static parameter that defines how long a route is kept in the routing table after the last transmission of a packet on this route. This parameter is arbitrarily set to 3 seconds by default. Comparatively speaking, DSR keeps a similar time-out parameter, denoted route cache timeout, but with a value set at 300 seconds. The use of static values for the ART parameter does not take into account either the actual lifetime of the path or the scale of the time correlation between two successive connections between the same end-points. Finding an optimal value requires a
balance between choosing a short ART that causes a new route discovery, even if a valid route is still available, and choosing a long ART, which risks sending packets on an invalid route. In the first case, the cost is the initiation of a new route discovery that could be avoided, and in the second case it is the loss of one or more packets and the initiation of a RERR process instead of a new route discovery phase.

3.3.2 Q-Learning in Multi-hop Wireless Routing

The varied features of wireless networks lead to many optimization problems with respect to achieving specific performance objectives. The idea of applying reinforcement learning to routing in networks was first introduced by Boyan and Littman [77]. They showed that the Q-learning [39] based routing can compete with the shortest path algorithms, without prior knowledge of the network topology. Q-learning has also been applied to routing in ad-hoc networks [37]. Collaborative Reinforcement Learning (CRL) is also introduced and evaluated in [43] as a self-organizing technique for building a MANET routing protocol. To the best of our knowledge, no existing routing scheme with reinforcement learning takes into consideration optimization goals (routing path length, load balancing, consistent link management, and aggregation) combined with a cross-layer approach.
3.3.3 Cross-Layer Autonomic Network Management Architecture

The cross-layer architecture for our proposed cognitive management framework is explained in Figure 9. One of the main advantages of cross-layer design is to make protocols aware of current network state in a localized but distributed fashion. By introducing network and application layer context to the network management agents, this improves the higher-level processes of the middleware, allowing our QLS mechanism to exploit broader knowledge of the network state, and improve overall system performance.
Other proposals for implementation of cross-layer information exchange have been put forth in the current literature. These proposals can be categorized in three main groups [47]: (a) direct communication between layers, (b) a shared database across the layers, and (c) completely
new abstractions. Specifically, we present and implement a cross-layer model which sets performance expectations relative to performance observed thus far, in support of application-layer performance optimization, calculates reward and penalty values in the middleware layer, and uses those values to inform protocol parameter tuning decisions at the network layer. Figure 9 is an illustration of the cross-layer design approach, which conveys how QLS in the middleware layer can interact with the other reconfigurable modules in the network layer. The following steps describe the detailed workflow of this management scheme.

Step 1: The management module at the middleware layer gathers application demands and determines the corresponding requirements (in this case, minimization of ETE delay).

Step 2: The $Q$-learning agent in the middleware layer receives the performance requirement in the form or reward and penalty formulas.

Step 3: At the network layer, the AODV protocol provides the $Q$-learning agent with the decision variables, including end-to-end delay, RERR and RREP.

Step 4: The $Q$-learning agent decides which action should be used to enhance performance.

Step 5: The $Q$-learning agent reconfigures the routing parameter(s) accordingly ($hello\ interval$ and ART).

Step 6: Loop back to step one to iteratively observe the effects of environmental actuation and reformulate decision parameter values for Q-Learning agent based upon new observations.
Table 1 summarizes the proposed autonomic management approach, with respect to qualitative analyses of the challenges faced in MANETs.

<table>
<thead>
<tr>
<th>Key Challenges</th>
<th>Previous Approaches</th>
<th>Our Approach Proposed Here</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route Management</td>
<td>- Routing protocols directly manage the path</td>
<td>- Reconfigure (Tuning) the routing protocols to discover more-optimal paths</td>
</tr>
<tr>
<td>Management Overhead</td>
<td>- Relatively fixed management overhead (e.g. proactive protocols (OLSR) have almost fixed overhead)</td>
<td>- Autonomously (Adaptively) change the control message frequency according to the change of application demands</td>
</tr>
<tr>
<td>QoS management</td>
<td>- Resource management is limited to network layer</td>
<td>- Cross-layer resource management from network layer to application layer</td>
</tr>
<tr>
<td></td>
<td>- Mostly static, flat QoS support</td>
<td>- Dynamic QoS support by reconfiguring with Q-learning agent</td>
</tr>
<tr>
<td>Overall Performance</td>
<td>- Not aware of application demands</td>
<td>- Support Performance by our cross-layer approach. When reconfiguring routing parameters, end-to-end delay is used to determine the reward/penalty values, as performance is more heavily dependent on end-to-end delay.</td>
</tr>
</tbody>
</table>

Table 1 - Cross-Layer Reconfiguration in Wireless Mesh Network Management
3.4 Q-Learning Based Self-Configuration (QLS) for AODV

3.4.1 Q-Learning

In Q-learning [39], each time an action $a$ is executed, an agent receives an immediate response from the environment, in the form of a reward or penalty. A reward value will indicate the degree to which the previous actuation positively changed the environment in the agent’s desired direction, whereas a penalty will indicate movement away from the desired goal as a result of the previous actuation. The agent then uses this reward/penalty and the expected long-term reward to update the $Q$-values, which in turn influences future action selection. Its simplest form, one-step $Q$-learning, is defined as:

$$Q(s,a) = (1 - \alpha) \cdot Q(s,a) + \alpha \cdot \max_{a'} Q(s',a')$$

where $\alpha$ is the learning rate ($0 < \alpha \leq 1$), which models the rate of the updating of $Q$-values. The variable $s$ represents the present state observation and $s'$ the new state which the algorithm will explore. The variable $a$ represents the action which led to state $s$ and $a'$ the action that leads to $s'$. The $Q$-value itself is a numerical value which represents the current state action pair. In this context, the state is the current performance of the network and the action is how to tune various protocol parameters. Finally, $Q(s,a)$ is the $Q$-value derived from the current state-action pair, and $\max_{a'} Q(s',a')$ is the max $q$-value (reward) that can be obtained from next state $s'$ over all possible actions $a'$. As a model-free Reinforcement Learning technique, Q-learning requires no knowledge about the underlying reward or transition mechanism; thus it is applicable to the problem of learning routing strategy in ad hoc networks, where explicit state-space mapping can become computationally cumbersome. Specifically, mapping out the possible permutations of
networks settings, nodal mobility, and traffic interactions would be potentially infeasible, and Q-Learning allows us to avoid this task by exploring the state space of local state-action pairs without globally mapping it.

### 3.4.2 Q-Learning Based Self-Configuration

In our implementation of AODV-Q, each node has two $Q$ values, $Q_{\text{penalty}}$ and $Q_{\text{reward}}$. $Q_{\text{penalty}}$ denotes the penalty $Q$ value for unstable network status, which makes the node take the action of decreasing $ART$ and $Hello$ interval. $Q_{\text{reward}}$ represents the stability reward of the network, which will make the node take the action of increasing $ART$ and $Hello$ interval. With respect to the Q learning calculation,

$$Q_{\text{penalty}} = (1 - \alpha) \ Q(s,a)_{\text{penalty}} + \alpha \ Q(s',a')_{\text{penalty}}$$

and

$$Q_{\text{reward}} = (1 - \alpha) \ Q(s,a)_{\text{reward}} + \alpha \ Q(s',a')_{\text{reward}}.$$  

In AODV-Q, each node makes its self-configuration decision based on the local routing information, represented as the two $Q$ values which estimate the quality of the alternative actions. These values are updated each time the node receives a RREP packet. The reward value $r$ is given by:

$$r = Q(s',a')_{\text{reward}} = n \left( \frac{1}{ETE_i / ETE_{\text{max}}} \right) = n \left( ETE_{\text{max}} / ETE_i \right),$$

when a ROUTE REPLY packet reaches the source and there is a path from the source to destination. Further, the penalty value is calculated as:
Penalty = \( Q(s',a')_{penalty} = n \left( \frac{ETE_t}{ETE_{max}} \right) \)

when a ROUTE ERROR packet is generated due to a broken route, or a ROUTE REPLY packet reaches the source and there is no path from the source to destination. In the above equations, \( ETE_t \) is the current end-to-end delay; \( ETE_{max} \) is the maximum end-to-end delay observed thus far. The value \( n \) denotes the normalization constant. The end-to-end delay is used to affect the reward and penalty because network protocol performance has been observed to be more tightly coupled with the ETE delay metric than other network performance metrics [73]. The algorithm for calculating the penalty/reward values and updating the ART and Hello interval is as follows:

1. For a period of time ("learning_phase_duration" = 30 seconds), increase or decrease the ART and Hello intervals by 2 with equal probability.

2. When receiving a successful ROUTE REPLY message, calculate the reward value, update \( Q_{rewards} \), and GOTO step 4.

3. When receiving an unsuccessful ROUTE REPLY message or ROUTE ERROR message, calculate the penalty value, update \( Q_{penalty} \), and GOTO step 4.

4. If \( Q_{reward} > Q_{penalty} \), Decrease the ART and Hello Interval by 1, ELSE Increase each value by 1.

To avoid overly-aggressive changes in the ART or Hello Interval values, we converged upon a learning rate of 0.1. Future research will entail an investigation of the effects of dynamic learning rate tuning in response to observed performance thresholds. Section VII conveys initial results of experimentation with respect to an adaptive learning rate scheme.
If MN A receives a RRER then node A is more likely to decrease ART based on its Q-learning agent. MN B is going to decrease the Hello interval because MN B receives RREP indicating no valid route exists. But MN C may increase its Hello interval when MN C receives RREP of successful route discovery. By virtue of distributed decision-making, the different nodes on a given path may have different timer values, allowing the intermediate nodes, which have different mobility patterns, to quickly and reactively reconfigure their routing parameters. This enhanced reactivity from our cognitive framework improves the stability of the generated routes as will be illustrated in the ensuing results discussion.
### 3.5 Simulation Environment

The performance of our QLS architecture has been evaluated with the NS-2 simulation tool. As shown in Table 2, the simulation network is defined in a flat terrain of \(2000 \times 2000\) m with 100 mobile nodes, 3 MANET GWs, 2 BGP routers, and 6 Fixed Servers. Table 2 displays the summary of NS-2 simulation parameters. At the physical and data link layers, the 802.11b standard was used for analysis. The main purpose of the simulation scenarios was to provide a framework to compare the performance of AODV-Q, the solutions proposed in [69, 70], and standard AODV protocols. Results were averaged over 20 runs for each protocol, for each max pause time.

<table>
<thead>
<tr>
<th>Simulation Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Area</td>
<td>(2000m \times 2000m)</td>
</tr>
</tbody>
</table>
| Number of nodes       | * Mobile Nodes (MNs) = 100  
                        | * 3 Wireless Gateway Nodes  
                        | * 6 Fixed Nodes |
| Mobility Model        | * Random Waypoint  
                        | * Speed (m/s) = Uniform (0, 15)  
                        | * Pause Time (s) = 0, 60, 120, 180, 240 |
| Wireless Interface    | * IEEE 802.11b, 11Mbps |
| Transmission Range    | * 350m |
| Traffic Flows: CBR    | * 5 MN sources  
                        | * ↔2 Fixed Server and 3 Mobile Client destinations  
                        | * Packet size: 15kB  
                        | * Transmission rate = 10 pkts/s  
                        | * Application Profile: Lo-Res Video traffic |
| Traffic Flows: TCP    | * 5 MNs source  
                        | * ↔ 2 Fixed Servers and 3 Mobile Client destinations  
                        | * Packet size: 512B  
                        | * Application Profile: FTP traffic |
| Simulation Time       | * 10 min (for each run) |

Table 2 - Summary of NS-2 Simulation Parameters
The traffic models used to gather the simulation results consists of Constant Bit-Rate (CBR) Video Conferencing and File Transfer Protocol (FTP) application traffic profiles. In this analysis, node mobility is assumed to be random (i.e., independently selected by each node using a uniform distribution) movement rather than group movement. The mobile nodes are assigned a maximum speed of 15 m/s. In the simulation scenarios, each mobile node changes its location within the network based on the “random waypoint” model; that is, the node randomly selects a destination, moves toward that destination at a speed not exceeding the maximum speed (15 m/s) and then pauses; this interval is known as pause-time. In order to calculate the impact of high mobility on the protocol overhead, pause-time ranged from 0 to 240 seconds in duration. It should be noted that a pause-time of zero represents the worst case scenario, in terms of high topological instability, as the mobile nodes are constantly moving during the simulation.

For the AODV-Q simulations, each training episode, from the beginning of the simulation to time $t=30s$, each node randomly chooses actions decreasing or increasing active route timeout and hello interval. During the simulations AODV-Q reconfigures active route timeout between 3s and 10s, and hello interval between 1s and 10s. Table 3 conveys a summary of the results of the simulation, with the value of each performance parameter averaged across pause time runs, with percentage improvement over standard AODV listed to the right of the value in parentheses.
### 3.6 Performance Evaluation

The performance of AODV-Q was evaluated in terms of responsiveness, protocol overhead, and packet delivery ratio. The performance results are compared with those derived under the standard AODV routing mechanism, as well as the solutions proposed in [71] (Mod-AODV) and [69] (Opt-AODV).

<table>
<thead>
<tr>
<th>Protocol</th>
<th>ETE Delay (s)</th>
<th>Route Discovery Time (s)</th>
<th>Routing Traffic (packets)</th>
<th>Route Errors (packets)</th>
<th>UDP Packet Delivery Ratio</th>
<th>TCP Packet Delivery Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>AODV</td>
<td>0.104</td>
<td>0.846</td>
<td>89068</td>
<td>10013</td>
<td>0.764</td>
<td>0.89</td>
</tr>
<tr>
<td>AODV-Q</td>
<td>0.074 (-29%)</td>
<td>0.562 (-33%)</td>
<td>50047 (-44%)</td>
<td>5378 (-46%)</td>
<td>0.846 (11%)</td>
<td>0.93 (5%)</td>
</tr>
<tr>
<td>Mod-AODV</td>
<td>0.08 (-23%)</td>
<td>0.61 (-28%)</td>
<td>59134 (-34%)</td>
<td>8018 (-20%)</td>
<td>0.794 (4%)</td>
<td>0.92 (3%)</td>
</tr>
<tr>
<td>Opt-AODV</td>
<td>0.094 (-10%)</td>
<td>0.768 (-9%)</td>
<td>63181 (-29%)</td>
<td>8413 (-16%)</td>
<td>0.781 (2%)</td>
<td>0.91 (1%)</td>
</tr>
</tbody>
</table>

Table 3 - Summary of Experimental Results

#### 3.6.1 Responsiveness

The network responsiveness resulting from decisions QLS applied to AODV was evaluated in terms Route Discovery Time and End-to-End Delay. The route discovery time, measured in seconds, is the measure of how long the protocol takes to determine a valid route.
once a request has been made. Figure 11 conveys the average measure of route discovery time for all four protocols. Opt-AODV tracks more closely with the standard AODV implementation with a 9% reduction in route discovery time, whereas AODV-Q and Mod-AODV offer consistent improvement in route discovery time (33% and 27% reduction respectively). The reduction in route discovery time can be attributed AODV-Q and Mod-AODV’s more temperate approach to tuning the Hello interval. Specifically, Mod-AODV uses thresholding to tune the hello interval by 25% at most (choosing a deviation fraction of 0.75, 1, or 1.25). Rather immediately adding or subtracting values to the Hello interval. And while AODV-Q does add or subtract values to the HELLO interval, it does so using a thresholded machine-learning based approach, whereby previous Q-values are taken into account to prevent radical swings in interval values. This approach leads to a more stable and accurate depiction of network dynamics than the approach used by Opt-AODV, which adds or subtracts values based upon immediate observation.

![Graph showing network responsiveness](image)

Figure 10 - Overall network responsiveness with respect to End-to-End Delay
End-to-end delay, measured in seconds, is the measure of the time taken for a packet to be transmitted from the source and received at destination node. As delay is a good measure of the fitness of routes being selected, end-to-end delay was used as another measure of network responsiveness due to the decisions made by the routing protocols. Figure 10 displays the results of end-to-end delay measurements for all four protocols. While all four protocols generally reduced delay as network stability improved with increased pause time, using standard AODV as the performance baseline, AODV-Q and Mod-AODV exhibited 29% and 23% reductions in delay on average, respectively, whereas Opt-AODV showed a 10% reduction. Overall, end-to-end delay reduction can be attributed to the fact that AODV-Q, Mod-AODV, and Opt-AODV are tending to reduce protocol overhead over time through tuning of the HELLO Interval (see Figure 12). Reduced traffic in the wireless medium allows the QLS scheme to realize a shorter queuing delay, resulting in shorter end-to-end delays.
3.6.2 Routing Overhead

Control overhead is measured in terms of the number of control messages generated by the routing algorithm. Figure 12 illustrates the number of routing control messages generated or relayed in the network. The standard AODV mechanism generates a greater number of control messages than does AODV-Q, Mod-AODV, and Opt-AODV, with reductions at 43%, 33%, and 29% respectively. This, in turn, translates into a higher probability of lost control messages in AODV due to collisions in the wireless medium. Consequently, routing paths are less reliable under standard AODV. All three protocols make reasonable progress toward overhead reduction, but the ability of AODV-Q to retain stable routes by tuning the active route timeout parameter yields further reduction in unnecessary route discovery traffic.

![Figure 12 - Resultant protocol overhead with respect to Routing Traffic Generated](image-url)
In simulations, the Q-learning agent at each node self-configures the active route timeout and hello interval according to the Q-value. Due to the distributed self-configuration of these parameters, the nodes send RREQs more appropriately to account for failed routes, improving the route freshness and the link failure detection processes. The route error is evaluated as the average number of RERR packets per second. As shown in Figure 13, Mod-AODV and Opt-AODV yield a RERR reduction of 20% and 16% respectively, whereas AODV-Q yields a RERR reduction of roughly 46%. While the previous two protocols have the ability to tune the HELLO message interval to alleviate unnecessary congestion and produce better routes, AODV-Q has the added advantage of being able to also tune the active route timeout interval. This allows the protocol to hold onto routes longer than the other three protocols when network stability is perceived over time. This behavior tends to favor stable routes being used longer, hence the reduction in route errors over time.
3.6.3 Packet Delivery Ratio

The third criterion we use for evaluation is that of the packet delivery ratio (or rate of successful packet transmission), which is the number or transmitted packets divided by the number of received packets, equivalent to

\[
\text{Packet\_delivery\_ratio} = 1 - \text{packet\_loss\_rate}.
\]

The delivery ratio was measured with respect to the two traffic types that traverse the wireless portion of the network, constant bit rate (CBR) UDP video traffic, and TCP-based FTP traffic. Figure 14 conveys the results for packet delivery ratios for UDP video traffic. While Mod-AODV and Opt-AODV yielded a 4% and 2% improvement respectively, AODV-Q showed an 11% improvement in delivery of video traffic. The tendency of AODV-Q to hold onto more stable routes in addition to the reduction in protocol overhead contributed to the larger percentage improvement for video traffic. Video conferencing applications generate packets with very short inter-arrival times, prompting the Q-learning agent at each node to self-configure a shorter active route timeout and hello interval. AODV shows large video conferencing packet delay variations due to its lack of efficiency and timeliness in finding new routes. AODV also shows larger end-to-end delay for video conferencing packets when the mobile nodes are highly dynamic, such as when the pause time is zero.
Figure 14 - Packet Delivery Ratio with respect to CBR-UDP video traffic

Figure 15 - Packet Delivery Ratio with respect to TCP-based FTP traffic model

Figure 15 illustrates the packet delivery ratio for FTP traffic governed by TCP congestion control. In this case, the improvement upon the standard AODV implementation was less pronounced, with AODV-Q, Mod-AODV, and Opt-AODV yielding 5%, 3% and 2% performance improvements respectively. TCP’s built-in congestion control can claim some
responsibility for the higher packet delivery ratios of all protocols, since the constant bit rate video traffic was sent regardless of congestion or loss feedback account for most of the difference in performance between the two traffic models. However, in general, we do see a larger performance improvement with AODV-Q due to the increased longevity of stable routes.

3.6.4 Impact of Learning Rates

In the preceding experiments, we executed the simulations with a relatively high learning rate ($\alpha = 0.1$). To analyze the impact of the learning rate itself upon the observed performance enhancement of QLS, we present the results of simulation scenarios with three learning rates: 0.01, 0.05, and 0.1.

First, we denote the various traffic types by index number, delineated by the mobility of the nodes involved:

- Profile 1: Ethernet-to-MANET
- Profile 2: MANET-to-Ethernet
- Profile 3: MANET-to-MANET

Figure 18 conveys the fact that higher learning rates improved the packet delivery ratio for MANET Traffic Profile #3 (MANET-MANET). However, we found that lower learning rates were more advantageous for Traffic Profiles 1 and 2, with respect to improving their localized delivery ratios. We infer from these results that higher learning rates are more advantageous in highly mobile environments, where quick adaptability can enhance network performance.
Further, such high learning rates could be harmful to semi-static nodes or node collections within the network, in that either the source or the destination is relatively stable with respect to the rest of the network.

Figure 16 - Packet delivery ratio with respect to mobility and learning rate
Figure 17 - Packet delivery ratio with respect to mobility and learning phase

Figure 18 - 2D Contour Graph of Packet Delivery Ratio with respect to Mobility and Learning Phase
Additionally, we devised a dynamic self-reconfiguration mechanism for determining the learning rate in real time. The learning rate updates occur each time a node agent computes its respective Q-values. If one of the following two events occurs, then the learning rate $\alpha$ is computed by the following means:

1) $\alpha = \frac{\alpha}{2}$: Occurring when an agent receives a reward (the ROUTE REPLY packet reaches the source and there is a path from the source to destination).

2) $\alpha = 2\alpha$: Occurring when an agent receives a penalty (a ROUTE ERROR packet is generated or a ROUTE REPLY packet reaches the source but there is no path from the source to destination).

As shown in Figure 18, self-reconfiguration of the learning rate enhanced the packet delivery ratio for Traffic #1 (MANET-MANET) and Traffic #3 (Ethernet-MANET). For traffic profile #2 (MANET-Ethernet) the self-reconfiguration of the learning rate was not as effective.

One solution to optimizing the learning rate is to use Bayesian exploration [78], (to be explored in our future work) to tune and optimize learning rate values with respect to network performance. There further exists a need to investigate the optimization accuracy and the process of reward value assignment in the Q-value computation, in addition to the selection of correct parameters for self-configuration.
3.7 Conclusion

In this work, we described a proposed framework for autonomously reconfiguring multi-hop wireless network systems with a cross-layer approach in heterogeneous environments. AODV-Q has been proposed to improve the performance of AODV, through the use of iterative network state observation. This approach is applicable to large heterogeneous networks, where the characteristics of the mobile nodes and application demands are different. We also presented experimental results. The performance results confirm that QLS-enhanced AODV-Q dramatically reduces the protocol overhead compared to the standard AODV. AODV-Q achieves a higher packet delivery ratio while incurring shorter queuing delays. Specifically, with AODV-Q, it is possible to achieve shorter end-to-end delays while reducing the incidence of lost data packets. Therefore, the proposed autonomous self-configuration mechanism successfully improves the scalability and adaptability of the original AODV protocol in a heterogeneous network environment.
Chapter 4  FlowBroker: A Software-Defined Network Controller Architecture for Multi-Domain Brokering and Reputation

4.1 Introduction

Today’s managed network deployments are largely comprised of devices, switches and routers, which are based upon non-programmable implementations of network protocols with little or no ability for adaptation or meaningful modification. Moreover, if a developer or researcher wishes to modify or innovate upon existing protocols, or even create new protocol architectures within the existing legacy device topologies, this would be impossible. To support the idea of a programmable and flexible architecture in which network development and experimentation can take place, Software Defined Networking (SDN) was proposed [79, 80]. SDN provides programmability over network resources, by means of centralizing the network control plane in a programmable controller, and distributing the data plane over SDN switches. By providing a modifiable controller, SDN provides great flexibility in configuration and management of programmable switches. Current SDN implementations, [81, 82] allow for the dynamic reconfiguration of switches using controllers to send “flow rule” updates to these switches, governing how the SDN-enabled switches handle incoming traffic. Customers deploying SDN-enabled network elements now have more control over protocol behavior and packet forwarding, and more freedom to run various disparate network implementations across the same physical network infrastructure. This programmability of SDNs also provides an
opportunity for increased automation of network monitoring and policy-based network management [83].

Initially, the SDN architecture assumed that a single controller would be responsible for any number of switches in an SDN network. This assumption is infeasible due to concerns of both scale and reliability. Single controller deployments would likely grow untenable as the number of switches increases due to the increased volume of flow requests and the processing limitations of the controller itself [84]. Moreover, a single controller would also put the entire network at risk if a failure were to occur at that controller. As the number of deployment scenarios of SDNs expands to include areas such as home network management and access and core network management [85], scalability concerns are introduced in light of the need for additional controllers. At the very least, a failover controller should be provisioned in the event that the primary controller is unable to communicate with the rest of the network. A more scalable approach would be to divide the network into managed domains, and assign a controller to manage each such domain, allowing for faster processing of flow requests and lesser computational burden upon each individual controller. While the current SDN standards do allow for multiple controllers in an software-defined network, where each controller is responsible for a subset of the SDN-enabled switches, there is currently a need for a cooperative framework whereby the controllers can communicate with one another and allow for flow table entries to be efficiently constructed for inter-domain traffic. Clearly an architecture is needed to deal with issues of scale and inter-domain [86] forwarding and management in a large SDN deployment scenario. In this chapter we propose a solution to multi-domain flow coordination, the FlowBroker architecture, as a collaborative approach to load balancing and network performance enhancement in software-defined networks. By introducing physically distributed
broker agents which aggregate network state information, compute efficient inter-domain forwarding paths, and periodically recommend updated disjoint inter-domain paths for load balancing, we show that FlowBroker provides a cooperative, distributed and scalable approach to multi-domain flow management in software-defined networks. Moreover, we propose a machine learning-based performance reputation learning agent within each controller, to allow for the determination of broker reliability in the event that a controller is associated with more than one broker [87]. Initial results illustrate the promise of the FlowBroker architecture in terms of the network performance improvement that results from employing both hierarchical load-balancing brokers as well as intelligent semi-autonomous controller agents.

The remainder of this chapter is organized as follows: Section 4.2 presents background on SDN networks, a brief survey of related work, and discusses the issues of scale and performance inherent in multi-domain software-defined networks. Section 4.3 discusses the proposed FlowBroker distributed SDN architecture for brokered load balancing. Section 4.4 discusses a distributed agent-based learning approach to broker reputation. Section 4.5 discusses the emulation environment in which FlowBroker is implemented and tested. Section 4.6 contains a discussion regarding simulation results and corresponding analysis. Section 4.7 concludes with a brief overview of this work and a discussion of future research directions.
4.2 Background and Related Work

In the standard SDN architecture, when a packet arrives at a switch port, the switch will interrogate the packet header and compare the fields with its flow table entries. The switch will look for a match with respect to various types of match rules and attributes, such as the ingress port number on which the packet arrives, or the destination IP address. The actions that can result from the lookup are:

- forward the flow’s packets to the following port(s),
- encapsulate and forward this flow’s packets to a controller (for the first packet in a new flow, allowing the controller to determine whether to add the flow entry to the table),
- drop this flow’s packets (used for security to curb DoS attacks or the reduce broadcast discovery traffic), and
- forward this flow’s packets through switch's normal processing pipeline (non-SDN normal L2/L3 approach).

If a match is found, the packet will be forwarded or dropped depending upon the specified rule for that flow table entry. If there is no match, then the packet is forwarded to the controller. The controller then has the choice to either add a new entry to the switches’ flow table, which will handle subsequent packets with similar header fields, or simply drop the packet.

The standard SDN process to handle packet forwarding is not feasible in multi-domain multi-controller implementations for two reasons. The first is that Link Layer Discovery Protocol (LLDP) packets will flood the entire network until the destination is found. As the network size grows, this overhead will increase. When a new packet arrives at a switch, if there is no match against the flow table entries, the switch will forward the packet to the controller for input. If the
destination of the packet resides within the domain of this switch, then the controller can simply install a flow table entry that handles the forwarding within its managed domain. However, if the destination of the packet resides within another managed domain, this will result in the controller flooding the network with LLDP packets to discover the destination and create the corresponding flow rules in each switch's flow table that it manages. However, the entire network is flooded just to update the flow tables of switches which reside in the current domain, not all domains. As a result the LLDP flooding will occur for each domain which encounters the flow packet for the first time, creating redundant discovery overhead. The second reason for the standard workflow being inadequate in a multi-domain scenario is that the route discovery process will not take into account network performance statistics, such as load on individual links, but rather will favor the shortest path. This shortest path determination is not cognizant of the load upon each of the links, the packet latency occurring along that path, or the difference between an inter-domain link and an intra-domain link, and hence could create sub-optimal routes which are detrimental to overall network performance. To overcome these limitations, we propose the FlowBroker architecture, which provides scalability by using regional network state information for inter-domain flow forwarding. The next section discusses recent approaches to handling multiple controllers in an SDN network.

4.2.1 Distributed SDN Controller Implementations

Recently, there have been a few approaches to a multi-controller architecture in software-defined networks. In [88], Tootoonchian and Ganjali describe the distributed control plane approach of HyperFlow. HyperFlow creates a distributed control plane for software-defined networks through replication of controller state by means of periodic updates. In Hyperflow, all
controllers have the same network view, and Hyperflow is an application extension to the NOX [89] controller implementation. Using publish/subscribe notification, the Hyperflow controllers publish relevant local events for other controllers to consume and update their local network state. The WheelFS [90] distributed file system (DFS) is leveraged to provide wide-area storage for the network events on each controller instance. Each controller subscribes to three event notification channels (data channel, control channel, and their own channel) Hyperflow instances publish selected local network/application events which are of general interest to the data channel. Events and SDN commands pertaining to a specific controller are published to that controller's channel. Each controller periodically advertises itself on the control channel for discovery and failure detection purposes. Hyperflow provides facilities for Event Publication, Event Replay, Event Filtering, Proxying Openflow Messages, and Health Check updates for failover purposes.

In [91], Sherwood et. al. introduce FlowVisor, which is a second, and more Virtualization-focused multi-controller implementation. FlowVisor’s aim is to create a hierarchical network virtualization layer between controllers and switches. FlowVisor builds an abstraction layer upon OpenFlow to "slice" network resources (e.g. bandwidth, topology, traffic, CPU time, forwarding tables) into virtual networks, defining a slice as a subset of flows running on a topology of switches. FlowVisor sits between the SDN controllers and switches, to ensure controllers only access and control the switches they are supposed to, partitioning the bandwidth, as well as the flow-table in each switch by keeping track of which flow-entries belong to each guest controller. FlowVisor is implemented as an OpenFlow proxy, intercepting messages between SDN-enabled switches and Controllers. FlowVisor instances intercept SDN control messages between controller and switch and rewrite the content of those messages to ensure the
controller only modifies flows it has access to (flows in that controller’s particular slice). It is also possible to have a hierarchy of FlowVisor instances, and therefore controllers, thereby virtualizing a virtual network(s). FlowVisor controllers do not coordinate with one another, and therefore do not optimize global network performance, but rather are only concerned with the operation of their respective network slices.

A third, and more datacenter focused, approach to distributed SDN controllers is Switching with In-packet Bloom Filters (SiBF) [92]. SiBF is an approach to distributed management which exchanges global state using a globally available NoSQL database. SiBF leverages Rack Managers (RM), one per switch rack, to implement routing logic by installing flow states at top-of-rack switches. Each "Rack Manager" maintains a distributed NoSQL database instance holding link states, topology info, and other event information. RMs independently write discovered events to and continuously read topology information from the distributed NoSQL database. RMs continuously cache network link map information in the datacenter. Each RM acts as a controller and SiBF RMs are implemented as an application on top of the NOX controller. Flow requests are handled locally (at the rack level). Additionally, directory and topology information are stored in the distributed database, and a standby RM is provisioned for failover purposes.

In [93], Phan et. al. propose a multi-controller architecture called RMOF. The goal of the RMOF model is to create a collaborative system in which domain controllers can efficiently route inter-domain flow packets using the assistance of a global RMOF collaborator instance. The RMOF collaborator system maintains a global view of the entire network by capturing routing information from the individual domains from each controller, and formulating the inter-domain routes for network-wide traffic flows. The RMOF collaborator maintains a global link table of all network connections.
cross-domain links, including the source controller/switch and the destination controller/switch. A secondary estimated routing cost table is maintained by the RMOF collaborator to track the cost of the shortest path to traverse each domain, where the cost is the number of hops in that path. In this architecture, controllers are responsible for local intra-domain routing, whereas the RMOF collaborator is responsible for global routing. While the division of forwarding responsibilities between controller (intra-domain) and RMOF collaborator (inter-domain) makes sense from a scaling perspective, eliminating the need to flood the entire network with LLDP packets, this approach seems to suffer from two drawbacks. Firstly, the RMOF approach does not take into account network performance in its path cost computation, but rather focuses simply upon the hop count, thereby leaving itself vulnerable to performance degradation due to congestion. The second potential issue with this approach is the lack of scale at the RMOF collaborator level. As the number of domains increases, the single RMOF instance remains fixed in computational capabilities and creates a single point of failure. While intra-domain routing will continue in the absence of an RMOF instance, inter-domain routing will be significantly degraded.

Finally, the latest OpenFlow switch specification [82] does allow for multiple controllers to be used within a single software-defined network. This approach allows for increased reliability due to the ability of a switch to fail over to another controller in the absence of the primary. However, the specification does not provide for the ability to use multiple controllers simultaneously to improve control plane scale and performance. Rather, the specification provides for a single software-defined network to have multiple controllers on standby, where only the primary controller can actively manage all switches within the network. Clearly there exists the need for a distributed and collaborative model in SDN, where controllers can actively manage a subset of the switches in the network to provide scale and performance benefits.
4.2.2 Issues in Multi-Domain SDN

As evidenced in various recent papers regarding SDN and scalability [94-97], any examination of SDNs that involves an increasing switch count, relative to a centralized controller, leads to concern about whether the SDN approach to network architectures will be scalable or performant. In [98], Levin et. al. discuss the inherent tradeoffs of a logically centralized but physically distributed control plane when attempting to provide a distributed controller implementation. Clearly a physically centralized control plane architecture will fail due to scale, performance, and reliability constraints, given sufficient numbers of switches to manage. However, the amount of overhead required to maintain a degree of logical centralization in the control plane is something to consider. Moreover there is the concern of how consistent the global network state view is in a distributed control plane. A strongly consistent view will naturally lend itself to high overhead due to the frequency of update messages and computations to reconcile any inconsistencies across controllers, whereas a weakly inconsistent network state view will lead to higher responsiveness but less precise policy responses by network management agents. In subsequent sections regarding performance analysis, we examine this tradeoff by simulating the proposed FlowBroker architecture, varying the update interval of the network controllers and measuring the corresponding network performance levels.

The present-day Internet architecture leverages administrative domains to create scalability and flexibility in the face of heterogeneous networked systems. However, challenges remain such as autonomy, security, and end-to-end QoS capabilities. We choose to leverage and build upon this
architecture by adding software-defined networking in conjunction with broker agents, to allow each domain the autonomy to freely associate with brokers which provide the services for each domain’s inter-networking requirements. Our proposed solution to create both scale and reliability in multi-domain multi-controller software-defined networks is the FlowBroker distributed controller and brokering architecture. FlowBroker differs from previous approaches in the following ways:

- whereas previous approaches centralize the management plane both physically and logically, our approach relies on a physically distributed management plane for brokering of load balancing services between managed controllers,

- metrics used to compute link cost for inter-domain forwarding and load balancing decisions are based upon the network performance state of the links involved, and not merely upon the hop count,

- controllers in each managed domain have the freedom to associate with more than one broker and choose whether or not to implement the flow rule recommendations that broker computes from distributed network state updates, and

- FlowBroker leverages machine learning for the distributed learning of a broker’s reputation of performance, where learning agents associate the current network state with the performance of the broker currently in use. These observations and training data then inform controllers over time which brokers are more ideal to leverage given the current state of the network. Essentially, the learning agents help the controller predict future network performance based upon past broker performance history.
The following two sections contain a more detailed description of the FlowBroker architecture, as well as the algorithms used to determine broker recommendations, broker performance reputation, and market-based brokering.
4.3 FlowBroker

FlowBroker is a multi-controller solution that employs distributed systems concepts to support multiple SDN controllers for coordination across managed domains. Physically distributed brokers are used to collect and aggregate network state information for each region of the network, and compute better global forwarding decisions on a per-flow basis to provide global load balancing and performance optimization. The following sections describe the FlowBroker architecture in detail.

Figure 19 - Multi-Domain FlowBroker
4.3.1 FlowBroker Architecture

In the FlowBroker architecture model, conveyed in Figure 19, the layered interaction begins with the Forwarding Plane, which consists of SDN-enabled switches, subdivided into their respective managed domains. Each domain is managed by a single FlowBroker-compliant controller, and network state and flow table update instructions are exchanged between the two via the standard SDN control protocol. The control plane then consists of the collection of distributed controllers, each of which are tasked with managing the flow tables of the switches within their respective domains. The management agents within the controllers extract link state, topology updates, and traffic statistics, such as Packet Latency, Link Utilization and Packet Loss ratios, from the switches which they manage. These data are then used for the optimization of internal packet forwarding within the switch domain. Controllers also collect this network state information to forward to the brokers with which they are associated. Moreover, controllers also have a management plane component, as they are responsible for autonomous management of their respective switch domains. This idea of controller autonomy is further illustrated later on when we discuss the ability of controllers to selectively associate with brokers, and listen to or ignore their inter-domain forwarding advice.

A broker is a physically distinct software process which facilitates the exchange of global network state, as well as offers policy-based forwarding modifications (flow table updates) for the controllers with which it associates. FlowBroker instances are not co-located on controller hosts in order to provide both flexibility in terms of the number of possible brokers, and independence between broker entities and the controllers with which they interact. Brokers are
independent service entities tasked with awareness of the global state of the network, and are ideally situated to give flow rule recommendations for the forwarding of inter-domain traffic. Each broker acts as an optional network state interface between a given controller and the rest of the network by periodically collecting the network state from the controllers it associates with, and exchanging this state with other peer broker instances. The FlowBroker architecture allows for the individual controllers to manage their domains with little or no interference, thereby minimizing the impact on intra-domain flow table maintenance. This allows the architecture to provide a scalable solution to inter-domain load-balancing as broker decisions are only required for inter-domain traffic flows. Moreover, the architecture provides for one or more failover controller to exist in the event that the primary controller fails. Secondary controllers monitor the health of the primary controller via CTRL_KEEP_ALIVE message exchanges between each controller in the same domain, every two seconds. The primary controller also sends periodic CTRL_TABLE_BACKUP messages to the secondary controller(s) every five seconds, to mirror any flow table updates, which occur on the primary controller. These messages also contain other data reflecting the current state of the primary controller, such as broker reputation calculations and broker association preferences.

At initialization time, each controller registers with at least one broker, and attempts to register with at least one secondary broker instance as well. The link state update process with the controller then proceeds to collect domain-specific traffic and topology state updates, and forwards them to each associated broker by means of the FlowBroker control channel. Each broker/controller association creates a distinct control channel on controller port 6680, which is used for sending NETWORK_STATE, GLOBAL_ROUTE_REQUEST and KEEP_ALIVE_REQ messages to the broker, and for the broker to send FLOW_TABLE_MOD and
KEEP_ALIVE_RESP messages back to the controller tier. The controller maintains a link statistics table for each link adjacent to a switch in its managed domain with the following format:

<table>
<thead>
<tr>
<th>Src Switch ID</th>
<th>Dest. Switch ID</th>
<th>Src Port</th>
<th>Dest Port</th>
<th>Link Capacity</th>
<th>Link Utilization</th>
<th>Delay</th>
<th>Loss Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>16 bits</td>
<td>4 bits</td>
<td>4 bits</td>
<td>16 bits</td>
<td>8 bits</td>
<td>16 bits</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

This table is used by the controller for intra-domain load balancing, where the controller checks the utilization level of a link when it is recomputed, and adjusts the forwarding rules in the appropriate flow tables accordingly (see the next section for a description of the load balancing procedure).

The fields of interest in the controller’s network domain state table are:

- Src Switch ID: the unique ID number of the switch on the source end of the link,
- Dest. Switch ID: the unique ID number of the switch on the destination end of the link,
- Src. Port: the port number on the source switch to which the physical link connects,
- Dest. Port: The port number on the destination switch to which the physical link connects,
- Link Capacity: The capacity of the link in Mbps,
- Link Utilization: the ratio of rate of traffic currently traversing the link to the link capacity since the last state update,
- Delay: the total average delay encountered when transmitting a packet out the egress interface of the source switch, combined with the propagation delay and processing delay at the destination switch, and computed since the last state update, and
• Loss Ratio: the traffic sent by the source switch minus the traffic received by the destination switch, divided by the traffic sent by the source switch, since the last state update.

The controller sends updates of this state table to each broker it is associated with. Each broker, then in turn, maintains a regional network state table consisting of only the intra-domain links which exceed the utilization threshold of 0.7, and inter-domain links. The broker network state table has the following proposed format:

<table>
<thead>
<tr>
<th>Src Controller ID</th>
<th>Dest. Controller ID</th>
<th>Src Switch ID</th>
<th>Dest. Switch ID</th>
<th>Src Port</th>
<th>Dest Port</th>
<th>Link Capacity</th>
<th>Link Utilization</th>
<th>Delay</th>
<th>Loss Ratio</th>
</tr>
</thead>
</table>

The additional fields of interest introduced in the broker’s global network state table are the following two 16 bit fields:

• Src. Controller ID: the unique ID of the controller which manages the domain in which the source switch resides, and

• Dest. Controller ID: the unique ID of the controller which manages the domain in which the destination switch resides.

Note that if the source controller ID and the destination controller ID are the same, then this implies the entry is a critical link in that controller’s domain. The broker periodically shares the updates to its regional network state table with other peer brokers in the network to provide a global view of current network performance. Given enough updates within the
STATE_CONVERGENCE_INTERVAL, a broker will then compute globally optimal inter-domain paths for both forwarding and load-balancing purposes, using network performance statistics to drive the link costs. The next section discusses the forwarding and load-balancing workflow of the FlowBroker architecture.

4.3.2 Load-Balancing and Forwarding with FlowBroker

FlowBroker considers the tasks of load balancing and forwarding to be directly related, meaning any forwarding decision, whether intra-domain or inter-domain in nature, will have link utilization as the primary driver of the path computation, and will therefore attempt to compute a disjoint alternate path to avoid links with high utilization. Since the controller does not know a
priori the data rate which a flow request will produce, and therefore does not know the network resources which need to be provisioned to achieve optimal load balancing and performance, our approach to forwarding and load balancing is divided into two scenarios: new flow requests and periodic flow updates. New flow requests occur when a packet of unknown flow type is encountered by a switch with no corresponding flow table entries which match the packet's various header fields. Figure 20 conveys the procedure which takes place in FlowBroker when a new packet type is discovered.

1. The edge switch receives a packet from a host or external network.

2. The packet in question does not match any rules in the switch's flow table, and is therefore forwarded to the switch's controller.

3. The controller determines whether the packet's destination is in the local managed domain by comparing the destination address in the packet's header with the list of known switches and hosts in the controller's domain. If there is no match, then the controller will forward this packet's information to the broker it is currently using via the FlowBroker control channel, sending a GLOBAL_ROUTE_REQ message for forwarding rules for the new flow type.

4. The broker then proceeds to compute an optimal inter-domain forwarding path based upon the entries in its global link state table, using Dijkstra's algorithm [99] to compute the shortest path where link utilization is the inter-domain link cost. The inter-domain path consists of all inter-domain links used in that path and the switches that they are attached to. The broker then forwards this computed path result to other brokers in a GLOBAL_ROUTE_CREATE message.
5. Any broker associated with a controller whose domain is in the inter-domain forwarding path will send this global flow path update to that controller in the form of a FLOW_TABLE_MOD message.

6. Controllers which received the TABLE_FLOW_MOD messages from their respective brokers proceed to create the intra-domain paths which support the new inter-domain path, by computing the shortest path using link utilization from the controller's local link statistics table as the link cost. Subsequent packets in the flow are now to be forwarded to the appropriate edge switch.

7. The new flow packet is then forwarded to the next domain in the inter-domain route, which has established a valid intra-domain path leading to the next-hop domain in the global path.

Figure 21 - Flow Broker Load Balance Update Workflow
Since the data rates of the traffic along the newly constructed flows is not known at the time of rule creation, there is a need to periodically monitor the network link utilization in order to provide optimal guidance on flow rule updating. The periodic NETWORK_STATE messages sent from controller to broker serve to update the broker-layer view of the entire network, and in particular, the utilization levels of intra-domain links. If these inter-domain links are utilized by 70% or above, the broker will automatically compute an alternate inter-domain path for the flow which contributes the most traffic to the congested path. Figure 21 illustrates the procedure of multi-domain periodic load balancing with respect to the following steps below.

1. Controllers collect network state information from each switch in their managed domain. These data include link utilization for every adjacent link, end-to-end delay for all flows which originate in this domain, and packet loss data for all links adjacent to a switch in this domain.

2. Brokers periodically receive link state information from the controllers with which they are associated, in the form of a NETWORK_STATE message, to gather network performance statistics from each domain. These data are then used to summarize the current network performance of the domain as a whole, as well as to determine the levels of link utilization of any inter-domain switch links in particular. If an inter-domain switch link has utilization equal to or greater than 70%, the broker will compute, if possible, an alternate inter-domain path for at least one of the flows traversing the congested inter-domain link, namely the one that contributes the most traffic to that link. Alternate paths
are computed using Dijkstra's algorithm, leveraging link utilization as the cost metric of interest. The alternate path with the lowest maximum link utilization becomes the recommended path and a flow table entry is formulated to re-route at least one flow traversing the congested inter-domain link(s).

3. Brokers exchange network state information with one another to provide global convergence of domain specific network performance knowledge. These data inform the brokers as to the current state of other inter-domain links in the network allowing for a more cooperative, accurate, cross-domain approach to traffic forwarding and load balancing. Once network state has triggered the path computation phase, and once the computation is complete, the broker then sends the GLOBAL_ROUTE_UPDATE to all peer brokers reflecting the new inter-domain path.

4. Brokers then send FLOW_TABLE_MOD messages to relevant controllers, which manage domains along the prior inter-domain path, and the new inter-domain path.

5. Finally, the controller implements the flow modification instructions the broker has sent, by installing or removing flow table entries in each relevant domain switch.

The FlowBroker architecture allows for the association of more than one broker per controller, and as such, a controller may receive potentially conflicting or sub-optimal flow table recommendations. We discuss in the next section a machine learning approach which allows the controller to reconcile this by means of calculating and categorizing a broker's performance reputation based upon the current state of the network.
4.4 Broker Reputation

Controller performance is one of the main concerns which drives the need for a multi-domain approach to SDNs. Though it is possible to maintain a logically centralized view of network state with current controller implementations [100], as the size of the network increases, it becomes increasingly infeasible to expect global knowledge to propagate between domains in an efficient manner without consuming unnecessary resources. The addition of the brokering agents alleviates the need to communicate intra-domain link state, and allows the management of SDNs to scale, and controllers to focus on only the links and switches within their respective domains, as well as the adjacent links which join them to other domains. However, this scalability is dependent upon the reliability of the broker and the inter-domain load-balancing and forwarding decisions that it makes being advantageous to network performance. Broker implementation issues, such as software failures or outright malicious behavior, can significantly impact the overall performance of the multi-domain SDN deployment. Therefore broker performance should be observed to allow both controllers and peer brokers to avoid harmful broker associations. In the following section we discuss the need for each controller and peer broker to intelligently determine the usefulness of forwarding and load-balancing decisions provided by each other broker, and the capability for learning agents within each controller and broker to decide which broker to listen to or peer with, based upon this determination.
4.4.1 Broker Performance Reputation

Though the broker provides input regarding optimal forwarding paths, the controllers and peer brokers have the freedom to listen to another broker’s input regarding flow rule changes and peer broker forwarding updates. This ability to decide which broker to listen to creates the need to distinguish the rules that different brokers propose by somehow quantifying the "goodness" of these recommendations, and therefore the brokers themselves. The issue becomes one of how to reconcile potentially conflicting load balancing policy. We propose framing this question as an issue of broker reputation. So given a peer controller’s experience, and current controller’s experience, relative to the broker in question, can the current controller trust the information given by the current broker? Moreover, can a given broker trust another peer broker to provide reliable forwarding information regarding regions of the network about which the original broker has little or no state information, and should that second broker be peered with in collaborative fashion with the first broker? To this end, there must be a way to measure the “goodness” of broker’s state-driven policy updates, given past experience.

With respect to the quantification of a broker’s reputation, we propose using three network measurement metrics of interest:

- End-to-End (ETE) Delay: Average ETE delay using this broker’s flow rules,
- Max Link Utilization Ratio: The maximum utilization observed thus far using this broker,
• Packet Loss Ratio: Number of unsuccessful packets divided by total packets sent using this broker.

Since the main purpose of the brokering framework in FlowBroker is to facilitate global performance optimization by means of local and regional load balancing, we chose these three metrics as key indicators of just how well the brokers are balancing the flows in the network. The Max Link Utilization tells the controller just how congested the worst links are, and the ETE delay and Packet Loss Ratio inform the controller learning agent regarding regional congestion. As we discuss in subsequent sections, these data are then used to inform the learning agents, within the controllers and peer brokers, of which broker to use, leveraging the machine learning technique of Linear Discriminant Analysis to take historical behavior into account relative to each broker.

4.4.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) [101, 102] is a commonly known machine learning technique used for classification and dimensionality reduction. LDA functions by finding a linear combination of features, which can be used to classify two or more groupings or classes. LDA explicitly looks for the differences in the classifications, whereas other approaches, such as Principal Component Analysis (PCA), do not. Moreover, LDA requires the distinction between continuous independent and dependent variables be made prior to the invocation of the learning algorithm.

For LDA to work effectively, the agent must first collect sufficient training data, otherwise referred to as feature vectors, represented by $F_i = \{f_1, f_2, ..., f_d\}$ where $d$ is the number of features
in the vector and $F_i$ is the $i^{th}$ feature vector. For example, a feature vector could represent a particular transaction, and each feature would then represent a characteristic of that transaction, such as the number of items or time of day the buyer wishes to execute the transaction. We could then use LDA to determine whether that transaction would be successful using a particular seller or not.

After sufficient training data is collected, we can then train the LDA learning agent to categorize different feature vectors based upon the categorization of the historically collected features vectors, by first associating the historical data with their groupings. In the case of the user transaction, it would be whether or not the exchange was successful. In the case of our broker distributed architecture, we categorize the state of the network relative to the broker currently used into a three feature vector $F_i = \{f_1, f_2, f_3\}$, where

- $f_1$ is the ETE delay,
- $f_2$ is the packet loss ratio, and
- $f_3$ is the max link utilization ratio.

The learning agent then assigns each tuple in the training set a classification value label $v_i | 1 \leq v_i \leq 10$, where a value of 1 corresponds to the most desirable network state and a value of 10 corresponds to the least desirable. The training data classification value is computed by the following formula,

$$v_i = \left[ \frac{(u_i - m_i)}{f_{3i}} \times |C| \right]$$
where the set of classification values \( C = \{1, 2, \ldots, 10\} \) and \( |C| = 10 \). The variable \( u_i \) represents the average link utilization observed, \( m_i \) is the minimum link utilization observed, and \( f_{3i} \) is the max link utilization observed up until state \( i \). The goal of the controller is to minimize the value of \( v \) in the future by training against past feature vectors and learning which broker has the best reputation, relative to the three features observed in the feature vector.

Upon classification of the training data, the LDA learning agent then categorizes the historical feature vectors into their respective groups

\[
G_{10} = \begin{pmatrix}
    f_{1,1}(10) & \cdots & f_{1,3}(10) \\
    \vdots & \ddots & \vdots \\
    f_{n,1}(10) & \cdots & f_{n,3}(10)
\end{pmatrix}
\]

\[
G_9 = \begin{pmatrix}
    f_{1,1}(9) & \cdots & f_{1,3}(9) \\
    \vdots & \ddots & \vdots \\
    f_{n,1}(9) & \cdots & f_{n,3}(9)
\end{pmatrix}
\]

\[
\vdots
\]

\[
G_1 = \begin{pmatrix}
    f_{1,1}(1) & \cdots & f_{1,3}(1) \\
    \vdots & \ddots & \vdots \\
    f_{n,1}(1) & \cdots & f_{n,3}(1)
\end{pmatrix}
\]

where group \( G_i \) corresponds the feature vectors with a classification value of \( i \), and \( f_{1,1}(10) \) is the feature value for the first feature in the first tuple in group ten. The learning agent then performs the discriminant analysis by first calculating the centroid of each group. This is done by determining the average for each feature for all vectors in the group.
\[ c_i = \frac{1}{n_i} \left( h_{n_i} G_i \right) \]

where \( n_i \) is the number of vectors in the \( i^{th} \) group, and \( h_{n_i} \) is a column vector of ones. Additionally, the global centroid \( c \) is calculated by

\[ c = \frac{1}{n} \left( h_n \right) \begin{bmatrix} G_{10} \\ \vdots \\ G_1 \end{bmatrix} \]

In LDA the degree to which the each class is separable is determined using internal and external variance. The internal variance for the \( i^{th} \) classifications is determined by

\[ S_{int} = \frac{1}{n_i} \left( G_i - h_{n_i} c_s \right)^T \left( G_i - h_{n_i} c_s \right) \]

and is the expected covariance for each class or *within-class scatter*. The external variance, or *between-class scatter*, for all groups is determined by

\[ S_{ext} = \frac{1}{n} \left( n_1 (c_1 - c)^T (c_1 - c) + n_2 (c_2 - c)^T (c_2 - c) + \cdots + n_{10} (c_{10} - c)^T (c_{10} - c) \right) \]

which is the covariance of all ten groups. LDA’s goal is to find the projection which maximizes the ratio of between-class means normalized by the within-class scatter, referred to as the Fisher criterion, and defined as:

\[ J(w) = \frac{w^T S_{ext} w}{w^T S_{int} w} \]
where the maximization of $J(w)$ leads to the projection vector corresponding to the highest eigenvalue. Finally, once the LDA learning agent is trained, subsequent feature vectors are classified by the following equation

$$D_i = v^T p - v^T c_i,$$

where the minimum $D_i$ determines the group membership of that feature vector as group $i$.

### 4.4.3 LDA-based Learning Agents

The LDA learning agents within each controller and broker are tasked with two responsibilities, namely those of learning the categorization of training data comprised of network state information, relative to the broker currently being used or peered with, and secondly to categorize current network state data to determine which broker to use or peer with in the future. Algorithm 1 comprises the broker reputation learning algorithm and is executed as follows. From time $t=0$ to time $t=$training_phase_dur, each controller will associate with, listen to, collect and implement advice from each broker for a fixed time interval. During the training phase, the LDA training data is generated for each broker. Data inputs are in the form of feature tuples and the value formula is then used to categorize the observation tuples into their respective groups. In the case of broker reputation learning, there are 10 levels corresponding to 10 groups, group 1 consisting of highest-performance to group 10 consisting of lowest performance or least usable flow rule advice. Each controller agent has an LDA learning instance for each broker it is
associated with. Each LDA instance learns to classify network state tuples based upon the training data relative to the broker it is tasked with learning.

Once each LDA instance has been trained to categorize the network state tuple training data, the controller and broker agents then leverage the trained LDA instances to determine which broker’s rule modifications to implement, or in the case of broker agents, which peer brokers to collaborate with. The broker selection process is implemented as follows. Upon completion of the training phase, the current network state feature vector is applied to each the classification equations for each LDA instance, yielding a classification score for each broker relative to the current feature vector. The controller agent then chooses the broker with the lowest classification score to use until the next update. In practice, we found a 15 second training duration to be sufficient to correctly classify broker performance reputation for varying levels of broker dependability (see section VI-D). During the execution phase, the question arises regarding how the broker distinguishes between poor broker performance and an overloaded network. We make a key assumption that network performance is highly correlated with broker performance. If there is any doubt, the controller or peer broker will see similarly degraded performance from other brokers upon associating with an alternative broker. Moreover, we impose a five second waiting period before changing broker associations, to minimize the amount of thrashing in controller/broker associations. Additionally, to provide for broker diversity, and avoid a situation where the most reliable broker becomes overloaded with domain requests, we limit the number of domains which may associate with a single broker to no more than ten. Future investigation could involve the study of optimal controllers per broker ratios in
terms of both broker and overall network performance relative to flow counts and domain counts.

**Algorithm 1** Broker Reputation Training algorithm (for controller $c_i$)

1: Train_Agents($L_{i1}, L_{i2}, \ldots, L_{ij}, B_i$)
2: for each broker $b_{ji}$ associated with controller $c_i$ do
3:   while time $t < \text{training\_duration}$ do
4:     $c_i$ implements broker $b_j$ flow rule changes
5:     $c_i$ records network state tuples $F_0 \ldots F_n$
6:   end while
7:   for each state tuple $F_i$ do
8:     classification value $v_i = \frac{(u_i - m_i)}{f_{si}} \times |C|$  
9:   end for
10:  for each training data group $G_i$ do
11:    train $c_i$’s LDA learning agent instance $L_{ij}$
12:  end for
13:  end for
14: Return $L_i$: the set of LDA learning agents for $c_i$

4.4.4 Market-Driven Brokering with Reputation

The FlowBroker framework, thus far, has been one in which brokers are cooperative with respect to inter-domain forwarding. As such this model does not capture a scenario in which brokers are competing for clients and deriving a reward for successful competition [103]. For a
subset of our simulation runs, we introduced a market-driven brokering mechanism in order to study the effects of market-based competition upon the overall system performance. The market-driven scenario [86], as with the cooperative brokering model, has two agent types; domain manager agents and broker agents. With respect to domain manager agents, they have an incentive to acquire better Service Level Agreements (SLAs) from broker agents by sharing more state information (e.g. an increase of frequency or type) and provide resources to guarantee connectivity between two adjacent ASes for the broker service in question. A better SLA creates more customers and more revenues for ASes, and the goal is to maximize achievable SLAs subject to the resources domain managers can make available for inter-domain broker for forwarding. AS managers provide intermediate link services between source/dest ASes with a given QoS level. As before, the AS manager agents choose brokers based upon reputational performance, leveraging Linear Discriminant Analysis to classify and choose reputation levels. The higher the reputation, the more expensive the service will be to subscribe to. Currency is network resources, and allowing that broker's flows to pass through the controller's AS subject to a minimum SLA guarantee. Broker Agents have an incentive to associate with and provide services for as many ASes as possible to maximize service revenues and retain customer ASes by providing maximal QoS and consistent reliability. Better QoS/SLAs are provided by maximizing network state updates and guaranteeing QoS levels through client ASes for inter-AS flow routes. Therefore the goal of the broker is to maximize customer subscriptions and QoS levels subject to the network state information they provide and resources they allocate for intra-domain flow forwarding. Figure 22 depicts the intra-domain network resource aggregation model used to simplify inter-domain forwarding, from the broker perspective, in both the cooperative and competitive brokering models.
There are two key differences between market-driven and cooperative FlowBroker, the first of which being the lack of cooperation in the market-driven approach. Each broker offers a distinct forwarding service and therefore must gather network state from multiple ASes and negotiate intra-domain path traversal agreements independently. The second difference is that our model assumes that the currency used is network resources, and that the higher the inter-domain QoS a traffic-originating AS requests from a broker, the higher the intra-domain QoS that the AS must provide for other traffic forwarded by that broker service. Network resources are no longer simply provisioned in a cooperative fashion, but rather must be paid for in exchange for other network resources. The negotiation workflow proceeds as follows:

• Step 1: Domain manager needs to forward a new flow to another domain, and interrogates its LDA-based broker rankings to find the best broker for this traffic,
• Step 2: The domain manager sends an AS_NEG_REQ message to request a forwarding path agreement from the chosen broker,
• Step 3: The broker responds with a BR_NEG_RESP message containing whether or not the broker has a path, and if so, what the cost is, in terms of intra-domain QoS guarantee, to receive inter-domain traffic forwarding services at the two QoS levels, and
• Step 4: The domain chooses the QoS level desired for forwarding its inter-domain flow, and agrees to provide the requisite intra-domain QoS for that broker’s other traffic flows.

For purposes of simplification, our model has a two-tiered cost structure, meaning that brokers may offer, and domain managers may purchase, forwarding services with two distinct QoS levels for throughput maximization: 40 Mbps intra-AS guarantee for low-quality 20 Mbps
inter-AS service, and 70 Mbps intra-AS guarantee for high-quality 35 Mbps inter-AS service. Though game theory shows that uniform maximal pricing results in maximal revenues for all providers [103], we vary the probability of AS managers choosing discount vs. full price service based upon a uniform probability distribution, provided the data rate of the flow allows for the low-end QoS level. The workflow for capacity allocation within domains is delegated to the domain controllers and enforced via link utilization monitoring. Domain controllers allocate capacity along dedicated intra-domain paths for each of the brokers with which they have contracted for forwarding services. Domain controllers monitor the data rates of the flows traversing these paths to make sure they fall within the agreed level of service. Flows which exceed the agreed data rate are throttled by queuing excess packets and if necessary outright dropping them. It is therefore in the best interest of both the broker, and the domain which originates the inter-domain flow, to adhere to the data rates agreed to originally.

![Figure 22 - Domain-based Brokering Abstraction](image-url)
4.5 Simulation Environment

The performance of the FlowBroker distributed controller architecture has been evaluated with the Mininet network emulation tool [104]. Mininet is a network emulator used to create a network of virtual links, switches, controllers and hosts on a single Linux OS instance. As we chose the OpenFlow SDN implementation to illustrate our multi-domain architecture, Mininet supports SDN controllers and switches, providing the capability to emulate hundreds of nodes and links on a single physical host machine. Mininet provides the ability to define custom topologies, leverage different SDN-compliant switches, and run various controller implementations. For our experiments, we chose to use the Floodlight [105] SDN controller. Floodlight is a Java-based controller implementation which is modularized and provides for ease of extensibility. Floodlight’s support for virtual and physical switches, along with its large community-based developer following, as well as the ease of modification by adding software modules, were key motivating factors for us to leverage this controller software for the implementation of the FlowBroker architecture.

With respect to the performance evaluation of the FlowBroker system, five distinct simulation scenarios were chosen to test both the load balancing efficiency of the load-balancing broker agents, as well as the reputation detection abilities of the controller-based learning agents. The five scenarios are described as follows, with parameters listed in Table 4:

- Scenario 1: Single Broker, 2 domains, 2 controllers
- Scenario 2: 4 Brokers, 5 domains, 5 controllers
- Scenario 3: 4 Brokers (one unreliable), 5 domains, 5 controllers
The initial simulation scenario, depicted in Figure 23, consists of two domains and a single broker. The link between switch 5 and switch 7 is the critical link in this scenario. Two traffic sources, one between Hosts 1 and 3 at 40Mbps and the other between Hosts 2 and 4 at 70Mbps, were used to test the broker’s ability to identify relevant global performance statistics and translate them into actionable instructions at the controller level. In this instance, the issue is that greater than
100Mbps of traffic is being pushed across a 100Mbps link resulting in catastrophic network load. It is the job of the broker to note this and quickly formulate a flow table update for controller 2 to forward its traffic from host 1 via the Switch3 ⇒ Switch4 link rather than the Switch3 ⇒ Switch5 link.

Figure 23 - Initial Two-Domain Simulation Topology

In simulation scenario 2, the intra-domain and inter-domain links each have a capacity of 100Mbps. Four TCP-based traffic flows are introduced to the network. Flows 1 and 2 have a source of domain 1 and destination of domain 5 and a data rate of 40Mbps. Flows 3 and 4 have a source of domain 2 and destination of domain 4 and a maximum data rate of 70Mbps. This scenario was devised to test the ability of the FlowBroker architecture to cope with multiple
controller-managed domains, multiple brokers, and the network state exchange between the peer brokers, in an effort to load balance inter-domain traffic between more than two domains.

The third scenario leverages the same topology as the previous, with the same traffic flow demands. However, we task broker 3 with ability to send flow rule recommendations which directly contradict the policy-based management goal of minimizing the maximum link utilization. Conflicting rules updates are generated by directing a controller to forward flow traffic over an already congested link, increasing the maximum link utilization or keeping it constant. LDA-based agents are introduced to controllers 2 and 5 to test whether they are able to
identify (categorize) broker 3’s policy decisions as unreliable, and identify an alternative broker to listen to.

The fourth scenario uses the same topology and traffic flow demands as the second and third scenarios, but is intended to test the overall impact of broker unreliability upon the entire network. All controllers are given one LDA learning agent per broker with which they associate. Each learning agent learns broker reputation, then categorizes current network state to determine whether to change brokers or not. We introduce a probability to determine whether each broker will give conflicting flow rule advice, and examine the maximum link utilization, average ETE delay and traffic loss level relative to this conflicting rule probability.

In the fifth scenario leverages the same topology and traffic demands as before, varying broker and flow counts to investigate the effects of a market-driven brokering model. Brokers no longer coordinate and exchange information, but rather are responsible for negotiating forwarding agreements with each respective domain. We investigate the effects of tiered service offerings upon the overall performance of the system, with respect to network availability, network latency, and network throughput.

The final section entails a comparative discussion of the FlowBroker traffic overhead with respect to the cooperative and the market-driven simulation models. We compare the effects of each brokering approach with respect to the overall traffic generated as well as the percentage of all traffic the overhead communication comprises.
4.6 Performance Evaluation

The performance of FlowBroker was evaluated with respect to five separate simulation scenarios captured in the previous section’s discussion. Five key metrics of interest were identified to determine the level network-wide performance in each scenario: traffic latency, maximum link utilization, packet loss ratio, network throughput, and network availability. Initially, a two-domain single broker scenario was used to evaluate the load-balancing broker base case of two domains, as well as examine the effects of varying the network state update frequency. Next, we used the multi-domain, multi-broker topology to test the ability of controllers to cope with multiple broker associations, as well as to test broker peering mechanisms such as distributed network state aggregation and exchange. The third scenario involves the introduction of trust-based learning agents where a single broker is configured to give policy updates which conflict with the goal of optimizing global network performance. The fourth scenario looks at assigning each broker a probability to determine how reliable its state updates will be, and evaluate how performant the FlowBroker architecture is in the face of increasing unreliability. The fifth scenario explores the notion of having brokers compete with one another for domain controller customers, having flow performance guarantees guide the this association as a means of currency for a market-driven brokering approach. Finally, we examine the traffic overhead which occurs with respect to both the cooperative and market-driven FlowBroker scenarios. Each simulation scenario was ran for 25 iterations, and the margin of error never exceeded more than +/- 1.89% of the mean. A description and analysis of the simulation results yielded by each scenario is discussed in the subsequent sections.
4.6.1 Two Domain, Single Broker Scenario

In [87], we present the initial simulation scenario, depicted in Figure 23, consisting of two domains and a single broker. This scenario was devised to test the feasibility of the broker architecture with respect managing two domains, and varying the frequency of updates. The link between Switch 5 ⇒ Switch 7 is the critical link in this scenario. Without the broker, Controller1 is unaware of the congestion in this link. Hence this scenario has high loss of traffic and inefficient utilization of under-utilized disjoint paths. However, when a broker is used to aggregate information and push rules down to each controller, load balancing is achieved and the max utilization of each link is minimized, as the critical link now only has a utilization 0.7, and the alternative path, which has a higher hop count is now being utilized, with a link utilization of 0.4 along that path.

Additionally, when varying the update frequency of network state between the domain controllers and their associated broker(s), Figure 26 and Figure 27 convey that a higher frequency of update is more advantageous to network performance. Specifically, average traffic loss observed with a 1 second update interval is roughly 3% where a 0.01s interval resulted in roughly less than 1% traffic lost on average. Moreover, average ETE delays were noticeably improved dropping from 0.24s for an update interval of 1s to 0.05s delay for an update interval of 0.01s.
Figure 25 - Two-domain simulation results for Link Utilization

Figure 26 - Two-domain simulation results for Traffic Loss per Update Interval
4.6.2 Multi-Domain Multi-Broker Scenario

In this second simulation scenario, depicted in Figure 24, we tested the ability of the FlowBroker architecture to scale by increasing both the number of controller-managed domains, and the number of brokers. Two TCP-based traffic flows (1 and 2) were created between domain 1 and domain 5 for a total maximum data rate of 40Mbps. Two other TCP-based traffic flows (3 and 4) were created between domains 2 and 4 for a total maximum data rate of 70Mbps. The default behavior of the multi-domain broker-less scenario is to largely forward traffic along the shortest path. This results in a domain1->domain2->domain5 path for flows1 and 2, whereas flows 3 and 4 follow the domain2->domain5->domain4 path, creating a critical link between domains 2 and 5.
Figure 28 conveys the maximum link utilization results with and without brokers. In the case without brokers, each controller essentially knows nothing of the state of the adjacent domains, and therefore blindly forwards traffic along the shortest path. The resulting maximum link utilization without brokers is roughly 0.98. This broker-less scenario also results in significant congestion and 5.6% average frame loss rate due to the critical inter-domain link of Switch11 ⇒ Switch14. However, when using the broker to coordinate among controllers and distribute global knowledge, efficient use of available capacity is achieved, reducing the maximum link utilization to the range of 0.71 to 0.82, depending upon the network state update interval.
Figure 29 - Simulation results for 5 domain scenario for Packet Loss Rate

Figure 30 - Simulation results for 5 domain scenario for Traffic Latency per Update Interval
With respect to the sensitivity to the update interval, we found that maximum link utilization was far more sensitive to an increase in the interval than were either the traffic loss ratio or the ETE delay metrics. We attribute this to the fact that an increase in link utilization does not necessarily lead to increases in delay or traffic loss. As the network state update interval increases, this naturally decreases the speed at which the brokers can both compute network state updates, as well as send load balancing recommendations. This delay results in an increase in the maximum link utilization due to the longer periods of higher traffic loads, and increased response time due to catastrophic utilization levels. There seems to be a point however, at an update interval amount of 0.2 seconds, where the rate of increase in traffic loss and delay increases more dramatically. This is attributed to the fact that total link saturation tends to occur more frequently for update intervals equal to or greater than this value.

Figure 31 - Broker count relative to domain count with respect to Maximum Link Utilization
Figure 32 - Broker count relative to domain count with respect to Traffic Latency

Figure 33 - Broker count relative to domain count with respect to Network Availability
Figure 34 - Broker count relative to domain count with respect to Network Throughput

Figure 35 - Broker count relative to domain count with respect to Packet Loss Ratio
In [87] we discuss the results in Figure 31 through Figure 35, which illustrate the relationship between increasing domain count, increasing broker count, and network performance in terms of maximum link utilization, average ETE delay, and observed traffic loss ratios. Each result is the average of ten simulation runs for each topological scenario. In each of the simulation runs we witnessed a clear performance enhancement with the increase in the number of brokers. Specifically, as the domain count increases from 6 to 10, the difference between utilizing 1 broker or 5 broker agents equals a 5 to 8% decrease in maximum link utilization, a 28 to 84% reduction in ETE delay, and 69 to 151% reduction in traffic loss. The increase in network performance is a direct result of increased broker responsiveness, in that the fewer domains a broker monitors, the more responsive it can be to load balancing and forwarding requests. This reduced management latency results in load-balancing and forwarding decisions which are better in sync with the current state of the network and its respective domains. Conversely, increasing the time between network state updates and policy-based decision making by broker agents results in performance degradation as seen in Figure 28 through Figure 30.

### 4.6.3 Broker Reputation Detection

In the third scenario, we use the same topology as the previous scenario to test the idea of broker reputation. Given broker 3 consistently gives flow rule recommendations which directly conflict with the load balancing policy to minimize the maximum link utilization, can controller 1, using an LDA-based learning agent, properly categorize this broker’s reputation and use a more reliable brokering source? Figure 36 conveys the average results of 25 simulations, where
controller 1 starts out listening to the flow rule advice of broker 1, and switches to Broker 3 at t=16s. The result is severe link congestion between switch 6 and 14. The LDA agent samples this performance and switches back to Broker 1, thereby relieving the congestion by leveraging the sw6->sw8 link for traffic between domains 2 and 4, per the advice of broker 1. When broker 3 was chosen by controller 2 at time t=16, the traffic loss ratio jumped from below 0.5% to around 5% and continued to grow until controller 2 re-associated with broker 1, implementing the broker’s flow rule modifications. This resulted in a prompt return to traffic loss ratios of around .005% at time t=22s, thereby mitigating link congestion and therefore traffic loss. Figure 37 illustrates the increase in traffic loss correlating with an increase in the network state update interval, and shows up to a 59% reduction in total traffic loss when leveraging LDA-based controller agents. Figure 38 similarly reveals up to a 16% reduction in ETE delay when LDA agents are allowed to categorize and change broker associations.

Figure 36 - Simulation results for the single conflicting broker scenario for Link Utilization
Figure 37 - Simulation results for the single conflicting broker scenario for Packet Loss Rate

Figure 38 - Simulation results for the single conflicting broker scenario for Traffic Latency
4.6.4 Performance Impact of Broker Reliability

In the fourth simulation scenario, we concerned ourselves with the effectiveness of using LDA-based learning agents within controllers. Specifically, we investigated just how much improvement would be yielded leveraging LDA to learn broker reputation relative to network state, using conflicting rule generation. Conflicting rule generation, is essentially the case when a broker proposes a rule to a controller that is not only sub-optimal in the load balancing sense, but would directly result in the very congestion that brokers and controllers are trying to avoid. By associating the current state of the network relative to the broker currently being used, we create a performance reputation and link it to the broker in question. To test the sensitivity of the learning agents to the reputation, we varied the probability that a single broker, for each controller, would provide conflicting rule suggestions, between 0.1 and 1, to determine the benefit of using LDA-based controller agents, relative to the traffic loss ratio, the average end-to-end delay, and the maximum link utilization.

Figure 39 - Broker reliability impact upon Packet Loss Rate
Figure 40 - Broker reliability impact upon Traffic Latency

Figure 41 - Broker reliability impact upon Maximum Link Utilization
When we take into account the probability of any given broker giving conflicting flow rule recommendations, or recommendations which conflict with the optimal load balancing policy, by recommending flow paths which result in link saturation, we see, in Figure 41, that the maximum link utilization for the scenario where controller agents do not use LDA varies from 75% to 98%. Alternatively, the scenario wherein controller agents leverage the LDA technique, taking past performance of brokers into account and learning from their classifications relative to current network state, maximum link utilization varies from 71% to 78% regardless of the probability of conflicting rules. Leveraging machine learning-based decision agents within the controller, in concert with a distributed load-balancing broker architecture, leads to significant improvement average end-to-end delay (from 0.45s to 0.22s), and traffic loss ratios (from 4.8% to 1.6%).
4.6.5 Market-Driven Brokering

In the final simulation scenario, we modify the FlowBroker architecture to allow for a market-driven model in which controllers, or domain managers, must negotiate with brokers to obtain their forwarding services, by providing a guaranteed level of service for the traffic that traverses that controller’s domain. In this scenario, both sides must agree upon a level of service for exchange of services; the controller must provide minimum capacity guarantees for the broker’s traffic across its domain, and the broker, in turn provides a minimum level of service to the traffic it forwards on behalf of the domain controller. Once the agreement is reached, the process of forwarding flow traffic from source to destination functions as described in Section 4.3.

![Figure 42 - Results for market-driven FlowBroker with respect to Network Availability](image-url)
Figure 43 - Results for market-driven FlowBroker with respect to Network Throughput

Figure 44 - Results for market-driven FlowBroker with respect to Traffic Latency
The results in above illustrate that while increasing broker counts still lead to an increase in overall network performance, with respect to network availability, network throughput and traffic latency, the magnitude of the improvement is less with the market-driven brokering.
approach than with the cooperative one. Specifically, Figure 42, Figure 43, and Figure 44 convey that for the market-driven brokering, network availability decreases, network throughput decreases, and traffic latency increases overall by 1.01x, 1.02x, and 1.10x respectively. This reduction in can be attributed to two factors. First, the very nature of market competition, and the need for domain managers to provision intra-domain path resources for specific broker traffic makes it harder for all competing flows to achieve the same maximal resource allocations. Secondly, there is an additional delay involved the negotiation of path data rates, and therefore this will slightly affect factors such as latency and availability. However, we did observe, that for those domains able to procure high-quality 35Mbps inter-domain paths, that those flows did see significant performance improvement over all flows in the cooperative model. Specifically, in Figure 45 we observed a 1.05x to 1.08x decrease in traffic latency for the subset of flows subscribed to premium service in the market driven model, over the cooperative model. Additionally, in Figure 46 the market-driven premium service yielded a 1.07 x to 1.18x decrease in packet loss over the cooperative model.

### 4.6.6 FlowBroker Overhead Analysis

In order to analyze traffic overhead incurred by the FlowBroker architecture, we measured the magnitude of traffic generated to maintain the FlowBroker architecture, including messages such as: NETWORK_STATE, GLOBAL_ROUTE_REQUEST, KEEP_ALIVE_REQ, FLOW_TABLE_MOD, KEEP_ALIVE_RESP, CTRL_KEEP_ALIVE, and CTRL_TABLE_BACKUP. For the market-driven approach, we also took into account AS_NEG_REQ and BR_NEG_RESP messages as well.
Figure 47 - FlowBroker overhead for Cooperative Brokering

Figure 48 - FlowBroker Overhead for Market-Driven competitive brokering
Figure 49 - Route Establishment Latency for Cooperative vs. Competitive FlowBroker approaches

With the market driven competitive brokering approach, we observed a 1.5x increase in the amount of overhead traffic generated by the FlowBroker architecture. This increase in overhead is attributed to both the need for additional message exchanges due to the broker/domain negotiation phase, as well as the fact that brokers no longer exchange information in a cooperative fashion, but rather compete for AS associations as clients. This competitive relationship necessitates each individual broker to gather the global routing information and forwarding guarantees required to provide each customer AS with reliable inter-domain forwarding. The additional traffic generated as a result of the lack of cooperation accounts for a large part of the additional overhead in the broker-based model. However, the increase in overhead due to the market-driven approach, the overhead traffic required to maintain state information updates and establish/maintain routes, even at a high update interval of 0.01s, never approached more than 0.097% of network capacity, in any given simulation scenario.
When analyzing the delay involved in broker establishment of inter-domain routes, we observed a 9% to 29% increase for market-driven brokers over the cooperative brokering model, as broker count increases. Moreover, we observed that for the cooperative model, route establishment delay slightly decreases or remains the same with increasing broker count. This outcome follows from the fact that as broker counts increase in the market-driven approach, more brokers are competing for the same number of domain customers, and the fewer customer agreements a broker has, the longer it will take to procure an inter-domain route for a potential new customer. For the cooperative model, the more brokers that are added, on average, the fewer domains a given broker must directly associate with to gather network state information, though after a certain point, too many broker processes could pose a scalability problem from a coordination perspective.

4.6.7 Heterogeneous Market-Driven Brokering

In previous works [87, 106] we introduced and analyzed the FlowBroker architecture as a multi-broker and multi-domain implementation, where homogeneous load-balancing brokers fully cooperate or compete with one another for distributed inter-domain forwarding, and domain controllers freely obtain forwarding services from any broker. In [107] we produce an enhanced FlowBroker architecture where heterogeneous market-driven brokers offer their inter-domain forwarding services to domain controllers in return for intra-domain transit for the flows coordinated by the brokers. Broker agents have incentives to maximize their rewards. Brokers still compete with each other for resources, rewards, and client ASes, but will measure their
success based on differing objective metrics. Specifically, our model enhances the FlowBroker architecture by introducing four distinct broker classes, each of which is labeled by their respective strategy, as shown in Table 5.

<table>
<thead>
<tr>
<th>Broker Type</th>
<th>Strategy Description</th>
<th>Strategy Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Maximization</td>
<td>Achieve maximal number of AS clients with decreased emphasis on revenue</td>
<td>Multiply the cost of all routes by a discount factor of 0.5.</td>
</tr>
<tr>
<td>Revenue Maximization</td>
<td>Capture maximal revenue from clients, with decreased emphasis on number of clients</td>
<td>All routes provided at full cost, no discount.</td>
</tr>
<tr>
<td>Packet Loss Minimization</td>
<td>Increase market share by minimizing packet loss, thereby improving performance reputation.</td>
<td>Set peering probability to 0.9 to encourage broker path diversity</td>
</tr>
<tr>
<td>Packet Latency Minimization</td>
<td>Increase share of clients by minimizing latency for flows and improve performance reputation.</td>
<td>Favor paths with minimum hop counts and per hop latency</td>
</tr>
</tbody>
</table>

Table 5 - Heterogeneous Market-Driven Broker Classifications

We examined the effects of broker heterogeneity in the market-driven architecture by examining three key performance metrics; market share per broker type, network availability per broker type, and route establishment latency per broker type. With respect to market share, Figure 50 conveys our studying the share of both overall revenues and client flow forwarding requests for all domain controllers and broker types in the network. Client maximization and revenue maximization have an inverse relationship when discounting is used. Specifically, client maximization brokers had the highest client share (31%) and lowest revenue share (18%), whereas revenue maximization brokers had the highest revenue share (29%) but lowest client share (21%).
Figure 50 - Heterogeneous Broker Market Share per Broker Type

Figure 51 - Heterogeneous Broker Competition with respect to Network Availability per Broker Type
From the perspective of the network availability, the Loss Minimization Brokers had the highest network availability under heavy load (98.8%) due to the increase path diversity resulting from a higher peering affinity, while the Client Maximization Brokers yielded the lowest network availability under heavy load (96.8%) resulting from the increase client count but decreased revenue, which resulted in decreased network resources for forwarding transit flows over client ASes. In regards to route establishment latency, the Revenue Maximization Brokers had the lowest route establishment latency (0.0982 s) due to the fact that fewer clients purchase non-discounted services, thereby increasing the availability of that broker’s existing resources for customers paying full price. Loss minimization brokers had the highest average route establishment latency (0.13 s) due to the increased likelihood that these broker types will peer with others, and therefore need to exchange network state information prior to route establishment.
4.7 Conclusion

In this chapter, we described a proposed framework for distributed multi-domain software-defined network management aided by market-driven brokers with respect to load balancing. We conducted experimentation both in the single-broker, two-domain instance, as well as in the case of multiple brokers and controller domains. We illustrated how multi-domain SDN can greatly benefit from a distributed brokering implementation where network state is aggregated upwards, and then policy recommendations are propagated back down to the individual controllers in the form of flow rule additions and modifications in the support of network load balancing and resource optimization. Specifically, we have shown that FlowBroker goes a long way towards minimizing the maximum link utilization in scenarios that would normally lead to link saturation and significant packet loss. Moreover, regardless of the broker update interval, FlowBroker yielded an average of 1.7x reduction in traffic loss and a 1.9x reduction in ETE delay over broker-less multi-domain deployments. Additionally, the introduction of machine learning agents within the controllers themselves greatly aided in the ability of each controller to determine the fitness of the flow-based policy updates which the brokers send, allowing for autonomy within the controller with respect to broker association and rule implementation, which still pursuing the global goal of network resource optimization through load balancing. Results showed that the LDA learning agents provided a 1.2x reduction in the maximum link utilization, a 1.1x to 2.0x reduction in average traffic latency, and a 1.2x to 2.5x reduction in the overall packet loss ratio, with varying the probability of incorrect policy updates. Moreover, when market-driven competitive brokering was introduced, we found a 1.5x increase in overhead versus the cooperative approach, but overall overhead traffic never
exceeded 0.097% of achievable network capacity. The FlowBroker research in this chapter is an initial investigation, and there remain more areas of interest to investigate.

With respect to multi-domain distributed brokering, network load balancing is one of many possible services that distributed brokers could provide. Other services could include energy-aware distribution of computation loads or storage-aware distributed replication schemes for failover and security purposes. Energy-aware applications of distributed multi-domain SDN brokering could be of particular use in heterogeneous multi-domain networks which take advantage wireless network deployments in certain domains, subject to both throughput and power constraints. Regarding storage-aware brokering service, both transport data and network management data can grow significantly over time due to replication. Network data deduplication mechanisms [108, 109] can be used within the context of traffic flow management to reduce the network load of state information exchanges in the multi-domain network scenario.
Chapter 5  Conclusion and Future Work

This dissertation makes key contributions in the area of architectural design and optimization of automated network control and management systems in the key areas of multi-hop wireless, hybrid wired/wire-lined and multi-domain SDN network deployment scenarios. This chapter serves to summarize those contributions contained within this dissertation.

In Chapter 2, we presented a case study the use of reinforcement learning to reduce the churn and overhead involved in maintaining stable clusterheads in multi-hop wireless network deployments. Specifically, by introducing combining Q-Learning with a weighted clustering algorithm with which we illustrated how leveraging nodal stability metrics in concert with reinforcement learning can lead to more stable cluster-head selections and therefore less overhead in maintaining clustering structures than standard weighted clustering. Specifically, we observed a 15-20% reduction in the number of cluster reformation, and a proportional decrease in communication overhead when machine learning was used.

In Chapter 3, we presented a closed-loop approach to tuning the routing layer protocol of a multi-hop wireless network in a heterogeneous network environment, taking network state into account to tune the HELLO Interval and Active Route Timeout parameters of the AODV routing protocol (AODV-Q). The simulation results derived from this approach were shown to improve the performance of the original Ad-Hoc On-Demand Distance Vector (AODV) protocol, reducing protocol overhead by 43% and end-to-end delay 29%. Moreover, the packet delivery ratio was increased by up to 11%.
In Chapter 4, we presented a framework for distributed multi-domain software-defined network management aided by market-driven brokers with respect to load balancing. We have shown that the FlowBroker architecture helps by minimizing the maximum link utilization in traffic scenarios that lead to deleterious capacity consumption and packet loss. FlowBroker achieved an average of 68% reduction in traffic loss and a 89% reduction in ETE delay over broker-less multi-domain simulation scenarios. The use of machine learning agents within the controllers themselves significantly improved the controller's ability to determine the whether the broker's flow-based policy updates were of value. Results conveyed that use of learning agents provided a 4% to 20% reduction in the maximum link utilization, a 10% to 98% reduction in average traffic latency, and a 21% to 155% reduction in the packet loss rate. When market-driven competitive brokering was examined, we did find a 47% increase in overhead compared to the cooperative brokering, however, the magnitude of the overhead traffic never surpassed 0.097% of the useable network capacity.

Future work in area of multi-domain multi-broker software defined networking will entail the modification of the FlowBroker architecture to support a fully dynamic inter-domain brokering pricing model, where the costs of inter-domain forwarding services fluctuate relative to the performance of the individual broker. We will study the impact of market-based competition upon this pricing model and overall network performance. Of additional interest is the degree to which domain controllers factor input from other controllers into the LDA-based performance-based reputation calculation and the degree to which domain controllers trust other domain controllers for this information, as well as for inter-domain forwarding tasks. We will also investigate the relationship between reputation and market-based brokering with respect to the social networking aspect of controller domains and the relationship between controllers and
brokers. The ability to quantify these relationships in terms of its impact upon market pricing, broker peering outcomes, and overall network performance is of key interest.
Bibliography


