Report Checkpoint

By: Tatum Maston , Justin Harsono , Charlie Tran

Introduction

Viasat Inc. is an American communications company that provides high-speed satellite broadband services and secure networking systems covering military and commercial markets. Therefore, ensuring high quality network performance is crucial for their business success and maintaining customer satisfaction. Fundamental factors that affect network performance are packet loss and latency. When accessing the internet or any network, small units of data called packets are sent and received. Packet loss occurs when one or more of these packets fails to reach its intended destination. This results in slow service and network disruptions for the user. Latency is the delay between a user's action and a web application's response to that action, often referred to as the total round trip time it takes for a data packet to travel.

The problem we are trying to explore is: can we detect anomalies within network traffic, whether it be an increase in the packet loss rate, larger latency or both. Packet loss can be caused by a number of issues, including: network congestion, problems with network hardware, software bugs and security threats. On the other hand, main factors that affect network latency are: the transmission medium (copper cable-based networks vs. modern optic fibers), propagation (how far apart the networks are), efficiency of routers and storage delays. Ultimately, all of these can cause some form of network degradation and result in an increase in packets being dropped and an increase in delay of data being sent from one computer to another. If we observe a drastic change in packet loss, latency and variables correlated with the two (i.e. an anomaly), then we would also likely expect to see network degradation causing the deviation

from normal behavior. The ability to detect anomalies on a data stream would be extremely useful as a monitoring & alerting system for poor network conditions.

We will be using data generated from DANE (Data Automation and Network Emulation Tool), a tool that automatically collects network traffic datasets in a parallelized manner without background noise and emulates a diverse range of network conditions that are representative of the real world. Using DANE, we are able to configure parameters such as latency and the packet loss ratio ourselves. More importantly, we are able to simulate an anomaly by changing the configuration settings of the packet loss rate and latency mid run. The data collected is already aggregated per second with these fields: timestamps, IP and port addresses of the source and destination, number of bytes and packets sent for each direction of traffic, the milliseconds of when each packet was sent, the size of each packet sent and the sequence of direction of each packet. By having the ability to change the parameters within DANE, we are able to generate as much data as we need with varying network conditions which is helpful in producing an accurate model.

Methods

Prior Work

Last quarter our objective was to develop a machine learning model that can predict the packet loss ratio $\in$ (0,1) and latency in milliseconds given a 10 second snapshot of network traffic. Through this work, we engineered various useful features that correspond to packet loss, allowing us to carry them over to our work with anomaly detection. Some findings from each of our projects last quarter include a handful of useful features, a random forest classifier and a regressor. Each of these models were highly dependent on the features we created, and also was

trained on data that was much more determinant than that we are using now. Since this quarter's work tries to avoid predicting packet loss and latency to determine an anomalous region, most of the work salvaged from last quarter was the domain expertise, exploratory data analysis and familiarity with the data generating tool DANE.

The Data

We ran only two scenarios at once to prevent overloading our CPU by running too many DANE scenarios concurrently. Each DANE run is 5 minutes long. Key differences from last quarter is that we are able to change the configuration settings mid-run and packets are now dropped randomly, rather than systematically. The configuration change happens at the 180 second mark and what we configure for our packet loss rate determines the likelihood of each packet being dropped. This way, we are able to simulate an anomaly within our data and are able to simulate packet loss in a more realistic manner. Our steady state had a packet loss ratio of 1/5,000 and a latency of 40 ms. We are focused on identifying changes of a factor of 4 and above. In our case a packet loss ratio of 1/1,250 and a latency of 160 ms or greater or a packet loss ratio of 1/20,000 and latency of 10 ms. First, we removed the first 20 seconds of each CSV file to get rid of the large spikes that occur in the beginning. These large spikes occur because the computer pushes as much data in the beginning of the DANE run but then the Internet provider slows it back down

The datasets generated include the following features:

DANE Features	Description
Time	Broken down into seconds (epoch)
Port 1 & 2	Identifier of where the packets are being sent to and from. Can originate from either.
IP	IP address of the ports in communication
Proto	IP protocol number that identifies the transport protocol for the packet
Port 1->2 Bytes	Total bytes sent from port 1 to 2 in given second
Port 2->1 Bytes	Total bytes sent from port 2 to 1 in given second
Port 1-> 2 Packets	Total packets sent from port 1 to 2 in given second
Port 2-> 1 Packets	Total packets sent from port 2 to 1 in given second
Packet_times	Time in milliseconds that the packet arrived
Packet_size	Size of the bytes of the packet
Packet_dirs	Which endpoint was the source of each packet that arrived

Cleaning:

At first glance of the data, we notice a few duplicate time points with unusual IP addresses. These rows include an insignificant amount of packets sent relative to the rest of the data, causing us to believe that these are not anomalies – they are irrelevant data points. For this reason, we have chosen to drop these points. We also perceive a large, sharp peak in the beginning of the DANE runs that eventually begins to find the steady state of packet transmission. This is due to the network initially overcompensating to understand how much information it can handle in transmission to where a significant amount of data is not lost.

Because of this, we have chosen to exclude this period from our training set, since it does not describe the steady state transitions that we are attempting to model.

Anomaly Definition

The type of behavior we are looking for is different from what a conventional anomaly would behave like. Typical anomalies are characterized by significantly large spikes or drops in a feature. The behavior we are looking for is more closely related to anomalous regions where the degradation in the network continues; these anomalies resemble shifts. As we will show in the next section spikes in our features that would normally resemble anomalous behavior are perfectly random and not caused by any change in network quality. These anomalies would be considered false positives if detected.

EDA and Feature Engineering

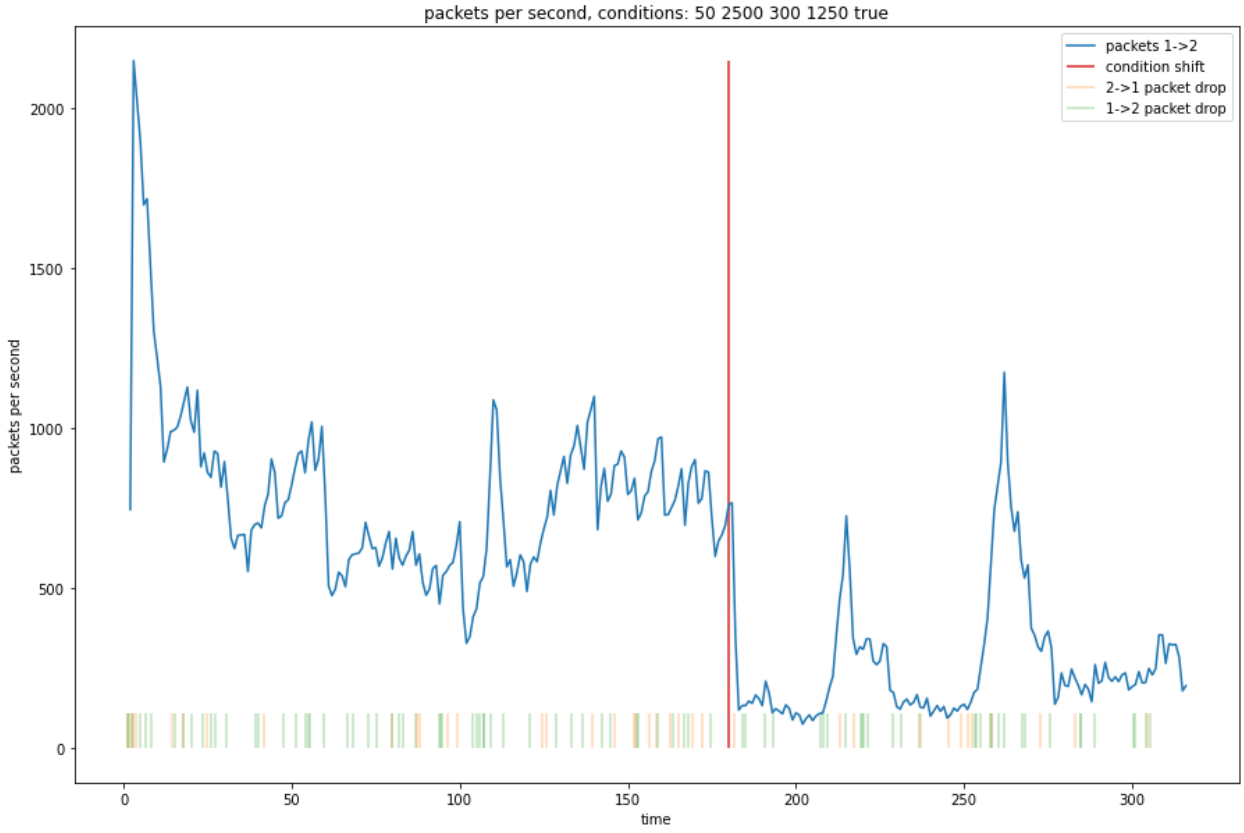

Figure 1: The packets per second sent from machine 1 → 2 with conditions of 50 ms latency and 1/2500 packets expected to be dropped with the conditions shifting to 300 ms latency and 1/1250 expected packet drops at 180 seconds.

Since the data generating process and the context we are using the features is different from last quarter's project, the features used to detect anomalies may have changed as well. Comparing last quarter's deterministic packet drops data with this quarter's random packet drops data, we can see that the correlation between packets per second and packet loss is still there. Since packet loss and latency are our only ways of generating an anomaly, features such as packets per second and other features correlated with packet loss and latency will be our basis for determining whether there is an anomaly or not.

Exploring Packets Per Second Feature

	First 180 seconds	**Last 120 seconds**
Mean	1783.72	428.43
Standard Deviation	710.10	260.22
Max	4404	1548
Min	680	152

In the figure 1 above the green lines at the bottom represent packet drops from the machine sending data while the yellow lines represent packets dropped from the machine sending acknowledgements. The spikes that would normally be looked at as anomalies happen when there is a lack of packet drops for a relatively large number of packets being sent. This can be roughtly modeled by the bernoulie distribution with the packet loss condition being the probability of a "failure". All the machine would see is the ratio of packet being dropped in the last X amount of packets but not the actual probability that we set the packet loss ratio at. So if we get a relatively low number of failures withing the last X packets this would signal to the

machine that the network could handle more packets sent and generate a spike in packets per second without actually changing the network quality (probability a packets is dropped). These unlucky low number of failures could be represented by the left tail in a bernouli distribution. As we can see in figure 1, spikes occurs at 210s and 260s where there has not been a failure in a relatively large amount of packets being sent but the probability of a packet drop has not changed. A shift in packets per second occurs when there is a change in the probability of a packet being dropped (the distribution of packet drops in a given number of packets changes). As a result our features or models have to be robust to these kinds of spikes but still have the ability to detect persistent shifts in the data generated. The change in network conditions happens at 180 seconds and as a result the usual packets per second shifts and stays low unlike a spike where the packets per second would recover in a fairly short amount of time.

Exploration of Anomaly Detection Methods

I. Forecasting

Since we are dealing with time series data, we can create an anomaly detection model through the use of forecasting techniques. The basic concept is that we will pick a feature, in this case total packets sent per second (volume of traffic) and build a forecast. If the expected value is outside of our prediction interval (threshold) we will flag it as an anomaly. We are employing a multivariate time series forecast because we are using predictors other than the series (a.k.a exogenous variables).

There are a multitude of different forecasting models to attempt and in our case, we will be focusing on building an ARIMA (Auto Regressive Integrated Moving Average) model. This model is actually a class of models that ‘explains’ a given time series based on its own past

values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values. Any “non-seasonal” time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models. An ARIMA model can be characterized by 3 terms: p, d, q. P is the order of the “AR” term which refers to the number of lags of Y to be used as predictors. Q is the order of the “MA” term which refers to the number of lagged forecast errors that should go into the model. D is the number of differencing required to make the time series stationary which is crucial because the term “AR” means that it is a linear regression model and works best when the predictors are not correlated and are independent of each other.

To hypertune these parameters of (p,d,q), we performed a grid search method and chose our model based on the AIC (Akaike Information Criteria). The AIC is a widely used measure of a statistical model. It basically quantifies 1) the goodness of fit, and 2) the simplicity/parsimony, of the model into a single statistic. When comparing two models, the one with the lower AIC is generally a “better-fit model”.

To simulate a live stream of data, we chose to implement window intervals of n seconds to train our ARIMA model on. With a focus on a lack of false positives and the perspective that it’s better to let the occasional false negative occur, we found that window intervals of 20 seconds worked best. Therefore, we aggregated our data into 20 second intervals with the mean of the total packet sent as our feature. Our training data consisted of eighteen 5 minute DANE runs for a total of approximately an hour and a half of network traffic. The first 30 minutes of data is our steady state with a latency of 40 ms and a packet loss ratio of 1/5000. Afterwards, we had 5 separate, spaced out instances of simulated anomalies with a return to a steady state in between each case. Each anomaly in our training data changed the latency to 160 ms and the packet loss ratio to 1/1250.

Our test data consisted of twelve 5 minute DANE runs with half of them being steady state traffic and the other half with anomalies simulated. These anomalies had different configurations from our training set and had latency configurations of 320 ms and packet loss ratios of either 1/1250 or 1/500.

To flag anomalous behavior, we calculated a 99% prediction interval around each forecast. If the actual value falls outside this interval, it is flagged as an anomaly. We tuned our prediction interval range from 95% - 99% and found that a 99% prediction interval was a wide enough range to limit the amount of false positives and capture extreme anomalous behavior. In addition, we also log transformed our data (mean of the total packets sent) to reduce the scale because the original data range was in the thousands and since our model considers forecast errors as part of the model, our prediction interval was much too wide. By transforming the data onto a smaller scale, we are better able to capture anomalies.

II. Isolation Forest

In our pursuit to identify anomalies, we explored using an Isolation Forest model on varying windows of our time series data to give an “anomaly score” for each data point. This model tries to separate each point in the data built on the basis of decision trees. Partitions are created by randomly selecting a feature, then a random split between the minimum and maximum value of the selected feature. Since outliers are defined to be less frequent than most regular observations, and are usually much different in terms of their values, we expect that this random partitioning places outliers closer to the root of the tree. In essence, a normal point requires more partitions to be identified than an abnormal point.

For each point, an anomaly score is calculated as:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

Where h(x) is the path length of observation, c(n) is the average path length of unsuccessful search in a Binary Search Tree, and n is the number of external nodes.

To emulate a live stream of data, we chose to implement window intervals of n seconds to train our models on. Using this approach would yield multiple anomaly scores for each data point, as the window moves through the time series. To get a final anomaly score for a given data point, we just average each score given by the windows including that point.

III. DBScan

The motivation behind using DBScan was to detect abnormal changes in the data. This was before we had a proper definition of an anomaly. DBScan (density based) is particularly sensitive to spikes since it uses distance and number of points within that distance. Spikes tend to change rapidly and not have enough points near its peak to form a cluster. As a result spikes were flagged as an anomaly because they were not in a cluster. Since we wanted to punish false positives the most we did not use DBScan in our model

IV. MAD and Median

This model tries to take advantage of the differences in the features of spikes and shifts in a time series. Median and median absolute deviation are transformations to the 1 → 2 packets per second data using a rolling window. Some properties of this transformation: smoothens out the data, depending on the window size median and MAD is robust against spikes, is sensitive to shifts in the data. Median and MAD depend on the assumption that spikes are relatively short

and uncommon, and shifts are persistent. When a spike occurs the median would remain almost unchanged since the data points outside the spike would make up more than 50% of the window same applies to deviation from the median, hence why larger window sizes tend to do better. Shifts in the data on the other hand eventually result in a shift in the median since they persist much longer than 50% of the rolling window size. The model determines if a data point is an anomaly if a transformation of the median and MAD are above a threshold based on the window size. Deviation from the median was included in the transformation to limit the effect of large variance in the window. If there is a lot of variance but no shift, the data would be above and below the median so the sum of the deviations would be low in magnitude but in a shift the data would be either strictly above or below the median and result in a high magnitude. All of these transformations that picked at differences between spikes, shifts and data with high variance in a window would be combined to make the presence of a shift more apparent. The transformation used was:

$$Transformation(window) = \frac{MAD(window) * DM(window)}{median(window)}$$

Where MAD is median absolute deviation and DM is the sum of the deviation from the median

Results

I. Metrics

We chose to use F1 score as our metric but also considered precision in our metrics. Our motivation behind choosing F1 score and precision is because anomalous regions cause by network degradation are assumed to be rare. Since the intended use for this ensemble model is to alert ISP employees what connections are degraded if too many false positives are being flagged

then there would not be much signal for an anomaly in the alarm. Our goal is to have the users trust that if there is an alarm they can be sure that there is network degradation, hence why precision is considered. We also do not want our model to always predict negative, making it trivial so F1 score was included.

II. Forecasting

We trained our model on the first 75 points of data (1,500 seconds or 25 minutes of network traffic). The two graphs show the results of the ARIMA model on the training set mentioned above. We can see that the model was able to detect whenever there was a configuration change which caused the significant drops and in one instance it detected the recovery when it jumped back to the steady state. One graph shows the results on the normal scale and the other graph shows the results on the log scale.

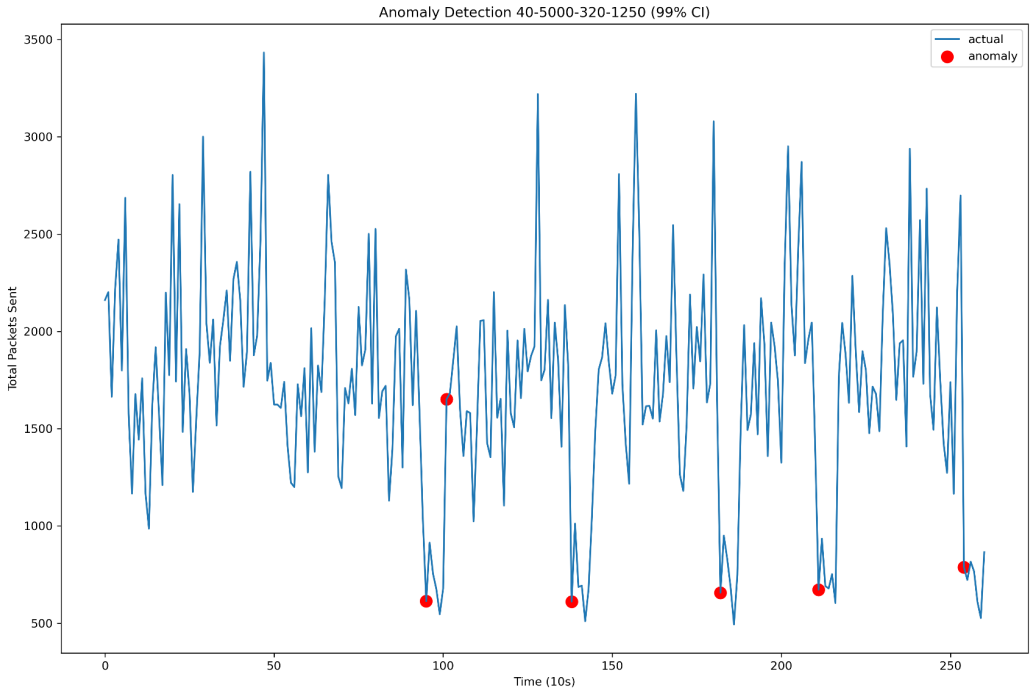

Figure 2: ARIMA model anomaly detections using a 99% CI on the training set. The conditions generating the data: 40ms latency and 1/5000 packets dropped shifting to 320ms latency and 1/1250 packets being dropped. Time is measured in units of 20s since the ARIMA model trains on 20s aggregations of packets per second as a single data point.

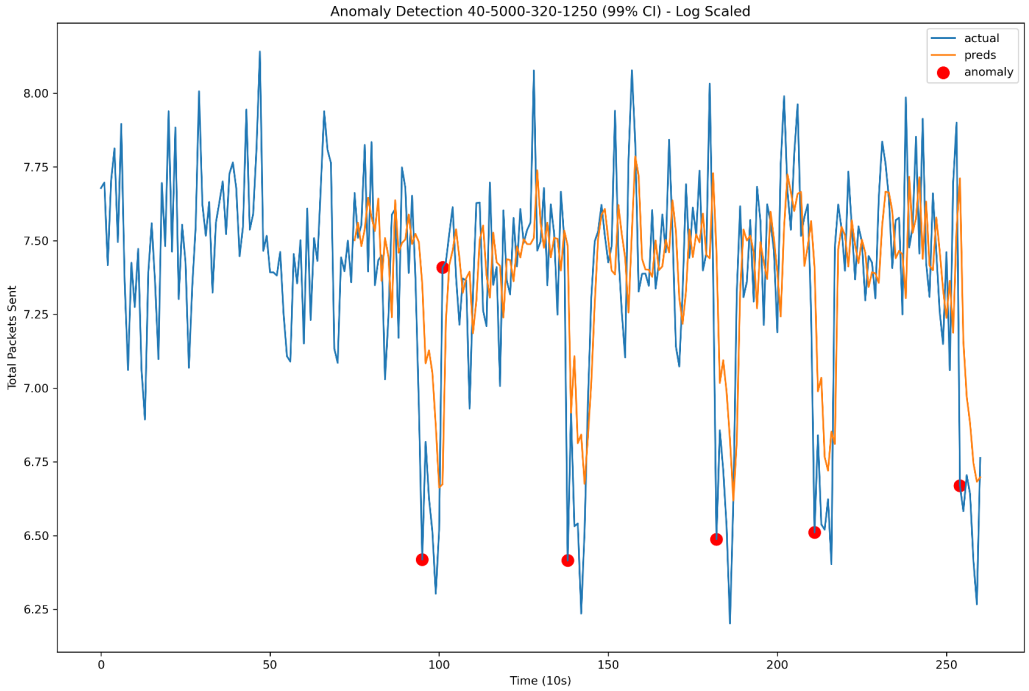

Figure 3: ARIMA model 99% CI predictions on training data with a log scale. The conditions generating the data: 40ms latency and 1/5000 packets dropped shifting to 320ms latency and 1/1250 packets being dropped. The ARIMA forecasts can be seen in orange.

We then ran the model on our test set concatenated to our train data. We can see in figure 4 that there were 2 instances of false negatives where our model did not flag the configuration change as an anomaly. There was also a large spike in the middle of our steady state traffic which was flagged as anomalous, representing one case of a false positive. This window of traffic just so happened to have an extremely large volume of packets sent and was not due to a configuration change. Once again, the first graph shows the results on a normal scale and the second graph shows the results on the log scale.

Another observation is that our model never detects an anomaly during a resurgence back to a steady state or during the timeframe soon after.

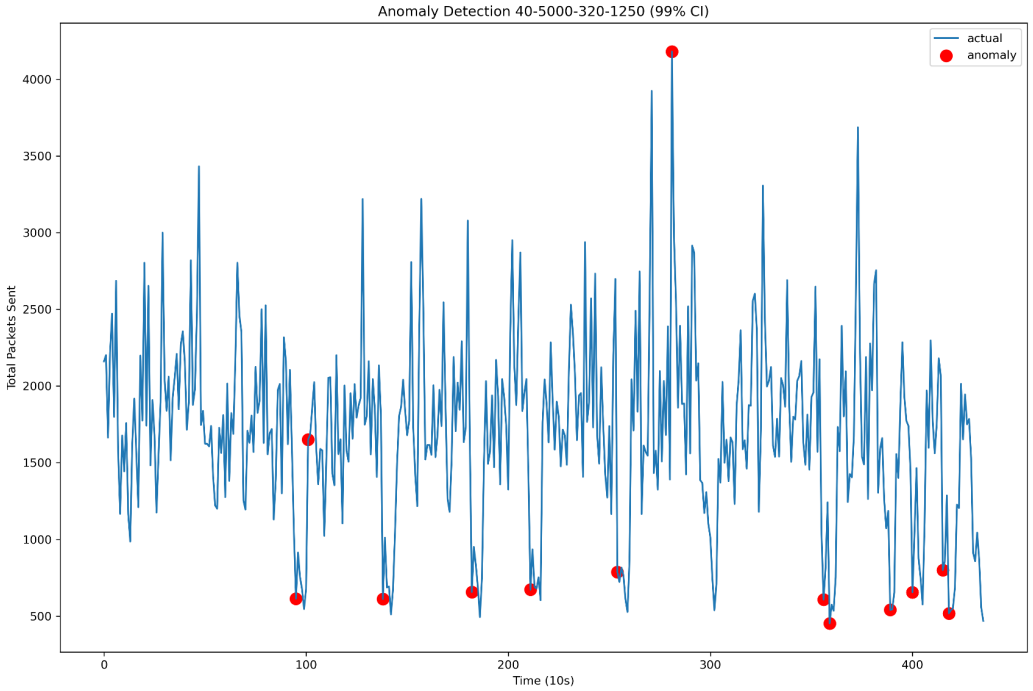

Figure 4: ARIMA model anomaly detections using a 99% CI on the test set. The conditions generating the data: 40ms latency and 1/5000 packets dropped shifting to 320ms latency and 1/1250 packets being dropped. Time is measured in units of 20s since the ARIMA model trains on 20s aggregations of packets per second as a single data point.

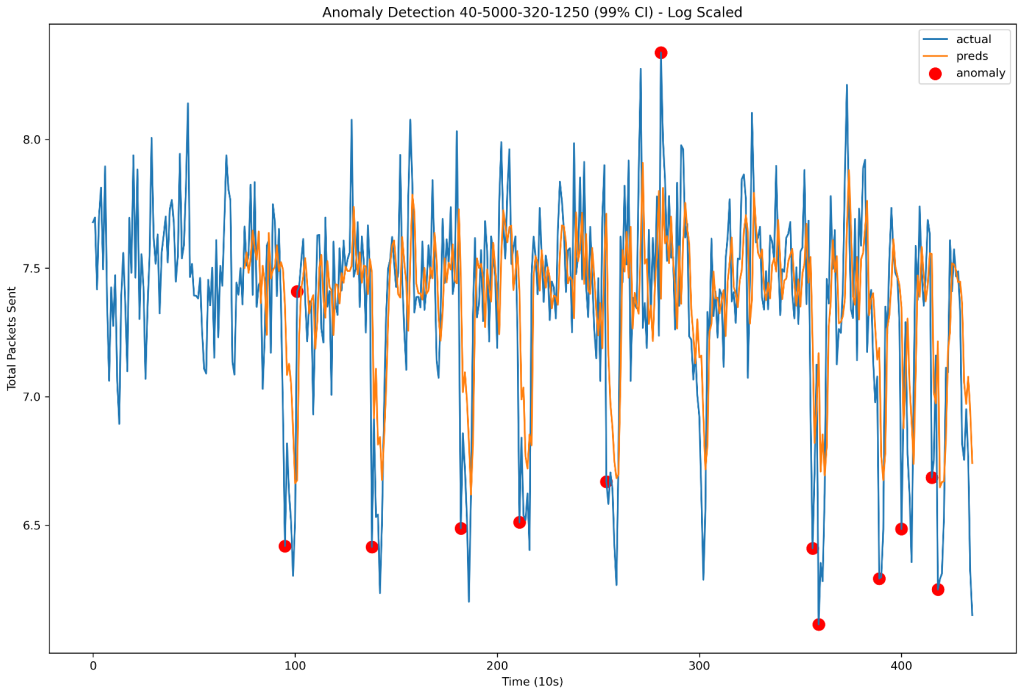

Figure 5: ARIMA model anomaly detections using a 99% CI on the test set with a log scale and forecast predictions.

III. Isolation Forest

Because of its good reputation with anomaly detection, we went to great lengths to get the Isolation Forest model to perform well on our data. However, it continued to perform poorly, leading us to depreciate it for our final anomaly detection approach.

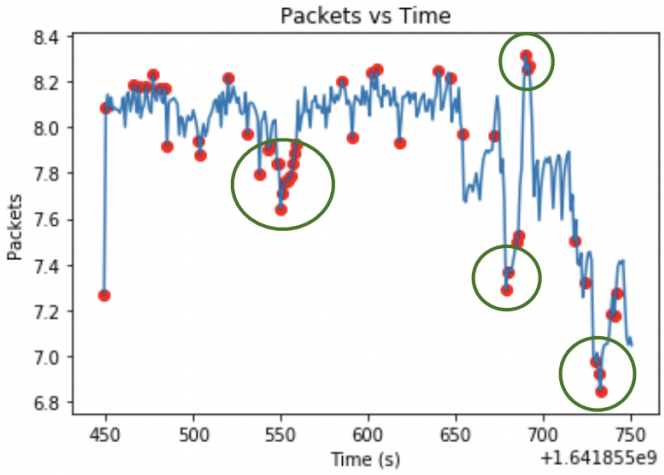

Figure 6: Isolation forest anomaly predictions.

As described above, the Isolation Forest is based upon the principle that anomalies are far and few. However, since our data is modeling real world instances of network traffic, we experience drops in information that are caused by network degradations. These degradations are the anomalies we are looking for. The nature of these degradations is that it lasts for more than a few milliseconds, leading the packet transmission to be affected for a period of time. This leads us to have anomalous regions rather than single anomalies, which is why Isolation Forests do not work well. After a failed attempt at creating many new features, transformations, and smoothing, we decided to not continue with Isolation Forests any further.

IV. DBScan

The anomalies we were looking for were shifts which generally had many points within the eps i.e. distance, close enough and abundant enough to form a cluster. These would not be flagged as anomalies. These two properties can be seen in the graph below. In order to make use of DBScan we would have to exclusively use features that created a spike or trough when the shift (an indicator of a change in latency or packet loss) happened and be robust agains spikes that are a result of unlucky sequences of packet drops.

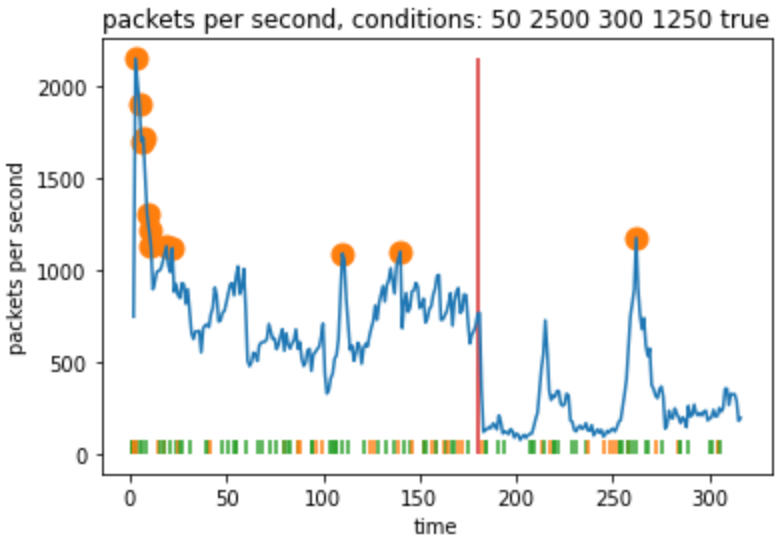

Figure 7: DBScan anomaly predictions. Conditions used to generate data: 50 ms latency and 1/2500 packets expected to drop shifting to 300ms latency and 1/1250 packets dropped. The red line indicated the time the condition shift happened. The ticks at the bottom indicate when a packt was dropped, green is 1 → 2 yellow is 2 → 1.

V. MAD and Median

The first graph shows the median transformation of 1 → 2 packets using a rolling window. The second graph shows the MAD of the same graph. The vertical red line is where the change in the network conditions happens. As shown in the first plot, the median is unaffected by any large spikes in the data but responds to the shift in the data, albeit delayed. The delay is a 50% of the window size. Likewise the MAD is fairly resilient to spikes and creates a spike much larger than any other as a result of the shift in packets per second. The third graph is a combination of the first and second graph, showing where the anomaly was detected in the second graph shifted by 50% of the window size (80s).

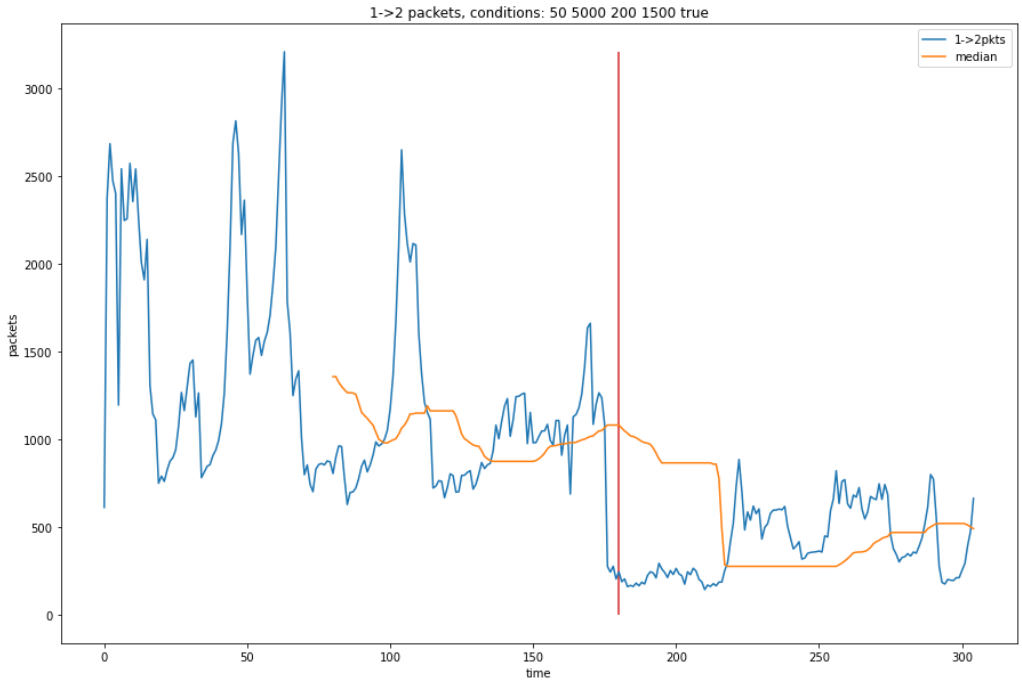

Figure 8: Median on 1 → 2 packets per second with window size of 80s. Conditions used to generate data: 50 ms latency and 1/5000 packets expected to drop shifting to 200ms latency and 1/1500 packets dropped. The red line indicated the time of the condition shift. The yellow line indicates the median of the 80s before it. The first point in the yellow line would indicate the median of the 80 data points without the yellow line.

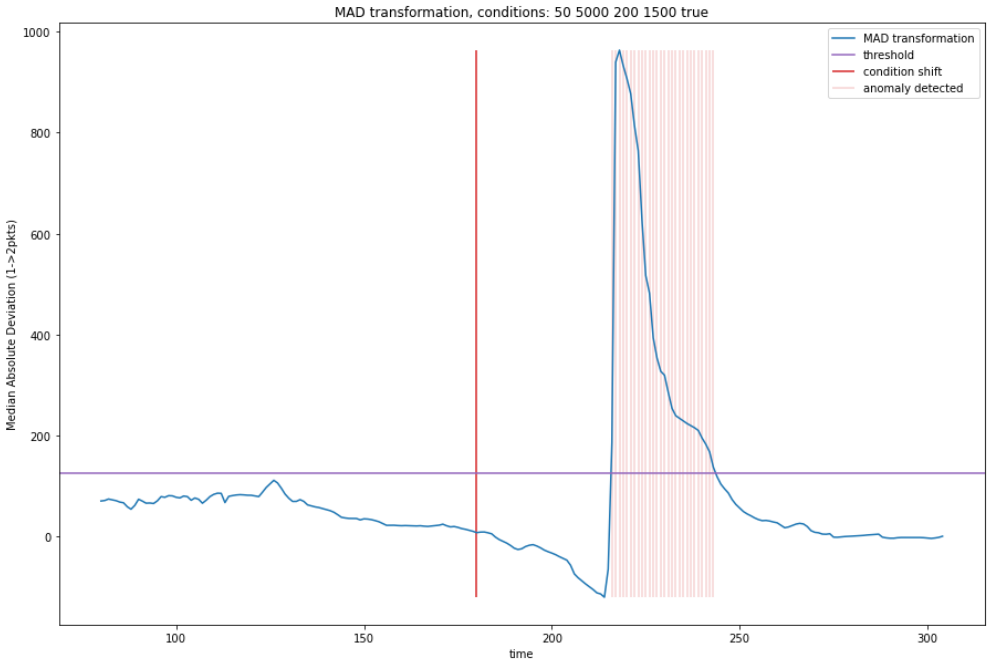

Figure 9: Transformation of statistics of a time window i.e. median. Since the transformation uses a window the spike as a result of the shift is delayed by a function of the window size.

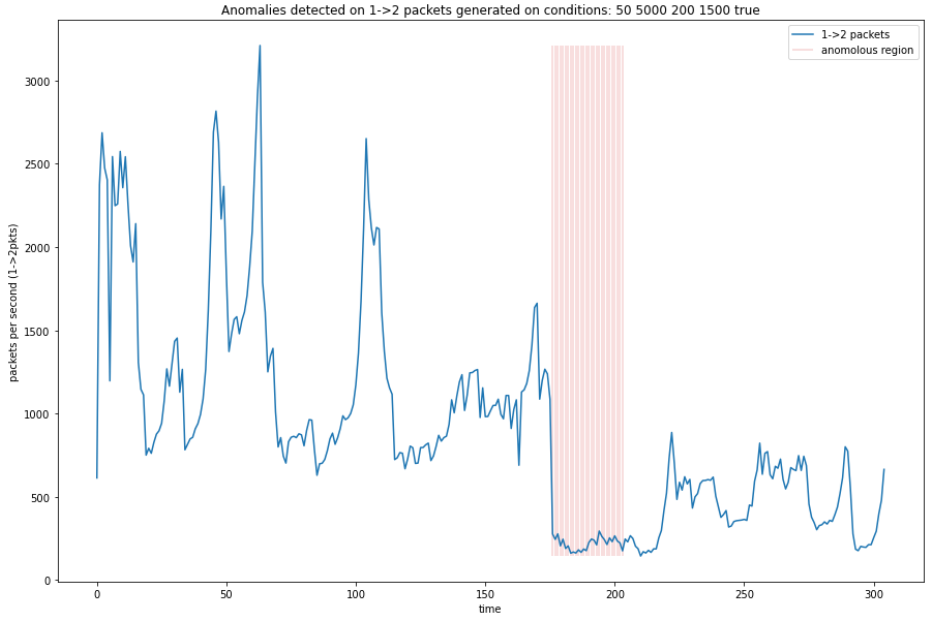

Figure 10: Anomalies detected using the transformation of the median absolute deviation. Window size: 80s

VI. Ensemble Model

Our method to combine our ARIMA and MAD model followed a simple logic. We ran both models independently and compared the results of each model. We used an AND operator to see where both models detected an anomaly and labeled it as an anomaly. Otherwise, if the condition is not met, then we do not label it as an anomaly. First, we look at the result of our MAD model on the same dataset used earlier in the ARIMA section. The MAD model utilized a window size of 80 with a threshold of 100 and 'total packets' as our feature.

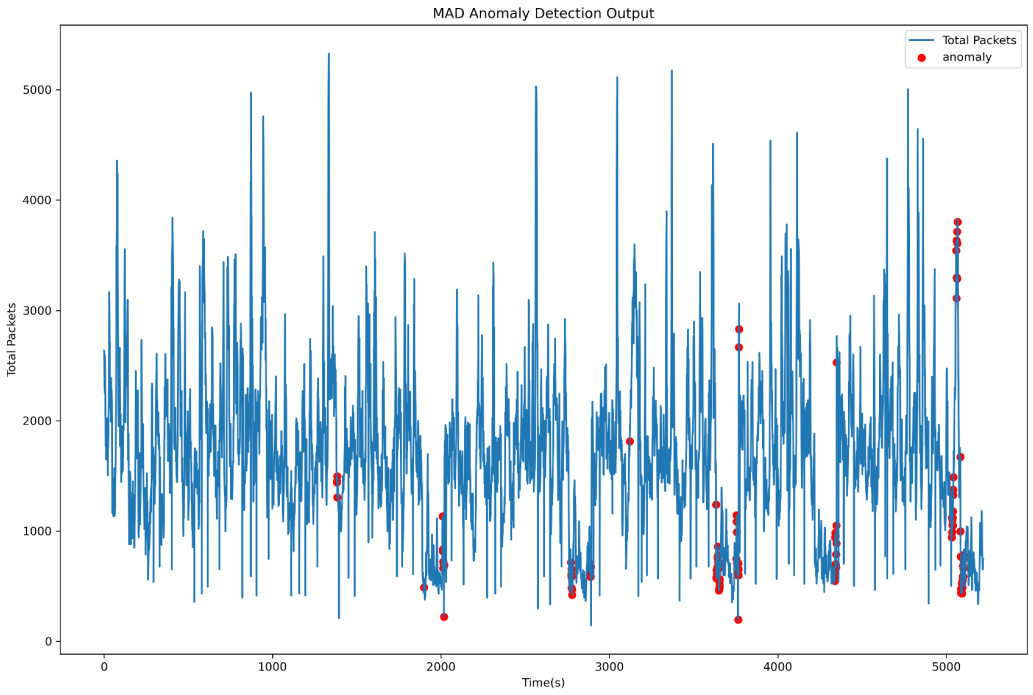

Figure 11: MAD anomaly detection model with window size 80s. Several dane runs were stitched together.

Here we can see that the MAD model captured every configuration shift and some of the timeframes when the network goes back to a steady state. However, there are several instances of false positives that occurred as well. Since MAD takes one second inputs and our ARIMA takes in 20 second windows as inputs, we had to transform our ARIMA results back into one second outputs. Our method of doing this was to go back and transform our 20 second window into one second windows and for each anomaly in our 20 second window, we labeled the entire 20 seconds as an anomaly.

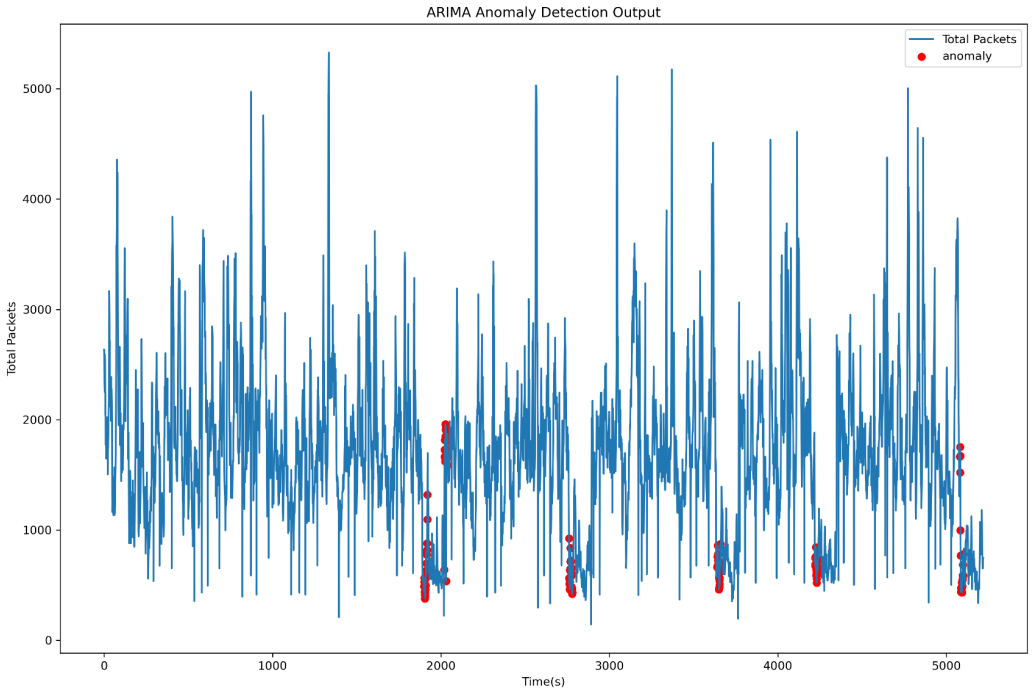

Figure 12: ARIMA anomaly detection model with window size 80s. Several dane runs were stitched together.

This graph represents the same results generated in the ARIMA section but our x-axis now represents one second windows rather than 20 second windows. Now, using the ensemble logic described above, our results show that we are able to detect anomalies early on when they occur.

However, we have one false negative where no anomaly appears due to the fact that our ARIMA model detects the anomalies early on but the MAD model detects the anomalies later, so there is no overlap for our ensemble logic to agree on an anomaly.

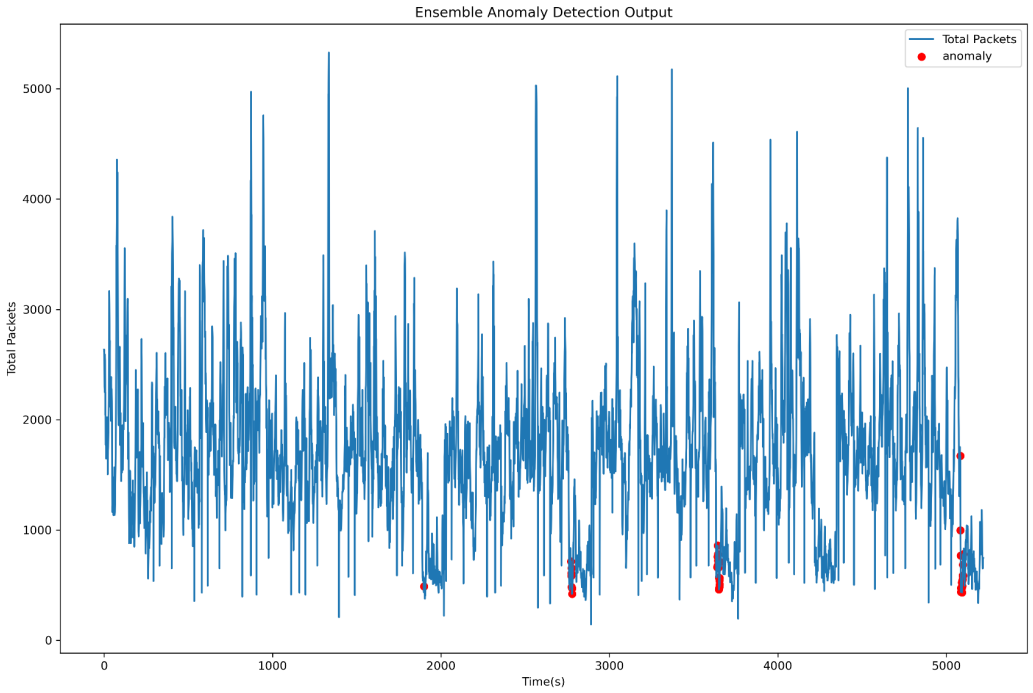

Figure 13: Ensemble of the MAD and ARIMA model. The two models were combined using the AND operator with the anomaly prediction from each model as input.

Discussion

A major portion of work that our group was not able to venture into was to generate a more realistic dataset to test our model on. Our methodology of concatenating multiple runs of DANE to "simulate" about an hour and a half of network traffic is not exactly accurate because we do not know if networks jump back to a steady state as quickly and abruptly as shown in our

dataset. Also, we were only able to train and test our model on one configuration for our steady state which was a network with a latency of 40 ms and a packet loss ratio of 1/5000. It would have definitely been interesting to see how our model would perform on a test set with a new steady state so we can further hypertune our model. For example, the ARIMA model only uses "total packets" as its feature and we know from our results from last quarter that different latency and packet loss ratios results in different behavior for total packets sent over the network. Therefore, the parameters for our ARIMA model would most likely change if our steady state changes. As we have it right now, our ARIMA model is only trained on one configuration so an addition we would have to make is to continuously or periodically retrain the ARIMA model to make sure that it is correctly hypertuned. Moreover, our ensemble logic was very naive and rough and it would've been extremely useful to flush our ensemble model out even further. As we currently have it, our MAD model is hypertuned to have a window size of 80 seconds while our ARIMA model is hypertuned to take in inputs of 20 seconds. The different inputs of each model may need the development of different data pre-processors for each.

Resources:

https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

https://towardsdatascience.com/anomaly-detection-with-isolation-forest-visualization-23cd75c281e2

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

https://people.duke.edu/~rnau/411arim.htm

https://coolstatsblog.com/2013/08/14/using-aic-to-test-arima-models-2/