Mining Historical Data towards Interference Management in Wireless SDNs

Maryam Karimi, Prashant Krishnamurthy, James Joshi, and David Tipper
University of Pittsburgh
135 N Bellefield Ave
Pittsburgh, PA 15260
mak322,prashk,jjoshi,dtipper@pitt.edu

ABSTRACT

WiFi networks are often planned to reduce interference through planning, macroscopic self-organization (e.g. channel switching) or network management. In this paper, we explore the use of historical data to automatically predict traffic bottlenecks and make rapid decisions in a wireless (WiFi-like) network on a smaller scale. This is now possible with software defined networks (SDN), whose controllers can have a global view of traffic flows in a network. Models such as classification trees can be used to quickly make decisions on how to manage network resources based on the quality needs, service level agreement or other criteria provided by a network administrator. The objective of this paper is to use data generated by simulation tools to see if such classification models can be developed and to evaluate their efficacy. For this purpose, extensive simulation data were collected and data mining techniques were then used to develop QoS prediction trees. Such trees can predict the maximum delay that results due to specific traffic situations with specific parameters. We evaluated these decision/classification trees by placing them in an SDN controller. OpenFlow cannot directly provide the necessary information for managing wireless networks so we used POX messenger to set up an agent on each AP for adjusting the network. Finally we explored the possibility of updating the tree using feedback that the controller receives from hosts. Our results show that such trees are effective and can be used to manage the network and decrease maximum packet delay.

KEYWORDS

SDN, WiFi, Resource Management, Classification Tree

ACM Reference format:
Maryam Karimi, Prashant Krishnamurthy, James Joshi, and David Tipper. 2017. Mining Historical Data towards Interference Management in Wireless SDNs. In *Proceedings of 20th ACM MSWiM 2017, Miami USA, November 2017 (MSWiM 2017),* 8 pages.
DOI:

1 INTRODUCTION

Wireless infrastructures play an important role in a growing number of environments, some of which, such as e-health care environments [5], are rather critical in their demand for specific Quality of Service (QoS) for specific flows. As an example, in hospitals, interference can impact the quality of a video which may be critical for a physician to make the correct diagnosis [1]. If the wireless network is controlled using a software defined network (SDN) controller, it is possible to achieve a complete view of the network in the controller which can monitor parameters. Through an observation of relevant parameters, the controller can manage changes to the characteristics of a flow (in the extreme case by changing the communication channel or stopping some flows).

This begs the question as to whether a specific combination of factors such as number of APs, location of APs, power, packet rate, packet size, number of flows and so on (specific value ranges) can be used to develop a model that can assess the emergence of QoS problems. In other words, is it possible to develop a model based on historical data that can automatically and rapidly assess the possible QoS in various situations when multiple APs are simultaneously sending data. Then, we may be able to use this model to decide whether the QoS is unacceptable and react by lowering (for instance) the packet size and packet rate of flows from other APs (based on the requirements and priority of specific flows). Since there are numerous parameter combinations, we consider building classification trees that allow quick decisions to be made by the controller. The purpose of this paper is to determine if this a viable approach. Our goal is not to develop new solutions to problems of flow control, scheduling, or congestion, but *to evaluate the potential of data mining based models in wireless SDN controllers* to automatically and rapidly impact QoS for specific flows. In order to see if such classification guidelines can be developed, different situations that might happen in the network were simulated under different configurations. After collecting various types of information about specific parameters, we created a *cleaned database* of scenarios and observed QoS. Using this historical data (in this paper from simulations), we applied data mining techniques using the Weka [13] data mining tool to create decision/prediction trees that can inform the controller what may happen to a flow under specific parameter ranges and network conditions.

We created a prediction tree that captures situations where there can be an observed decrease in QoS for flows and situations that may block communications. This "QoS tree" can be employed to predict delay based on QoS parameters (packet loss and delay) that a communication flow needs. Using the the QoS tree, an SDN controller in the network can detect situations causing interference and may, for example, switch the channel of the flow (or neighboring flows) or lower the specified bandwidth allocated to APs that do not require high QoS. We then applied the tree in a POX

MSWiM 2017, Miami USA

DOI:

[1] Interference can increase delay and packet losses which means a decrease in QoS.

[19] SDN controller. OpenFlow is sufficient for programming flow table rules but it cannot in general provide required information for wireless networks [29]. We used POX messenger and set up an agent on each AP. The controller used this new channel to receive information from APs, make a decision based on the QoS tree and to determine how it affected the QoS. As the historical data changes, we stored the leaves in an update-able data structure, making the tree dynamic. Agents were created on "Hosts" to communicate with the controller and send feedback to update the tree.

Our evaluation shows that such classification trees may be used to perform necessary management by the SDN controller. Using the dynamic data structure, we update the tree using online traffic. The results show that the tree is stable in the same network. We use the same approach when we have multiple flows and create decision trees for those configurations as well. Further work, when extended, may enable us to also determine which parameters are critical and need to be monitored more closely in the network to change (as needed) potential network slices/configuration, rather than managing only flows in one network.

The paper is organized as follows. In Section II, we provide a brief background of data mining, SDN, and some related work. In Section III, we describe the experimental design and present the QoS Tree. Section IV presents the results of applying the QoS tree in the SDN controller for simple situations with one flow. Section V provides similar experiments for more complicated situations with more flows and flow sizes. Section VI provides a discussion of limitations and future work and Section VII concludes the paper.

2 BACKGROUND AND PRELIMINARIES

In this section we review some basic concepts of data mining and SDNs. We also discuss some related work in this area.

2.1 Data Mining

Data mining includes four main steps to create knowledge from collected data: selection, pre-processing, data mining, and interpretation/evaluation. Selection is the process of choosing tuples and attributes that are required for answering questions. Data preprocessing includes the following actions: *Cleaning:* detecting and correcting or deleting inaccurate or corrupts records; *Normalization:* reduction of data to any kind of canonical form; *Transformation:* conversion of a set of data values into the data format of a destination data system; *Feature extraction:* deriving some attribute values from an initial set of measured data; and *Selection:* selecting a subset of relevant features based on a domain knowledge.

Data mining algorithms can be applied on the data to find patterns of interest. Classification and regression are considered important tasks in data mining. In this paper, based on our continuous class variables (delay), we chose Random Tree, REP and MP5 regression decision tree learners in Weka [13]. We also try to make the class variable "delay" categorical (> 100ms and < 100ms) and use J-48 classification algorithm. We chose these algorithms because they are popular with data mining researchers (e.g. [3, 7, 22, 24, 30]). These methods (described next) help to extract information relationships and hidden patterns in large data sets.

Random Tree: It is one tree from the set of possible trees, with k random features at each node [33]. The random tree generates many individual "learners". It constructs a decision tree by employing a random set of data. Each node is split using the best split comparing to other variables. At each splitting step all attributes are selected randomly and the tree is grown as much as possible [6].

REP: It is a fast decision/regression tree builder uses the regression tree logic to create multiple trees over different iterations. The algorithm uses a "gain" for splitting and pruning the tree by reduced error pruning and sorts numeric attributes. It uses the C4.5 method [2] of using fractional instances to deal with missing values. [12, 33].

M5P: M5P generates "M5 Model" trees and rules. M5 constructs a tree that relates the target value to other attributes using a divide-and-conquer method. First it computes the standard deviation of the target value in a node. Then it will consider all possible splits and calculate their standard deviations and the reduction in error of the parent node with that split. The maximum reduction in error will specify which split should happen. The algorithm stops when the number of tuples in the node reaches a specific threshold. Then it uses standard regression techniques to provide a linear model for tree nodes. It uses a greedy search to minimize the number of effective parameters by removing the variable that contributes only a little to the model. Finally it will prune the tree comparing the estimated error of each node with its parent node [23].

Following paths in random trees or REP trees will give us a result (by having specific parameter values). For example, a random tree or a REP tree may tell us: "if the packet size is smaller than N bits and the transmit power is smaller than P dBm, the delay will be t ms". In M5P trees, instead of getting a clear value as a flowchart result, we will have models left in the leaves.

J-48: The algorithm J-48 is a Weka implementation of the C4.5 classification algorithm with categorical class variable. We use it with two categories of delay as described previously [16].

We use 10-fold cross validation to test the created decision trees. The data set was split into ten equal size subsets. Nine subsets are used to train the model, and the model is tested with the remaining tenth subset. The number of correct classifications over the number of all instances is used to estimate the accuracy of the tree [16].

2.2 Managing interference by using machine learning algorithms

Machine learning algorithms have been used in wireless interference management, some of which we briefly review next. In [10] authors process data from real-time reporting of sessions for network optimization. In order to predict packet drops, before the end of the session, machine learning was used on offline LTE data. In [18], authors identify interference modulation order by using source automatic modulation classification. They use supervised learning techniques to achieve channel estimate in inter/intra cell interface, with/ without accurate information. This method can be used in cancellation of interference in cellular networks.

In [31], authors optimize radio resources with poor performance by using a statistical learning process which uses regression to

[2] C4.5 is an algorithm that splits data into smaller subsets by calculating "entropy" (the measure of data disorderliness) and "gain" (decrease in information entropy) for possible attribute splits, and makes a decision. For each split it chooses the highest gain that is the lowest entropy to branch on. It stops when it reaches a completely pure subset that all instances have the same class attribute in a tree leaf. Then the tree will be pruned to eliminate outliers [6, 16]

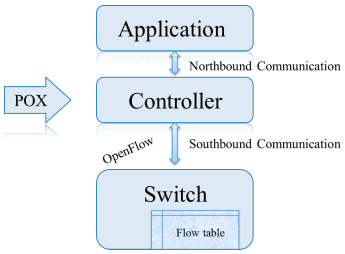

Figure 1: Software Defined Network architecture

extract relation between performances attributes. The objective is to heal inter-cell interference coordination. In [17] a Kalman-Filter approach is used to predict interference by deriving the correlation of co-channel interference. Based on interference prediction and path gain, the transmit power can be adjusted to achieve the required signal-to-interference ratio (SINR). None of these works have considered SDN networks and their control as their objective.

In [10] and [18], authors use historical data and libSVM to create models which predict interference but they did not apply their model to the network to see how it can improve performance metrics. In [31] and [17] they apply their model in the network but they did not use historical data. In these papers, authors consider signal-to-noise ratio and none of them examine QoS. In this paper we simulate parameters that affect QoS and based on data from all APs across the network, we use models that decide how interference can affect the QoS. We also apply the models in an SDN controller to react to network conditions by lowering the packet size and packet rates of flows from APs with lower priority to improve QoS.

2.3 Software Defined Network

SDNs push the control plane of the switches and routers to software. The data plane in SDNs is separated from the control plane. The high-level architecture is shown in Figure 1. The central controller in the SDN architecture provides the infrastructure for managing the network. Routing algorithms are placed in the controller. The SDN controller receives policies and instructions from the "application" via north-bound communications. [27]. Routing is performed for each flow by the controller and installed rules in the switch's flow tables. Switches forward the data according to these rules. When a flow enters a switch, the switch compares flow fields with the flow table. If it matches an existing entry, the corresponding action will be taken; otherwise the switch uses the OpenFlow protocol to send the first packet of the flow to the controller. The controller then calculates the route for this flow and adds an entry with flow fields and suitable action to the flow table. SDN provides an intelligent and controllable architecture, less dependency on hardware or specific vendor, simple management, faster innovation, implementation, and testing [1, 27].

SDNs may be used in a variety of environments. As an example, consider a healthcare application where it is required to stream approximately 360 Mbps uncompressed video from two discrete endoscopic cameras [26]. Processing this data needs a high performance real-time computing (HPC) environment, to minimize the risk to a patient. In [26], authors utilize an algorithm on OpenFlow SDN to use its capability of connecting multiple remote HPC servers and medical devices. Similarly, the use of SDN in wireless networks

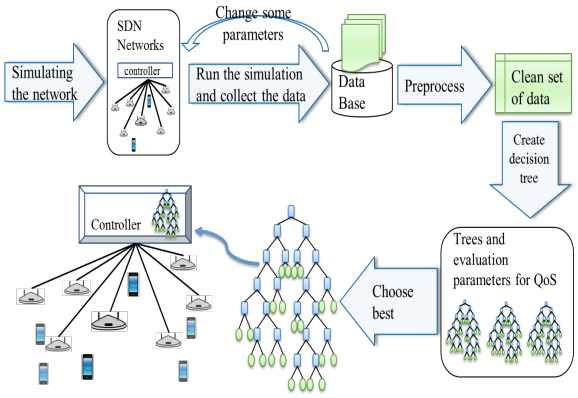

Figure 2: Steps in experiment

is possible. In particular, an SDN controller can set parameters in WiFi APs (which are the switches in Figure 3) such as the channel and the transmit power in addition to the flow tables [4].

In [29], authors claim that there is not any uniformity of feature set solution available for wireless networks management, and OpenFlow does not address WiFi complexities such as interference management, mobility and channel selection. They used Odin to propose Light Virtual Access Points (LVAP) which is per client AP with unique BSSID (mac address of wireless interface), it provides isolation in control logic. In the case of handoff these LVAPs migrate between APs without triggering re-association in clients. Some applications were developed over Odin, such as: mobility manager, load balancer, trouble shooting (Interference and jammer detection using channel snapshots using WiFiNet cards), channel selection and energy efficient WiFi networks (by selecting one AP as master with couple of APs as slaves) and guest policy enforcement [29].

3 EXPERIMENTAL DESIGN AND RESULTS

We simulate an SDN in which some hosts (2 to 6 in number) are communicating with each other (1 to 3 flows) through an AP and neighboring APs cause interference. We then examine different situations by changing the number of interfering APs, their power, packet sizes, packet rates, their locations, different numbers of hosts, and different flow sizes. Then we use WEKA to apply data mining methods (specifically the random tree, REP, and M5P) to create a decision/prediction tree that considers the current state and predicts the QoS of the tagged flow for that state. Each state is composed of the above parameters - number of APs, APs' locations, packet size, packet rate, power and etc. QoS is defined as the maximum delay that can happen in each state and it will be compared to the delay that the flow can tolerate based on the SLA. Based on the QoS the flow needs, the prediction tree (described later) can be used to predict whether a situation can provide the necessary QoS or not. Figure 2 shows all the steps used in our experiments.

3.1 Test Scenario

We first run the controller and create the WiFi network with the topology in Figure 3. Hosts (in this case h1 and h2) start to communicate while other WiFi APs continue to broadcast packets. This process is explained in more details next.

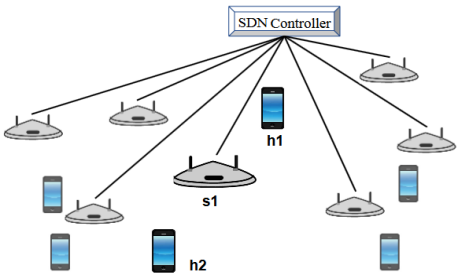

Figure 3: Experiment's topology

Experimental environment: Experiments are done with Ubuntu 14.04, python 2.7.6, java 1.8.0_111. Tools that were used include but were not limited to OpenNet SDN simulator, POX controller, Weka, eclipse 3.8.1 and pydev.

Network setup: For each experiment, we first ran the POX [19] SDN controller, then OpenNet is started. OpenNet [8] is an open source simulator for wireless SDN formed by two simulators: Mininet [20] for simulating the SDN, using OpenFlow 1.3.1 and NS3 [25] for simulating wireless networks. We used OpenNet, without any change [3], to simulate 420 different configurations with a different number of APs (1 to 21) in different locations. In this simulation `ns3::YansWifiPhyHelper` is used to set-up WiFi PHY in the emulation, which uses `ns3::LogDistancePropagationLossModel` as the propagation loss model and `ns3::NistErrorRateModel` as the error rate model. The received power after adding the propagation loss is calculated as:

$$rx = 10\log(Pr_0(tx)) - n \times 10\log(d/d_0)[14] \quad (1)$$

in which n is the path loss distance exponent, d_0 is reference distance (m), L_0 is path loss at reference distance (dB), d is distance (m), Pr_0 is the received power at d_0 (W), and tx is the current transmission power(dB) [14]. Different modulation/coding schemes have different error rate models [4].

In Figure 3, the lighter/smaller APs are changed in numbers, locations, power, etc. with each configuration. We vary the number of APs from 1 to 21. The number of hosts vary between 2 and 6. APs and hosts are placed in a rectangular region between local coordinates of −120 m (lower / left) to 120 m (upper / right). The Other ranges are as the following: transmit power range between 0 and 40 dBm, packet size range from 0 to 100000 bytes and packet rate was between 10 and 1000 packets per second. Packet size ranges from 64 bytes to 4000 bytes. All APs use channel 11 and simply broadcast packets to influence the QoS of the tagged flow(s). The APs were connected to a POX controller. Each configuration was defined in a python script. Each python script defines the AP's position, links, host's position, host's mobility, the channel characteristics and other details about the simulation. Default parameters are used in most cases. As mentioned previously, our objective was not to develop any new algorithms for solving specific network problems, but to evaluate the feasibility of using data mining for network resource management in an automated manner.

[3] The simulated network models the IEEE 802.11g standard.

[4] Validation and description for OFDM modulation is presented in [21]. It calculates bit error rate (BER) for different modulations such as QAM, BPSK, QPSK at giver SNR after and before applying Forward Error Correction (FEC) [15].

Running experiments: Hosts **h1** and **h2** send packets with variable sizes to each other via the AP **s1**. The other APs and hosts may be there or not in different tests. In each experiment, the transmit power of each AP is specified. Hping3 [28] is used in each AP to create and send TCP/IP packets. This provides us the possibility of specifying different packet rates, packet sizes, packet counts and other protocol details and varying them easily. The interference from these transmissions influences the delay, packet loss and thus the QoS of the tagged flow. There are 1 to 3 tagged flows with different sizes that we monitor in the experiments, which is between hosts **h1** and **h2** that passes through AP **s1**. For each flow, we sent 50 packets to be able to see the changes.

Collecting the results: In each experiment some outputs were gathered to form the database used as historical data. This database was analyzed to drive the prediction/decision trees (explained later). In each experiment the following files were stored: (a) python files of the simulated network consisting of APs and hosts' locations, (b) a script for setting the transmit power and executing Hping3, containing power, packet size, packet rate and number of packets, and TCP/UDP mode. Table 1 shows all of the extracted attributes.

In some cases, we decided to use the option "flood", which sends as many packets as it can with the maximum possible rate. The output response of every single packet delivery (between hosts) was stored along with the packet size, overall max delay, min delay, average delay and packet loss. The results show that the delay range can be between 4.08 ms and 50672 ms. This experiment was repeated 1480 times with different configurations with different numbers of APs, packet size, packet rate, AP locations, sending power, flows and other variables. The total amount of gathered data was about 50 Gigabytes and the attributes in each file were surrounded by many unnecessary data fields. Thus the collected data needed to be pre-processed.

3.2 Pre-processing

For data pre-processing, we wrote Java programs to apply some string processing to extract features from the hosts' files and compute flow size, meanDelay, maxDelay, minDelay and packet loss (see the tuple in Table 1). Next, we processed the APs' files to determine the power and Hping3 command parameters. Then we processed the network simulator python files to get the APs and hosts locations. The result is a summarization and integration of each experiment into one file. Later, another Java program was developed to integrate all files into one single excel file. We further pre-processed the data by data cleaning (deleting some records in which the APs stop working), transformation and normalization (to bring data into an acceptable range[5]), feature extraction (APs distances from hosts and s1 considering their location) and selection[6]. The result was a table with more than 200,000 tuples.

[5] Some parameters like power should be in an acceptable range. The maximum possible transmitter output power in most devices is 30 dbm and there is no legal device that can support a transmit power more than 40 dbm [9, 11].

[6] Several data cleaning operations are not discussed here for lack of space. For example, floods were replaced with the maximum possible rate and size. Some standard techniques were applied to create better trees - the mean size and rate were multiplied by the number of Sender APs and the sum of the distance to **h1**, **h2** and AP **s1** were divided by the number of sender APs. Where some records are important and they should not get pruned, were duplicated. Also, since we have only one class variable, which is delay, when packet loss occurs we assume the packet's delay was more than a threshold, so we set the maximum delay in that record to that threshold.

Table 1: Fields and Description

Fields	Description
Mean Delay	Average delay between h1 and h2
Max Delay	Maximum delay between h1 and h2
Min Delay	Minimum delay between h1 and h2
Packet loss	Number of lost packet between h1 and h2
Num Of Ss	Number of APs that are sending packets
MeanDist Ss sH	Average distance between APs & Sender Host
MeanDist Ss rH	Average distance between APs & Receiver Host
Mean Dist Ss&S	The average distance between APs and AP1
TPacket Size	Total size of the packets that APs sent
TPacket Rate	Total rates at which APs sent packets
Mean Ss Power	The average of the power of sender APs
Num of flows	Number of Hosts that send packet
Packet size	The size of packets sent by the considering host
Total flow size	Summation of all flows in the network

4 SIMPLE DECISION TREE FOR ONE FLOW

We build a simple tree based on some part of the data to see the effectiveness of our method. We put this tree in the controller and create sufficient agents for APs to communicate with the controller and change their bandwidth usage based on the prediction tree.

4.1 Creating the tree

We used Weka [13] to build decision trees to decide whether a configuration may affect the QoS. First we built the tree for tuples that contained only a single flow of 64 bytes (between **h1** and **h2**), and we aggregated all packets of one flow to one tuple by considering the maximum delay as its class variable. If the tree predicted a block or very high delay in the flow, the controller should react to it by lowering the data rates of other APs. The attributes that were used are as follows:

QoS Tree
numOfSenders
meanDistOfSendersToH1
meanDistOfSendersToH2
meanDistOfSendersToS
meanPacketSize
meanPacketRate
meanPowerOfSenders

The class attribute was “maxDelay” for the QoS tree. The attributes are used to split the tree branches in each step and the class attribute is used in the leaf nodes as the result for subsequently predicting the delay for packets in the tagged flow.

Since the class parameters were continuous, we needed regression classification algorithms and in Weka, we used “M5P”, “REP” and “Random Tree”. The correctness of the trees was checked with 10 fold cross validation (see Section II). The summary of QoS prediction trees are shown in Table II. For each tree in Weka, the correlation coefficient is calculated to estimate the effectiveness and correctness of the tree. There is the exact value of variables and the value that the model estimated for that variable; The correlation coefficient indicates how much these two variables are related [2].

Considering Table 2, comparing correlation coefficients, it turns out that random tree has the highest correlation (0.9709). So we believe it can be chosen as the representative tree to predict QoS.

Table 2: Evaluation of QoS trees

QoS tree	M5P	REP	Random
Correlation coefficient	0.9303	0.968	0.9709

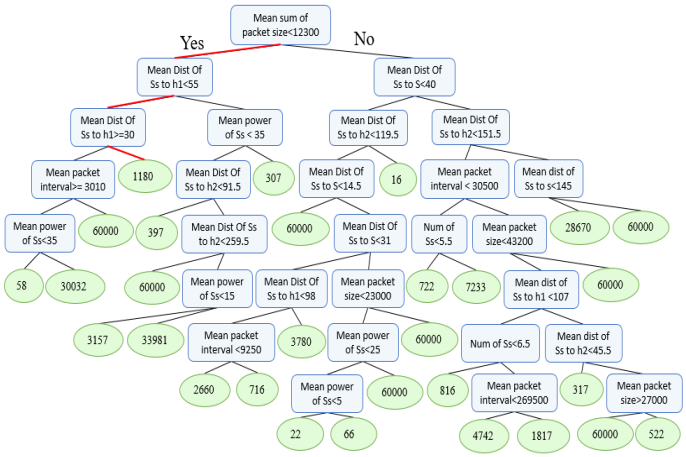

Figure 4: The selected decision tree

In the QoS tree, by following the trees' flowcharts, the delay for the flow in a specific configuration with specific parameters can be predicted. The tree is shown in Figure 4. As an example, in the Random Tree for QoS, if the packet size is less than 12300 bytes and the mean distance to **h1** is less than 30 m, then the maximum delay that can result will be about 1180 ms. As shown in Figure 4, in each branch of the tree, just some of the parameters are needed. This allows the controller to quickly make decisions when necessary.

4.2 Measuring the effectiveness of the tree

In order to see the effectiveness of the tree, it is embedded in the SDN controller to predict network performance and apply necessary topology changes to see how this will impact the network performance in practice. It is necessary to determine the importance of various parameters in prediction to minimize the monitoring overhead. Weka created the decision tree for us. We wrote a program to convert that tree into “if and else” conditions to aggregate it into the POX controller. POX uses the port 6633 to communicate with APs, but this communication is used for OpenFlow information. This is not enough for wireless networks and management and it cannot receive all required information. To provide a communication mechanism between the controller and APs, we used POX messenger, which uses the port 7790 to receive information from APs. Then we created a Python agent on each AP which is responsible for communicating with the POX messenger. Agents build a message consisting of location information, packet rate, packet size and power and then send this message to the controller. Agents repeat this action each 100rtt (round trip time). The controller receives the information and uses the QoS tree to decide whether the required QoS is provided or not. If the maximum delay was more than acceptable for the tagged flow, it sends APs suitable commands to reduce the rate by a factor of 10 and size of the packet by half. On the other hand, if there was extra bandwidth, APs may increase their packet size by 100 kilobytes and also increase packet rate by a factor of 10. The agent on AP will hear that and apply changes.

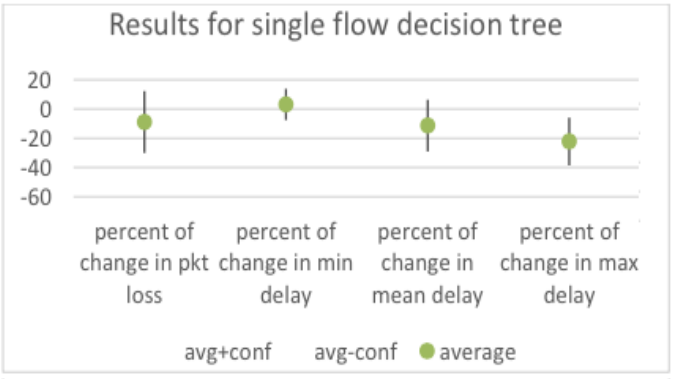

Figure 5: Results for single flow decision tree

4.3 Results

Considering the amount of information that should be exchanged among the controller and APs, we repeated the experiment in 12 different situations, both before and after applying the QoS tree to see how it affects the QoS. The results are summarized in Figure 5. The percentage of changes in mean delay, max delay and packet loss are negative which shows a reduction that concludes a higher QoS. There is a small increase in minimum delay in some cases due to the additional control packet and processing time in the controller. We calculated 95% confidence interval for each parameter.

In Figure 5 the dots below the zero line shows a reduction in delay or loss. Considering the confidence interval, there was no big change in the case of minimum delay (a little increase due to extra communication between AP and controller), average delay or packet loss. But the amount of maximum delay is decreased considerably, which is very good and beneficial especially in real time applications or communications.

5 DECISION TREES FOR MORE FLOWS

In this section we used the entire database including the experiments with more than two hosts which are sending packets (multiple flows). Then we built different trees based on the whole database using REP, M5P, Random Tree and J48 algorithms.

We first built a tree for delay. In this tree we can have more than one flow and the size of the flow can also change. In spite of the previous decision trees in this section, each packet of the flow formed a tuple, while in the previous one, each flow were summarized into one tuple. For each tuple some of the other flow sizes are also added. The attributes that were used are as follows:

QoS Tree
meanDistOfSendersToReceiver Host
meanDistOfSendersToSenderHost
totalPacketSize
totalPacketRate
meanPowerOfSenders
sumOfFlowSize

As described in the following sections, we built 3 different trees using these attributes and different class variables. We first built a regression tree for delay and later we built classification trees.

Table 3: Evaluation of All Data Delay Decision Tree

QoS	Linear regression	REP Tree	M5P Tree	Random Tree
Correlation coefficient	0.39	0.804	0.795	0.80

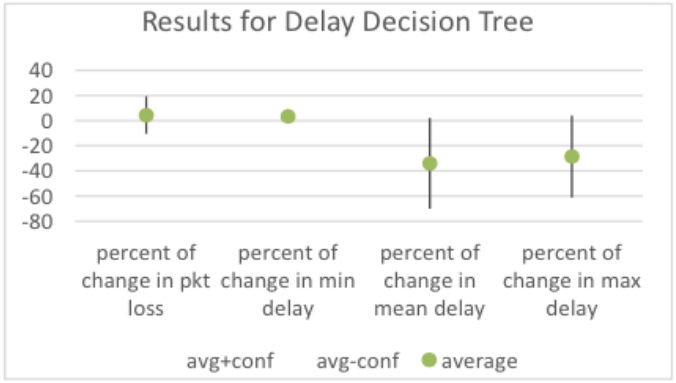

Figure 6: Results for delay decision tree

5.1 Regression Tree for Delay

Here the class variable was delay (shown as "delayh" in Figure 6). The total number of instances in the data set was 206261. We do not show the tree due to space limitation (the size of the tree was large with 869 nodes). Table 3 shows the output of 10 fold cross validation on the tree. The important variables in the REP tree were distance of senders to receiver, power and packet size. We also compare the decision tree methods with linear regression.

As we did in the previous section, we placed this tree in the controller and created agents on APs. We also need to create some agents on hosts to send their packet sizes. We tested this tree in 12 different configurations and for 3 hosts which gives us 36 tuples (36 experiments) and we summarized the results. The results are shown in Figure 6. Like previous results, the maximum and mean delay and packet loss are all reduced while there is an increase in min delay in some cases due to the additional control packet and processing time in the controller. The dots below the zero line show a reduction in the delay or loss. The amount of average delay and maximum delay is decreased but based on confidence interval, this tree was not as effective since the confidence range crossed zero.

5.2 Delay Classification tree

In order to apply classification trees we changed the class variable by setting a threshold of 100ms (requirement for time sensitive applications [32]) for delay to label the class variable, considering the delay is more (false) or less (true) than this value. Then we built the tree using REP, Random Tree and J48 algorithms in Weka.

The output of 10 fold cross validation on the tree is shown in Table 4. We also apply libSVM [10, 18] on our data to create the model and compare it with classification trees. The time it takes to build a tree for libSVM was two days, while it takes few minutes to build a tree with decision tree algorithms.

We chose J48 prediction tree since it has higher overall performance; REP has the lowest correctly classified items, libSVM was so slow (it takes two days to build the model) while it doesn't have

Table 4: Evaluation of Classification Delay Tree

QoS	REP Tree	J48 Tree	Random Tree	libSVM
Correctly Classified Instances	82.95%	83%	83%	83%
Tree size	1709	1159	6541	---

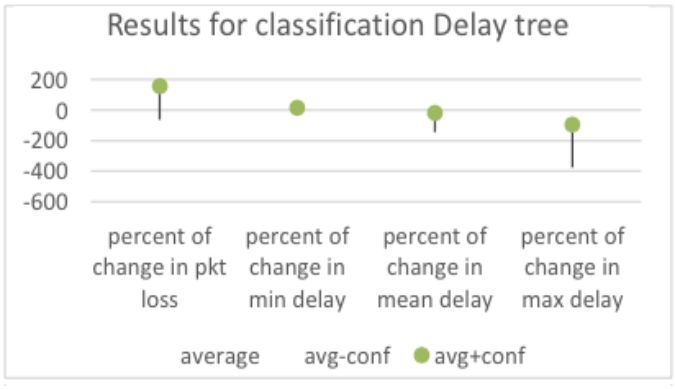

Figure 7: Results for delay decision tree

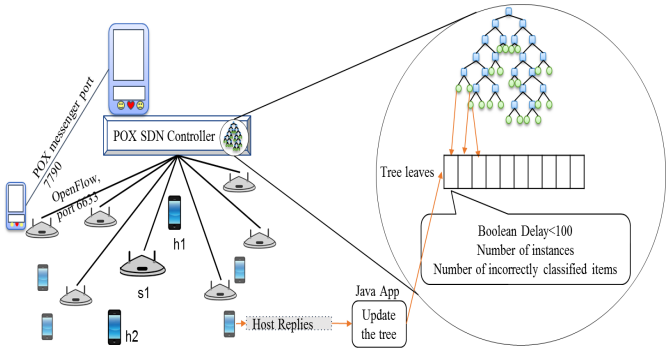

Figure 8: Dynamic tree

higher correctly classified items. Random Tree has a tree 6 times bigger than J48 with the same correctly classified items. So in this case we pick J48 decision tree as our delay decision tree. The important variables in the J48 derived tree were packet size, distance of senders to receiver, and transmit power.We placed this tree in the controller as well and repeated the experiment as in previous sections. The results are shown in Figure 7. Although there is a slight increase in minimum delay due to sending control packets, we can see that the tree was much more effective and the average delay and maximum delay were both decreased. It especially has a big effect on decreasing the maximum delay. Although there is a slight increase in minimum delay due to sending control packets, you can see that the tree was much more effective and the average delay and maximum delay were both decreased. It especially has a big effect on decreasing the maximum delay.

5.3 Dynamic Tree

In order to update the tree by receiving feedback from hosts, each leaf is tagged with the value of the class attribute, the number of instances and the number of incorrectly classified instances in that node. Then we store the conditions related to sequence number and leaves values in an update-able data structure (e.g., array).

We put this tree in the controller and create proper agents on APs and Hosts to communicate with the controller and update the tree as new training instances get available as feedback from hosts. APs send their attributes such as location, packet rate, packet size, power, etc. The controller reads values related to the condition from that data structure and replies with whether they should decrease their packet rate and packet size or not. Then the controller also receives feedback from the hosts to see how it affects the delay then it updates the tree using that feedback. The feedback is the number of instances and the delay values. In the dataset the controller adds up the number of instances in one group and decides whether the class tag for that leaf should change. Based on the requirement the coefficient for old data (instances that are already in tree leaves) and new data (feedback from hosts) can be different.

We try this on the same network to see how it changes the tree, the results show that in 19% of the times the statistics in tree leaves may change but the labels of the leaves do not change. This provides us with a reasonable confidence in teh stability of the trees and their ability to allow the SDN controller to manage the network.

6 DISCUSSION AND LIMITATIONS

Our objective in this work was to examine the potential of mining historical data towards wireless SDN management. SDNs aim to provide compatibility among different networks. In considering complex situations (e.g. in hospitals with different applications such as multimedia streaming, internet of things, medical devices, and personal area networks) we believe a good, yet simple, model based on historical data may be a great help in managing the network configuration, monitoring, troubleshooting, modifying and optimizing the network. SDNs can be applied in a large scale environment – management will be hard and it is necessary to make it automatic to reduce the amount of effort for network management. Providing a reasonably accurate yet simple approach for SDN management is not trivial. Many different machine learning and artificial intelligence approaches have been applied in many applications, such as supervised learning (e.g. classification), unsupervised learning, and reinforcement learning (e.g. evolution and swarm algorithms and neural networks). In this work we examined the applicability of simple models using historical data for automating wireless SDN management. We applied the models in the network to see whether there is an improvement in performance and also use feedback to update the decision trees. Previous works use historical data to create models but do not test the models in experiments nor do they get feedback from online data.

In this paper we focus on constructing a simple model with decision tree regression and classification techniques. Our reasoning is that trees allow an SDN controller to quickly check the important conditions for reconfiguration to see if the performance metrics (SLA) can be satisfied. This is in contrast to computationally expensive approaches that try to optimize the network performance. Our goal instead is to simply meet the performance metrics. We were also interested to make the tree more dynamic and use the feedback from hosts to optimize and update the tree and see the results over a longer duration of time. Decision trees may change in different networks with different configuration and different topology. Also multiple channels may need multiple trees.

A problem that is not covered in this paper and planned for the future is to consider mobility and other complexities, more diverse data sets and determining how much historical data is required for it to be effective in dynamic wireless networks. Clearly, this work is nascent and does not address several complex issues. In the simplest limitation, while considering mobile nodes that need heterogeneous QoS metrics to be satisfied, the decision trees may change substantially. This work also leads to hope for exploring automatic bandwidth management, potential of using such models in dynamically slicing the network into partitions and network reconfiguration. We can potentially use these kinds of models to automatically manage the network i.e., bandwidth allocation or slicing the network. Using real world data sets which are not available at this time can also help us to confirm the results. Creating an optimal tree and evaluating the scalability and applying other dynamic tree approaches to update the model as new training samples are available are possible solutions to improve the models.

7 CONCLUSION

In this paper we used historical data to predict quality of service and decide what flows to throttle (e.g. reducing packet size) towards managing an SDN wireless network where interference from competing transmissions may impact the quality observed by critical flows. In order to create a database consisting of different network configurations and traffic situations, we first simulated many SDNs with different topologies. In each topology, some hosts try to connect via an AP while other active APs cause interference, decrease the flow QoS and may block the media by overusing the bandwidth or sending with high transmit power. We collected the data and pre-processed them to achieve a clean set of data that consists of the number of actively interfering APs, APs' location, packet size, power, delay, etc. The data mining tool Weka is used to apply data mining methods to create one tree for prediction of the QoS. Based on the QoS prediction tree, the SDN controller can decide whether a situation can meet the demanded QoS or not. The Results show that such trees can be used by SDN controllers to rapidly managing the network to maintain QoS for critical flows. The controller can also use feedback from hosts to update the tree on a continual basis to improve the delay performance.

REFERENCES

[1] Adnan Akhunzada, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Muhammad Imran, and Sghaier Guizani. 2015. Securing software defined networks: taxonomy, requirements, and open issues. *IEEE Communications Magazine* 53, 4 (2015), 36–44.

[2] the University of Waikato Albert Bifet. 2012. Regression. (2012). http://www.cs.waikato.ac.nz/~abifet/523/Regression-Slides.pdf

[3] Gaurav Ambekar, Tushar Chikane, Shiben Sheth, Abhilasha Sable, and Kranti Ghag. 2015. Anticipation of winning probability in poker using data mining. In *Computer, Communication and Control (IC4), 2015 International Conference on*. IEEE, 1–6.

[4] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. 2012. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 109–114.

[5] Petros Belsis, Dimitris Vassis, Stefanos Gritzalis, and Christos Skourlas. 2009. W-ehr: a wireless distributed framework for secure dissemination of electronic healthcare records. In *Systems, Signals and Image Processing, 2009. IWSSIP 2009. 16th International Conference on*. IEEE, 1–4.

[6] Lakshmi Devasena C. 2014. Comparative Analysis of Random Forest, REP Tree and J48 Classifiers for Credit Risk Prediction. *International Journal of Computer Applications (0975 -8887), International Conference on Communication, Computing and Information Technology (ICCCMIT-2014)* (2014).

[7] Lakshmi Devasena C. 2015. Article: Comparative Analysis of Random Forest, REP Tree and J48 Classifiers for Credit Risk Prediction. *IJCA Proceedings on International Conference on Communication, Computing and Information Technology* ICCCMIT 2014, 3 (March 2015), 30–36. Full text available.

[8] Min-Cheng Chan, Chien Chen, Jun-Xian Huang, Ted Kuo, Li-Hsing Yen, and Chien-Chao Tseng. 2014. OpenNet: A simulator for software-defined wireless local area network. In *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*. IEEE, 3332–3336.

[9] Ronald H Coase. 2013. The federal communications commission. *The Journal of Law and Economics* 56, 4 (2013), 879–915.

[10] Bálint Daróczy, Péter Vaderna, and András Benczúr. 2015. Machine learning based session drop prediction in LTE networks and its SON aspects. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*. IEEE, 1–5.

[11] Shyamnath Gollakota, Haitham Hassanieh, Benjamin Ransford, Dina Katabi, and Kevin Fu. 2011. They can hear your heartbeats: non-invasive security for implantable medical devices. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 2–13.

[12] Mihaela Göndör and Vasile Paul Bresfelean. 2012. REPTree and M5P for measuring fiscal policy influences on the Romanian Capital Market during 2003–2010. *International Journal of Mathematics and Computers in Stimulation* 6, 4 (2012), 378–386.

[13] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.

[14] Tom Henderson. 2015. ns3::LogDistancePropagationLossModel. (2015). https://www.nsnam.org/docs/release/3.22/doxygen/classns3_1_1_log_distance_propagation_loss_model.html

[15] Tom Henderson. 2015. ns3::NistErrorRateModel. (2015). https://www.nsnam.org/docs/release/3.22/doxygen/classns3_1_1_nist_error_rate_model.html

[16] Maryam Karimi and M Ahmadzadeh. 2014. Mining RoboCup Log Files to Predict Own and Opponent Action. *International Journal of Advanced Research in Computer Science (IJARCS)* 5, 6 (2014), 27–32.

[17] Kin K Leung. 2002. Power control by interference prediction for broadband wireless packet networks. *IEEE Transactions on Wireless Communications* 1, 2 (2002), 256–265.

[18] Tze-Ping Low and Jangwook Moon. 2015. Interference Modulation Order Detection with Supervised Learning for LTE Interference Cancellation. In *Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd*. IEEE, 1–5.

[19] Murphy McCauley. 2017. POX Wiki. (2017). https://openflow.stanford.edu/display/ONL/POX+Wiki

[20] G Neri, RCS Morling, GD Cain, E Faldella, M Longhi-Gelati, T Salmon-Cinotti, and P Natali. 1984. Mininet: A local area network for real-time instrumentation applications. *Computer Networks (1976)* 8, 2 (1984), 107–131.

[21] Guangyu Pei and Thomas R Henderson. 2010. Validation of OFDM error rate model in ns-3. *Boeing Research Technology* (2010), 1–15.

[22] Narasimha Prasad, Kishor Kumar Reddy, and Ramya Tulasi Nirjogi. 2014. A Novel Approach for Seismic Signal Magnitude Detection Using Haar Wavelet. In *Intelligent Systems, Modelling and Simulation (ISMS), 2014 5th International Conference on*. IEEE, 324–329.

[23] John R Quinlan et al. 1992. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, Vol. 92. Singapore, 343–348.

[24] Manickam Ramasamy, Shanthi Selvaraj, and M Mayilvaganan. 2015. An empirical analysis of decision tree algorithms: Modeling hepatitis data. In *Engineering and Technology (ICETECH), 2015 IEEE International Conference on*. IEEE, 1–4.

[25] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*. Springer, 15–34.

[26] Zahra Ronaghi, Edward B Duffy, and David M Kwartowitz. 2015. Toward real-time remote processing of laparoscopic video. *Journal of Medical Imaging* 2, 4 (2015), 045002–045002.

[27] Shiva Rowshanrad, Sahar Namvarasl, Vajihe Abdi, Maryam Hajizadeh, and Manijeh Keshtgary. 2014. A survey on SDN, the future of networking. *Journal of Advanced Computer Science & Technology* 3, 2 (2014), 232.

[28] Salvatore Sanfilippo. 2016. hping3(8) - Linux man page. (2016). http://linux.die.net/man/8/hping3

[29] Julius Schulz-Zander, P Lalith Suresh, Nadi Sarrar, Anja Feldmann, Thomas Hühn, and Ruben Merz. 2014. Programmatic Orchestration of WiFi Networks.. In *USENIX Annual Technical Conference*. 347–358.

[30] Mayank Taneja, Kavyanshi Garg, Archana Purwar, and Samarth Sharma. 2015. Prediction of click frauds in mobile advertising. In *Contemporary Computing (IC3), 2015 Eighth International Conference on*. IEEE, 162–166.

[31] Moazzam Islam Tiwana, Berna Sayrac, and Zwi Altman. 2010. Statistical learning in automated troubleshooting: Application to LTE interference mitigation. *IEEE Transactions on Vehicular Technology* 59, 7 (2010), 3651–3656.

[32] University of Michigan Z. Morley Mao. 2010. Network Service Model. (2010). http://www.eecs.umich.edu/courses/eecs489/w10/winter10/lectures/lecture16.pdf

[33] Yongheng Zhao and Yanxia Zhang. 2008. Comparison of decision tree methods for finding active objects. *Advances in Space Research* 41, 12 (2008), 1955–1959.