# A Quantitative Comparison of Algorithmic and Machine Learning Network Flow Throughput Prediction

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

by
Cayden Todd Wagner
May 2022

Accepted by:
Dr. Jim Martin, Committee Chair
Dr. Long Cheng
Dr. Rong Ge

# Abstract

Applications ranging from video meetings, live streaming, video games, autonomous vehicle operations, and algorithmic trading heavily rely on low latency communication to operate optimally. A solution to fully support this growing demand for low latency is called dual-queue active queue management (AQM). Dual-queue AQM's functionality is reduced without network traffic throughput prediction.

Perhaps due to the current popularity of machine learning, there is a trend to adopt machine learning models over traditional algorithmic throughput prediction approaches without empirical support. This study tested the effectiveness of machine learning as compared to time series forecasting algorithms in predicting per-flow network traffic throughput on two separate datasets.

Autoregressive integrated moving average (ARIMA), a deep neural network (DNN) architecture, and a long short-term memory (LSTM) neural network architecture were used to predict future network throughput in two different datasets. Dataset one was used in establishing the initial performance benchmarks. Findings were replicated with a second dataset. The results showed that all three models performed well. ANOVA failed to demonstrate a statistically significant advantage of machine learning over the algorithmic model. From dataset one, ANOVA F = 0.138 and p = 0.983. From dataset two, F = 0.087 and p = 0.994. The coefficient of determination tested the fit of models in the two datasets. The r squared value ranged from 0.971 to 0.983 in the machine models to 0.759 to 0.963 in the algorithmic model.

These findings show no evidence that there is a significant advantage of applying machine learning to per-flow throughput prediction in the two datasets that were tested. While machine learning has been a popular approach to throughput prediction, the effort and complexity of building such systems may instead warrant the use of algorithmic forecasting models in rapid prototyping environments. Whether these findings can be generalized to more extensive and variable datasets is a question for future research.

# Table of Contents

# List of Tables

# List of Figures

cheive

# Chapter 1

# Introduction

In October of 1986, the Internet experienced its first large-scale congestion collapse, where the Internet's throughput dropped by a factor of almost 1000. At that rate, a computer would take over 20 seconds to send a 100 character email. A solution to the congestion collapse had to be made for the Internet to continue its growth. The congestion collapse event spawned a seminal paper in computer networking authored by Jacobson [9]. It detailed methods and algorithms to prevent congestion collapse from happening again. The paper was highly influential, and the detailed core concepts are still used today.

Similar to 1986, the Internet is approaching another bottleneck in performance. TCP, the most commonly used Internet protocol today, has a problem with throughput fairness. As it is currently, TCP is a greedy protocol that attempts to obtain as much network bandwidth as possible. This behavior can cause unfair and inefficient performance for small communication messages traveling along the same network paths. This issue has been partially solved with active queue management (AQM) algorithms like RED and CoDel, which aim to slow TCP's bandwidth acquisition. These algorithms are adequate from a throughput perspective, but there is room for improvement from a latency perspective. Low latency networking is the next frontier of computer networking, and current Internet traffic control techniques are not enough to support this frontier.

It is critical to discuss real-world cases for low latency networking on the open Internet. Queueing delays can be orders of magnitude greater than that of the propagation delay (the time it takes for information to move across physical network lines) for a network message sent halfway around the world [5]. According to Carlucci et al. [5], adding bandwidth is comparatively inexpensive but is ineffective for reducing latency. Instead, these authors argued that the better approach is to focus on AQM algorithms to reduce Internet latency. Real-time applications like video meetings (Zoom, WebEx, Teams), live streaming (Twitch), video games (many multiplayer first-person shooters), autonomous vehicle operations (including airborne drones swarms), and algorithmic trading applications that demand low latency to function effectively [2, 5]. Such applications perform noticeably worse or not at all when their low latency needs are not met.

Improvement in latency performance positively affects a user's experience with the application's efficiency (and vice versa), as shown by Amin et al. [1]. These authors specifically demonstrated that degradation in network latency and increased network jitter negatively affected a user's perceived experience while playing an online game. The previously mentioned applications that can suffer from poor latency, if implemented correctly, stand to gain orders of magnitude improvement from supporting their low latency needs.

A critical piece of technology that needs to be introduced before discussing solutions to providing low latency is time series forecasting methods. Before the popularization of the GPU and, consequently, neural networks, the primary method for predicting future trends with past data were time series forecasting algorithms. In

1970, a seminal paper was published by Box and Pierce [3]. These authors introduced ARIMA (autoregressive integrated moving average) as an effective method for predicting future trends. Since then, ARIMA has successfully been applied in many fields, including economics, medicine, astronomy, and computer networking. However, ARIMA can fall short when multiple data features and small subtleties exist in the historical data. In 2012 there was a resurgence in neural network popularity following the publication of a paper by Hinton et al [8]. The paper showcased the merits of neural networks run on the GPU. Until that point, ARIMA was the defacto standard for trend prediction in every area it was applied. Nowadays, the hierarchies for the most effective prediction methods almost always contain some form of a neural network at the top of the hierarchy.

With a proven and reliable way to predict future trends, a new AQM technique called dual-queue has been proposed as a solution to delivering low latency to the Internet. Dual-queue active queue management (AQM) was proposed by Briscoe et al. [4] then adopted by White et al. in a white paper [17]. Prior to [4], the best forms of low latency supporting AQMs could not provide low enough latency or the needed consistency to support applications when subjected to extreme traffic loads. Both CoDel and RED fall short in providing latency below five milliseconds. Briscoe states that even current state-of-the-art AQMs like Fair/Flow Queueing and CoDel (FQ CoDel) achieve an average of 5-20ms latencies. This is far from optimal for increasingly demanding, low latency applications. Further, queuing latency can easily reach upwards of 100ms during the extreme loads of peak Internet use when using CoDel or RED. White et al. argued that this new form of low latency queueing would positively impact business applications and end-users alike [17].

Dual-queue AQM has promising prospects to be a powerful supporting mechanism in the effort to create a low latency Internet. Part of the reason dual-queue

AQM is promising is that before now, no mainstream queuing algorithm has logically separated network traffic that needs lower latency from the bulk of latency agnostic traffic. Previously, a video call across the country using respectively low bandwidth would need to wait in the same queues as network traffic mass downloading family photo libraries from the cloud and thousands of Netflix streams with a decent buffer of video already received. The Netflix streams and photo library downloads could cause an extreme delay, creating a poor real-time experience for the video call users.

The underlying component of this mechanism needs to be evaluated and refined to support dual-queue AQM's rise in popularity and effectiveness. Dual-queue AQM relies heavily on per-flow network throughput prediction to make its decisions. Essentially, the dual-queue AQM needs to know how network flows will behave in the future so it may queue current traffic accordingly.

Currently, there are two methods for predicting future network flow throughput. The first is a classical algorithmic approach using time series forecasting, and the second method uses machine learning to predict network flow throughput.

At the top level, machine learning may seem like the obvious option because of its proven capabilities in predicting future trends. This view, however, may not be correct as machine learning is very computationally expensive and burdensome to custom tailor for variations in its various deployment environments. On the other hand, algorithmic approaches are simpler to implement and deploy as they are "plug and play." One pitfall of algorithmic approaches like ARIMA is that they are known not to recognize subtle features in the data. Both approaches have their merits and drawbacks.

For the above reasons, this study tested the classical algorithm and modern machine learning approaches to predicting network per-flow throughput to determine which approach is better suited for the open Internet. This study aimed to test and discuss the benefits and limitations of each approach in terms of computational cost, accuracy, viability at scale, and in constrained domain areas under a predefined set of assumptions and operating conditions.

## 1.1 Dual-queue AQM

Dual-queue AQM separates queue building and non-queue building network application flows into separate packet switching queues. This queue separation allows latency-sensitive application flows to be "fast-tracked" as they are generally non-queue building and can skip to the front of the queue to reduce their latency. The queue building application flows that are typically not latency sensitive are put into a low-priority queue. Both of these actions taken by a dual-queue AQM help support the low latency requirements of the modern-day Internet. An important note about dual-queues is that although the fast queue has priority, all applications using the dual-queue system have fair and comparable throughput for their respective flows. Fair and comparable throughput is possible because the low latency queue enforces a minimal amount of data transfer allowed by each application flow. The low latency queue attempts to maintain the smallest possible queue size. Any offending application flows that hinder its attempts to maintain a minimal queue size will be relocated either partially or totally to the standard queue.

Small queue size enforcement is one of the primary motivations behind creating an accurate and consistent application throughput predictor. Machine learning and time series forecasting algorithms are the main two methods for predicting application and network throughput. This study directly compares these throughput prediction methods to ascertain if there is an optimal choice for per-flow throughput prediction. Many studies have investigated the performance differences between machine learning and algorithmic approaches to time series forecasting, but to this author's knowledge, not within the domain of network throughput prediction on a per-flow basis. A direct comparison between the two methods was not found during a review of the literature. Each technique has potential associated benefits and costs that need to be tested before either solution is adopted for helping maintain a minimal low latency queue size.

## 1.2 Explicit Congestion Notification

In conjunction with the previous prediction methods, dual-queue AQM systems can also take advantage of a mechanism in internet communications called Explicit Congestion Notification (ECN). ECN is a process for marking internet packets to notify relevant systems and services of network congestion as proposed by Ramakrishnan et al. [14]. These authors state that while using ECN, the waiting time for retransmission timeouts becomes far less, enabling more efficient communication in worst-case scenarios. ECN is an extension of the IP standard but is in no way a requirement of IP. For this reason, the ECN flag in a network packet can signify other information if the network it is traversing has been set up to interpret that flag. For instance, if a network has a predetermined set of applications allowed on the low latency queue, then packets from these applications may be marked with the ECN flag to be let onto the low latency queue.

Another scenario where ECN is used is in conjunction with machine learning and algorithmic flow classification. A flow in the context of this study is a series of network packets sent in a single direction from a specific set of addresses and ports. Using ECN in conjunction with machine learning and algorithmic thresholding works by marking a flow's packets to notify down and upstream senders of imminent con-

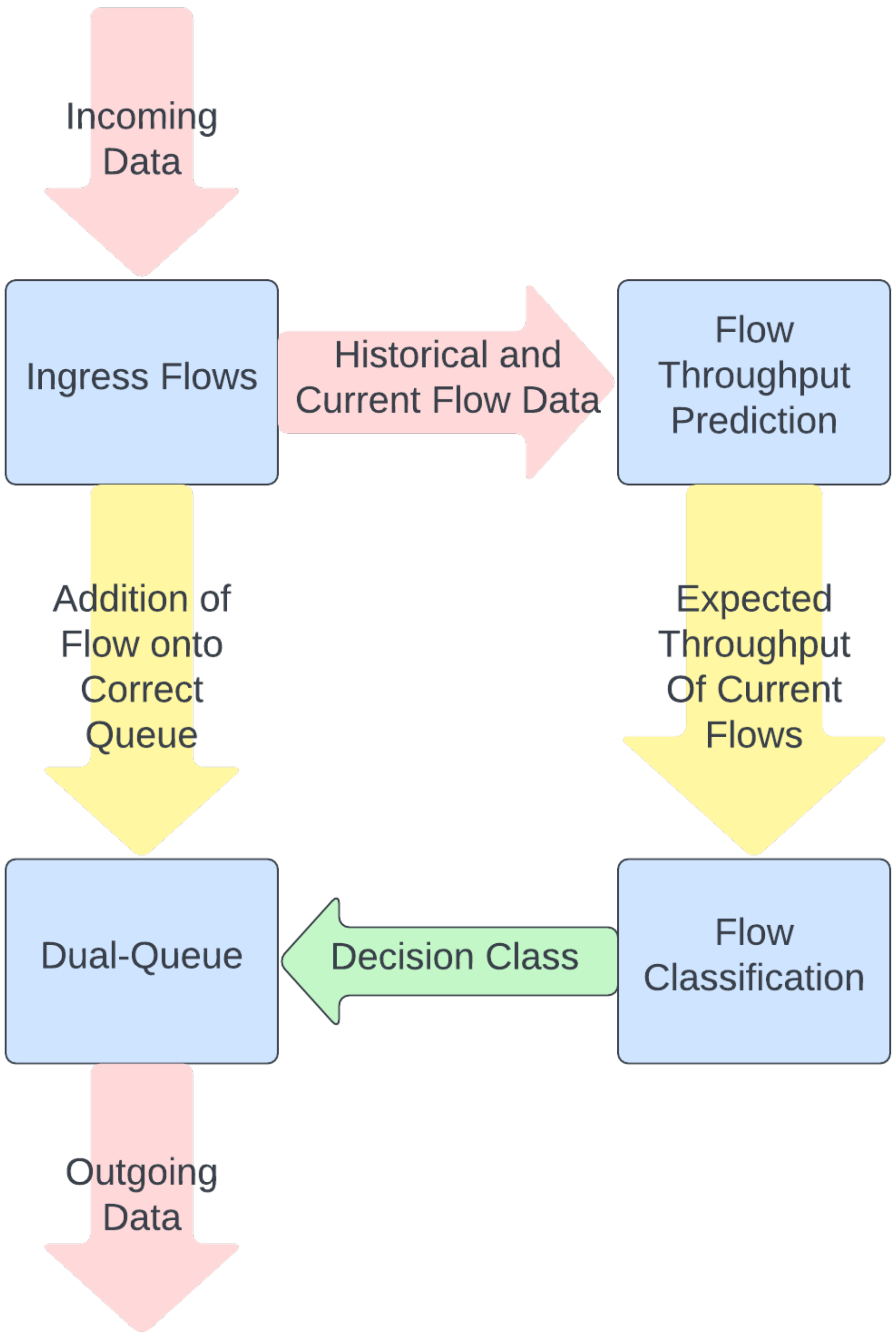
Incoming Data
Ingress Flows
Historical and Current Flow Data
Flow Throughput Prediction
Addition of Flow onto Correct Queue
Expected Throughput Of Current Flows
Dual-Queue
Decision Class
Flow Classification
Outgoing Data

Figure 1.1: Diagram illustrating the general structure of Dual-Queue AQM and the flow of data through it with respect to the work of this study

gestion, as explored by Majidi et al. [12]. These authors found that using a machine learning-based dynamic thresholding to mark packets with the ECN flag allowed for support of low latency and low loss in datacenters. This study will specifically not include ECN because this study deals with throughput prediction over the open Internet, and ECN serves a different purpose on the open Internet.

To explain the 1) relationship between network throughput prediction, flow classification, and dual-queue AQM systems; 2) what each system does; 3) how they feed into each other are all necessary topics to discuss. All three components must be refined, integrated, and appropriately evaluated to build genuinely robust and functional dual-queue systems. As illustrated below, in Figure 1.1, there is considerable data sharing needed between each component to decide in what queue a flow belongs. Each of the blue squares in Figure 1.1 represents a component needed to make a fair, efficient, robust, and reliable dual-queue system. Between each of these components are colored arrows representing data flow between components. The color of these arrows signifies the relative amount of data transfer needed to support the next component in the system.

## 1.3 Dual-Queue and Queuing Delay

As loosely defined in the previous section, dual-queue active queue management is a form of congestion control centered around reducing latency and variance in queuing latency stemming from queuing delays [17]. Queuing delay is wildly variable. If no packets are in front of an incoming packet in a queue, the queuing delay will be zero. Conversely, suppose a packet arrives at a fully saturated queue. According to Kurose and Ross [10], the queuing delay can take a tremendous amount of time proportional to the other delays a packet can experience.

Dual-queue AQM partially solves this issue of variable and disproportional queuing delay by creating two queues. One queue accepts non-queue building network traffic while the other accepts all traffic rejected by the low latency queue. Applications that require low latency networking tend to create network traffic in a non-queue building way. This non-queue building behavior means that network traffic self identifies in which queue it should belong.

### 1.3.1 Integration of Dual-Queue AQM With Flow Classification

In a classic dual-queue AQM system, the flow class (to which queue a flow belongs) is decided retroactively. Flow class is decided by defining or calculating a threshold of throughput. When this throughput threshold is passed, a flow will be categorized as queue building and will no longer be put on the low latency queue. Oljira et al. [2] raised an issue when evaluating dual-queue performance using data center TCP (DCTCP). Suppose a non-queue building flow creates a temporary burst of packets through one mechanism. In that case, the latency-sensitive flow will experience an extreme degradation in performance because of the single burst incident. Oljira argued that the next step was to look at other TCP protocols to see if this issue could be solved using a different protocol. A different solution to the problem Oljira discussed is to instead try and account for the burstiness of TCP and correctly classify a flow that may have unexpected but short bursts in throughput. This solution can is grouped under flow classification.

## 1.4 Flow Classification

### 1.4.1 Flow Classification Defined

In the context of this study, there are two types of flow classification techniques, but they both have the same goal. The goal of flow classification is to identify and prioritize short non-queue building flows as defined by Guo and Matta [6]. The first and current technique of classic dual-queue AQM is retroactive classification. This classification technique takes recent data of each flow and classifies that flow based on how it has recently behaved. This technique has two variants: the first waits until a flow breaks the throughput threshold and permanently classifies it as a classic flow. The second variant decides the class of a flow actively based on recent flow behavior and will allow reclassification often.

Each of these two variants has its benefits and drawbacks. The first variant (wait and see, no reclassification) has the advantage of never letting a repeat queue building offender back into the low latency queue, preserving the performance of the low latency queue. The drawback to this approach is that if a latency-sensitive flow temporarily breaks the throughput boundary, it will have a permanent and severe hit on its latency performance. The second variant has exactly the opposite advantages and disadvantages to the first variant. The second variant will let repeat queue building offenders back into the low latency queue if their traffic behavior is sinusoidal with dips and peaks in throughput. This disadvantage will temporarily and frequently ruin the low latency performance of the dual-queue. On the other hand, the second variant will never permanently remove a non-queue building flow from the low latency queue, meaning that a latency-sensitive flow will never have a permanent hit to performance.

Both of the variants of the first method have substantial drawbacks that the second method addresses. The second method obtains the best of both variants in terms of advantages and disadvantages from the first method's two variants. The second method attempts to proactively decide a flow's class to prevent the disadvantages of the two variants of the first method. In proactively predicting which class a flow will be before that flow's data reaches the queue, this technique achieves two new positive outcomes. First, the dual-queue has a better chance of moving flows that will break the throughput threshold before they can cause a performance hit. Second, the dual-queue can preemptively move a flow back to the low latency queue once it predicts the flow will no longer produce queue-building traffic.

### 1.4.2 Integration of Flow Classification With Flow Throughput Prediction

With the convincing benefits of proactive flow classification discussed above, the next step needed to create a functional and robust dual-queue is a way to predict what class a flow will be in the future before it becomes that class. The simple answer to this step is to predict the future throughput and apply the usual throughput threshold to the predicted throughput. The tricky part is to predict the future throughput accurately. Attempting to predict future throughput is where the techniques of time series forecasting and machine learning come into play.

## 1.5 Flow Throughput Prediction

The final component of a robust and functional dual-queue AQM is the prediction of the future throughput of a network flow given past flow data. The prediction needs to be relatively accurate, accounting for the scale of the network, i.e., 0.01Mb of average prediction error on a 1Mb/s network path and 1Mb of average prediction error on a 100Mb/s network path. There are two main methods for predicting the throughput of a flow. The first and more traditional computer networking approach uses time series forecasting algorithms to understand the direction a flow's throughput is headed. The second and more modern approach uses the subsets of machine learning, either deep neural networks (DNN) or recursive neural networks (RNN).

Flow throughput prediction research using RNN has previously been explored by Zhong et al. [18]. The research primarily involved the integration of an RNN into the flow queue classification process and evaluated performance based on latency in the network. Zhong's work was a great showcase of the potential of machine learning when applied to a dual-queue AQM system. However, it provided little to no comparison of an algorithmic classification baseline. Another area of interest that was not covered was when machine learning approaches did well at prediction and when they failed to meet a baseline. Producing a set of assumptions under which the prediction model will operate builds a more stable and reliable model that yields control to a baseline algorithm when its operating assumptions are not met.

## 1.6 Research Question

This study tests the viability of classical algorithmic time series forecasting against a modern machine learning approach for network flow throughput prediction to support dual-queue AQM congestion (flow) control. It was hypothesized that a machine learning model would surpass the accuracy of an autoregressive integrated moving average algorithm when predicting future network per-flow throughput as measured by the mean absolute difference between the actual and predicted values of two independent datasets created by sampling network traffic.

# Chapter 2

# Related Works

The studies that have particular weight and importance to this study are reviewed here. The first study entails a direct comparison of ARIMA and LSTM models as they relate to general time series forecasting. The following and most relevant study involves the prediction of per-user throughput in cellular networks using ARIMA and LSTM models. Lastly, a similar analysis to the previous study uses a DNN instead of an LSTM.

## 2.1 General Time Series Forecasting

In a paper by Siami-Namini et al. [16], the general performance of ARIMA, DNN, and LSTM forecasting models was tested. Siami-Namini et al. reported that their DNN implementation of time series forecasting acted randomly and did not converge on a particular amount of error. The study in this paper found similar behavior in DNNs, having randomness in training at low error rates. This phenomenon did not transfer to the testing set after the DNN finished training.

Siami-Namini et al. used historical data from financial and economic indices to train and test their models in their experimental setup. To judge the accuracy of their models, a root mean square error (RMSE) metric was used. This metric places greater importance on high error values while mean absolute error (MAE) provides the absolute unscaled magnitude of the error. The study in [16] uses financial and economic data where high levels of error are particularly undesirable. Siami-Namini et al. found that the LSTM produced about six times less error than the ARIMA forecasting algorithm.

### 2.1.1 Limitaitons of the Study

In [16] the scales of error are not put into perspective. The RMSE values lack any associated units to help the reader understand their significance. Along with a lack of scale, the paper does not contain any statistical analysis of the two prediction sets to see if they differ statistically. The reader is left to guess whether the decrease in error is significant and, if so, by what amount.

Further, the authors state an 84% to 87% reduction in error when migrating from ARIMA to the LSTM. This percent reduction in error can be a misleading metric. Take, for example, an RMSE of 0.01 on values that range in the millions. This should be considered a low error for any models at this scale. Now imagine a new model with an RMSE of 0.0001, which is a 99% reduction in error. The 99% can be misleading because, on a scale of millions, the difference between 0.01 and 0.0001 should be negligible except in extreme cases. The same case applies to [16] where the readers have no idea of the scales involved. A better metric would have been the decrease in percent error.

With the previously mentioned issues of the study in [16], the reduction in the errors reported may be significant both in scale and in terms of statistical difference.

The problem is that the reader is left to guess. If the reported error is assumed to be a significant reduction, there would be strong evidence that LSTMs are more capable than ARIMA in predicting financial and economic data. The reader may go one step further and incorrectly abstract this finding to all forecasting domains. For this reason, research and testing in all domain areas of time series forecasting are essential to expanding the understanding and differences between LSTM and ARIMA models.

## 2.2 ARIMA and LSTM

The study by He et al. in [7] is highly relevant to the current project. He et al. compares ARIMA and LSTM forecasting models, similar to the previous work discussed in [16]. These authors evaluated LSTM and ARIMA performance in predicting cellular encrypted user traffic throughput. He et al. found that the LSTM model was either on par or outperformed the ARIMA model. One intriguing aspect of the study was that the training and deployment context was the same. He et al. proposed using "on-line" training where the deployed machine learning model would first train on network data flowing through the network switch. At the same time, an algorithmic prediction approach handles the prediction decisions. When the machine learning model is mature enough (i.e., lower average absolute error than the algorithmic model), the model may take over traffic prediction. This "on-line" training approach solves a generalizability problem most machine learning models face when exposed to wide ranges and types of Internet traffic.

A critical difference between the work of He et al. and this study is the type of network flow being predicted. He et al. predicts throughput on a per-user basis, while this study predicts on a per-flow basis. The distinction is minor but has profound impacts on granularity and use cases of the prediction. User-level throughput prediction can be helpful in scaling the proportioned bandwidth in multiple-input and multiple-output (MIMO) cellular networks, for instance, a stadium hosting a large sporting event. Flow level throughput may be used for the same purpose. However, it can also support low latency networking because of its ability to distinguish queue building and non-queue building flows within a single user's network traffic. The limitation of user-level throughput prediction is that all users' network traffic must be considered equal. If a user is running a queue-building application, other applications they may be using at the time are not eligible for the low latency queue.

### 2.2.1 Limitations of the Study

The main limitation of [16] was briefly discussed by themselves: computational cost. The computational cost of training may be mitigated with several cloud-based training strategies. A computational cost that is more problematic to deal with is the exponential decrease in handleable network flows when dealing with time scales between 10 to 100 milliseconds. As the resolution of the time scale increases, so do the number of predictions needed in a second. An average execution time of 0.0001 seconds can support up to 10,000 predictions per second while only making 100 predictions in 10 milliseconds. The decrease in the number of network flows a 10ms prediction resolution can maintain discounts that model from any MIMO network as tens of thousands of users may be active at a time.

## 2.3 Deep Learning Throughput Prediction

In Zuo [19], a less common machine learning approach to time series forecasting was tested. Zuo [19] tested the prediction performance of a deep neural network on per-user throughput in cellular networks. Zuo also assesses the generalizability of

DNN models when presented with new networks and their traffic. ARIMA was not considered in Zuo's paper. Instead, support vector regression, LSTM, and random forests were compared to a DNN. All error was measured with MSE in megabytes.

The random forest machine learning algorithm obtained the highest error in all test datasets, followed by the support vector regression (SVR) model with a slightly better error. Interestingly the DNN and SVR scored almost the same error on both datasets. This error scored by the DNN and SVR was about 0.038MB less than the 0.64MB to 0.56MB of error from the random forest algorithm. Lastly, it was noted that the LSTM model was very unreliable during training and possibly required more data to become reliable during training. In the dataset the LSTM was capable of being trained on, the LSTM scored the lowest error rate at 0.48MB.

One of the study's research questions was whether the machine learning models were generalizable to other datasets. Zuo found that in the case of the chosen best model from the study, the DNN performed worse on the second dataset. This finding provides further evidence that the "on-line" train proposed in [7] may become a standard solution to generalizability.

# Chapter 3

# Methods and Implementation

A comprehensive overview of the methods and techniques used is outlined below. The topics of interest are the production of test data, model creation, training, prediction, and calculations used to judge the accuracy and generalizability of the models. Much consideration was put into providing both methods a fair and equal opportunity for reasonable performance by supporting their particular requirements. Therefore, this study created two unique datasets to provide a broader variation in network topography and application traffic for testing.

## 3.1 Network Traffic Collection

The study used two datasets for training and accuracy validation. These datasets were created by capturing network traffic on two separate and geographically disconnected networks. The perspective of the captured network traffic is that of a client performing various online activities. The length of time for which the study recorded network traffic varied depending on the main task performed while the capture was active. During each capture, user activities that generate network traffic were performed. These activities ranged from website browsing, video streaming, music streaming, online multiplayer games, and moderately sized ( 500MB) file uploads and downloads.

Network traffic was captured from a laptop and a desktop depending on the dataset in question. This was to add real-world variability to the datasets. Both machines were running the Windows 10 Operating system. Wireshark was the packet capture program used. No filters were applied to Wireshark, and all packets on the relevant Internet-facing network interface were captured. Once the data were captured and recorded, it was exported to a JSON file format where every packet had a corresponding entry. The JSON file was then stored and processed later in the study.

### 3.1.1 First Dataset

This dataset was recorded on a Windows 10 desktop with a 1 Gbps network interface. The maximum throughput of the network is about 800 Mbps downlink and 24 Mbps uplink. The traffic workload was kept relevant but straightforward in the first recorded dataset. The main application and traffic generator for this dataset was a competitive first-person shooter game called Apex Legends. Its game sessions consist of 59 other players grouped into teams of two or three. The players are spread across a large map and forced closer by an ever-shrinking cylinder. The goal is to be the last team left standing by the end of the game. In first-person shooters, low latency is an essential aspect of the game. The lower the game's latency, the fairer the gameplay will become. Apex Legends is considered a non-queue building application and would be a great candidate for using dual-queue AQM's fast queue.

The second application running during the recording of the first dataset was

Spotify. Spotify is a music streaming service that works in a similar buffered fashion to YouTube. Technically, Spotify is not considered a non-queue building application as it works in a bursty manner. The bursts of data that Spotify does generate are small compared to other internet traffic; however, the application's flow may be classified as non-queue building without much consequence. These two applications were recorded for about twenty minutes, or the length of an entire Apex Legends game.

### 3.1.2 Second Dataset

The second dataset was recorded on a Windows 10 laptop with a 600 Mbps WiFi interface card. The maximum throughput of the network is about 20 Mbps downlink and 3 Mbps uplink. The location from which this sample was taken is geographically separate from the United States and uses WiFi 2.4Ghz as its primary connection medium. This dataset's main application and traffic generator was a GitHub repository upload. The total upload payload size was around 350MB. During this upload website, browsing and Spotify streaming were also performed. In this scenario, the interesting aspect of the data is that both queue building and non-queue building workloads are present simultaneously. It is crucial to correctly predict the expected throughput of each workload to maintain low latency for the non-queue building applications.

## 3.2 Computational Workspace

Model training, validation, and accuracy testing were performed on the same desktop environment. The computer was running Windows 10 Pro. Visual Studio Code (VS Code) was the environment of preference for code development and as an execution interface. Within VS Code, a custom Python environment containing all necessary libraries runs the study's models. Python kernel version 3.8.5 was used for code execution. The main two libraries that support this work are Pytorch by Paszke [13], a machine learning framework library, and statsmodels by Seabold et al. [15], a statistical modeling library including time series forecasting algorithms.

The hardware specifications of the desktop are as follows.

- CPU: 12th Gen Intel(R) Core(TM) i7-12700KF 3.61 GHz
- Motherboard: ASUS PRIME Z690-P D4
- Memory: 2x 16GB G. Skill DDR4 SDRAM 4608MHz
- GPU: NVIDIA GeForce GTX 1080

## 3.3 Dataset Preprocessing

Much data reduction had to be done to make the collected datasets usable by the ARIMA and neural network models. The size of the capture files ranged from 1.5 gigabytes to 47 gigabytes. These captures have all possible information Wireshark could gather about the packet and some calculated metrics like time since the last packet. Neither ARIMA nor the neural networks need collected information like interface name, IP version, or packet checksums, so much of the captured packet information was discarded. On the first pass of the raw capture, every packet had seven items recorded and stored for further processing. Those items were Linux epoch time, frame length (bytes), network protocol, source address, destination address, source port, and destination port.

The recorded data were then filtered again using the last four collected items as a key to identify flows. Given the destination address and port and the source address and port, any packet can be grouped with its respective flow. Time series for each flow could then be generated as the whole timeline of each flow was grouped. Packets may have arrived out of order during collection, so their recorded Linux epoch times were used to reorder them correctly. During this reorder, each packet was put into an appropriate bin along the flows timeline. To clarify, along the timeline of a flow from zero to n seconds, a bin representing one second was placed for each second on the flows timeline. Within each bin, packets were discarded, and only their length and count were added to the total of each respective bin. At this point, the dataset contains everything needed to train and test every model discussed in this study.

Before feeding the datasets into the various prediction models of the study, the grouping of data flows was partially undone. This degrouping was performed because, in real-world workloads, flows are constantly being created and terminated. There is no reliance on flow identity as it reduces the model's generalizability. This partial degrouping does not affect ARIMA's performance as its training data is also the input sequence. When partially degrouping flows, segments of a flow's series must be maintained; otherwise, the information generated is random.

To preserve the segments of every flow, each flow was cropped into blocks of time. The study settled on 10-second blocks as it allowed smaller burst flows to contribute their data while maintaining enough data to train and test an accurate model. Five of the seconds in a block are used as the input sequence. One second is used as the prediction target, and the last four seconds are used as a buffer space for the next flow segment. A rolling window algorithm for increasing the training data was assessed, but the performance increase was marginal as enough training data already existed.

Next, every flow's throughput bin was in units of bytes. Due to the scale and variation of throughput measured in bytes, the machine learning models did not perform well. To remedy this problem, a simple multiplicative conversion of 1e$\hat{-}$6 was applied to each bin. The resulting units of each bin became in terms of megabytes.

Lastly, when the dataset was fully assembled, the minima and maxima were calculated to support the normalization of the input and output data for the machine learning models. The dataset inputs were normalized to be between zero and one (inclusive). The special exception to this normalization was the DNN model, which received normalized data between zero and ten. Pytorch's data loader class used the normalized data to split and shuffle the data into training and testing sets. These sets were then fed into each model (test set only for ARIMA) during training and evaluation.

## 3.4 ARIMA Setup and Parameters

ARIMA has only three parameters to consider, and the algorithm is well defined, so the setup is straightforward. Further simplifying things, the study used a Python library containing a premade version of the ARIMA algorithm. Since ARIMA is an algorithm, no implementation-specific parameters should affect performance. During testing, all parameters were iterated to find the best combination. By best combination, it is meant that the best produced the smallest error compared to other combinations of parameters. The computational cost of iterating through these parameters depends on the dataset, but the cost was acceptable for the two aforementioned datasets.

The first parameter, the lag order (number of trailing observations in the model), produced the best results when set to a value of one. A lag order of one was counterintuitive as trends are usually observed over more data. The differencing order is the second parameter the ARIMA model needs to create an accurate prediction. The differencing order is essentially the differencing (subtracting) of previous observations in the time series to attempt to make the series stationary [3]. Higher differencing orders are computationally more intensive, but the study found the best differencing order to be one. The last ARIMA parameter is the moving average window. This parameter controls the window size of the moving average. For the study, a value of zero was found to be the best window size. When the moving average window was more than zero, the model tended to be less accurate on burst traffic. The model performs best on the parameters ARIMA(1,1,0).

## 3.5 DNN Setup and Parameters

This study tests two machine learning approaches to the problem of throughput prediction. The first approach utilizes a deep neural network (DNN) built in Python using the Pytorch library. The DNN model, illustrated by Figure 3.1, contains four linear layers with three ReLUs in between each linear layer. All outputs are passed through a soft plus function at the end of the model. The first linear layer broadcasts the input sequence into 1024 neurons. These broadcasted values then flow through each layer of neurons and ReLUs, progressively being gathered into the soft plus function. The output from the neural network is then denormalized to undo the previous normalization in the data preprocessing stage.

Much tuning was done for the hyperparameters of the model to produce hyperparameters that yielded fast and accurate models. A mean absolute error criterion (a function to judge the model's progress during training) was chosen as this is also one of the metrics used to evaluate the accuracy of the models. The AdamW optimizer was selected for parameter tuning as it generally improves the widely accepted Adam algorithm [11]. The learning rate and weight decay used in AdamW were found to work best when set to 0.00001 and 0.001, respectively. During training, a learning rate scheduler was used to decay the learning rate of the AdamW optimizer by a rate of 0.98 (multiplicative decay; a smaller number means more decay) in every training epoch. The training epoch number was found to start overfitting the model past 100 epochs, so the epoch number was set to be 75. At the end of every epoch, the model was evaluated on the test set, and if it outperformed the best-saved model, it would become the new best-saved model. This model-saving method prevents models from being ruined by overfitting.

## 3.6 RNN Setup and Parameters

The study's second machine learning approach is a recurrent neural network (RNN) built in Python using the Pytorch library. Specifically, the RNN uses a recursion architecture (internal input-output loop) called long short-term memory (LSTM). LSTMs are favored in machine learning when attempting time series prediction; for this reason, the LSTM architecture was chosen as a candidate for comparison. The LSTM model, illustrated by Figure 3.1, contains two linear layers and two LSTM layers. Other than LSTM layers, an essential distinction between the RNN and DNN models is that the RNN model has no activation functions like ReLU or soft plus. When building and testing the RNN model, the study found that activation functions severely hinder the model's performance.

The RNN model's hyperparameters are similar to the DNN hyperparameters discussed previously. There is, however, one slight difference that is important to note. The RNN's learning rate was found to work best when set to 0.0001 instead of 0.00001. The reason is that the loss of the model does not converge to its minima fast enough when using the lower learning rate of the DNN.

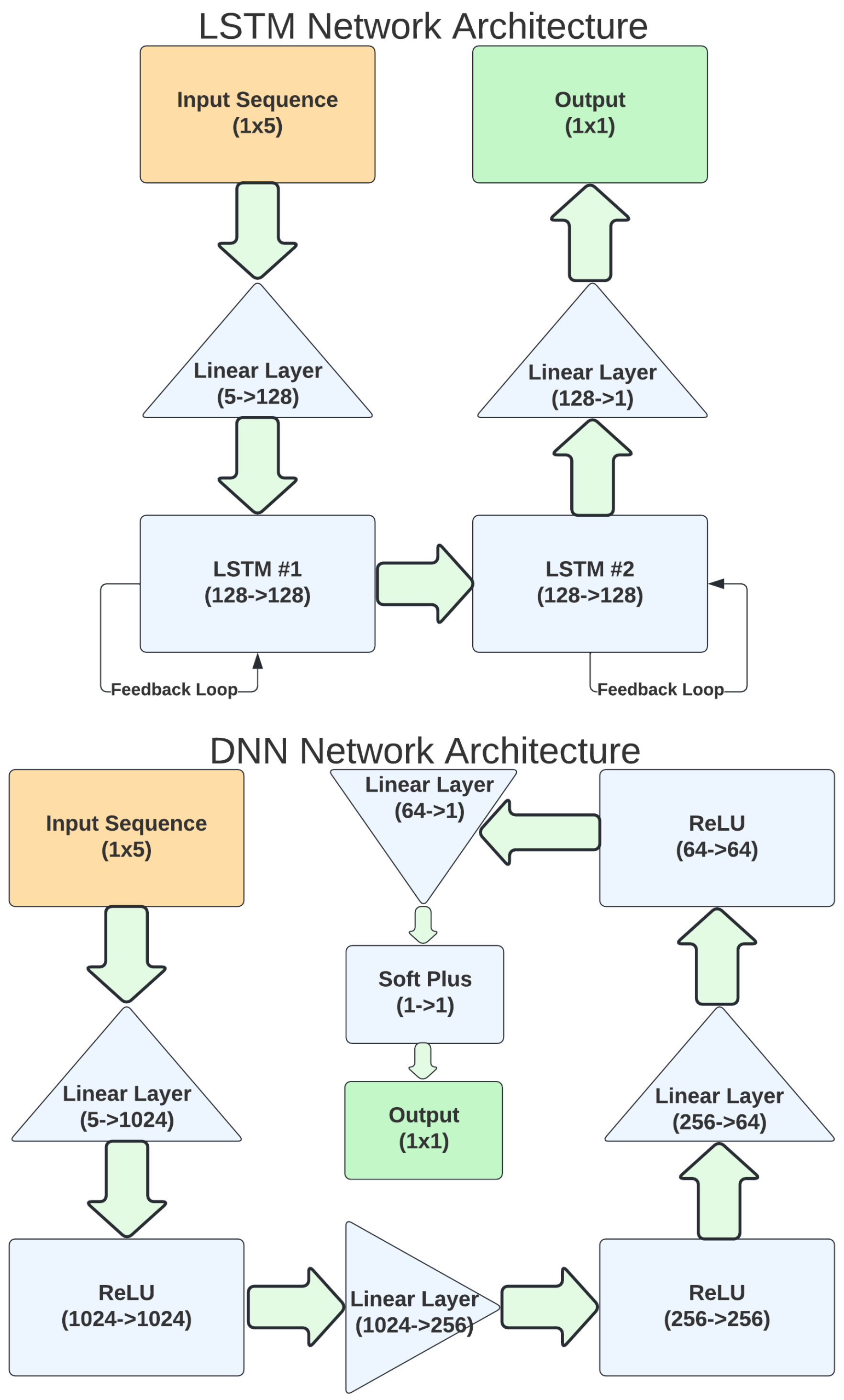

Figure 3.1: Diagram illustrating this study's LSTM (top) and RNN (bottom) architectures

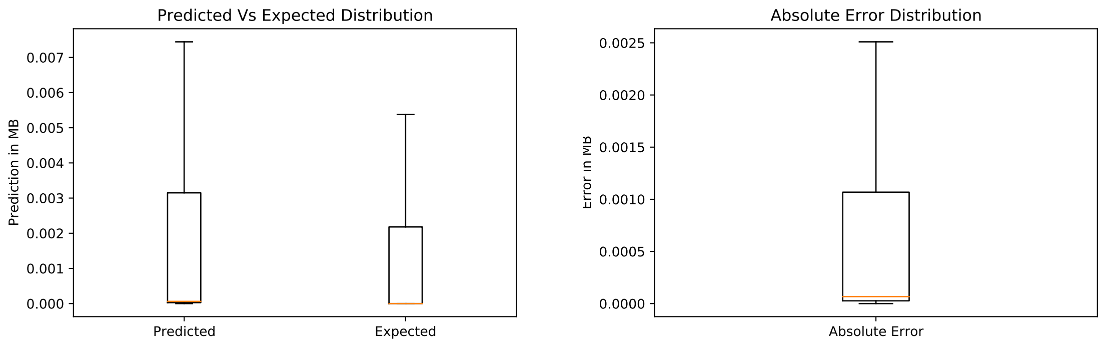

Figure 3.2: Box and whisker plots comparing the distributions of predicted and expected data (left), and a box and whisker plot of the distribution of error (right)

## 3.7 Accuracy and Validation

When testing the accuracy of the models, it was important that the models had never been fitted to or trained on the test dataset. This constraint was straightforward for the ARIMA model as the fitting process happens during testing, so it can be said that the ARIMA model has not seen the testing data before being tested. Separating the training and test sets required more work for neural network models. As the data preprocessing section mentioned, the Pytorch loader class was used to shuffle and split the given dataset into a training and test set. The split used in this study was 70% training data and 30% testing and validation data.

### 3.7.1 Testing

The models are first presented with the input data of the test set. They then iterate over all test inputs predicting the $6^t h$ second of each 5-second input. The results are denormalized and recorded with the target (actual) data. The mean absolute error (MAE) is calculated using the equation below. For the equation below, $p_i$ is the predicted value of the $i$th input sequence of the test set, and $e_i$ is the expected value of the $i$th input sequence of the test set.

$$MAE = \frac{\sum_{i=1}^{n} |p_i - e_i|}{n}$$

A general understanding of the model's performance can be obtained with the calculated MAE. Along with the MAE, two derived graphs are also created. An example of these graphs can be found in Figure 3.2. The first graph is two box and whisker plots comparing the distribution of expected data points with predicted data points. The second graph shows the absolute error distribution from the previously mentioned data. The maxima, minima, average, median, and standard deviation are gathered for all models. With these interpretations of a model's accuracy, the study can adequately assess the effectiveness of each model.

Lastly, the study applied two statistical analysis techniques to the output distributions produced by each model. The first technique used was the Pearson Correlation test applied to the expected (observed outcome) distributions for each dataset and their corresponding model prediction distributions. This test was used to judge the goodness of fit for each model without the subjective problems of evaluating the scale of error. Using the p-value produced by this correlation test in conjunction with the error comparison between each model, a better understanding of each model's performance was obtained.

### 3.7.2 Statistical Analysis

The first statistical analysis technique used is the coefficient of determination. This value, denoted as $r^2$, is a common measure of the goodness of fit of a model's predictions to their expected values. The statistic is derived from the Pearson Correlation test, which measures the correlation between two distributions mapped to x and y. The less correlation between the two values, the further away from the linear line $x = y$ a point will lie. The Pearson correlation coefficient $r$ is squared to obtain the coefficient of determination. The coefficient of determination multiplied by 100 gives a percentage of how much expected data the forecasting model was able to fit. In general, a higher coefficient of determination indicates a better fitting model for the data.

The second analysis technique used to support the assumption that the models predicted or performed differently was the analysis of variance (ANOVA). This test was applied between the three predicted distributions of each model within a dataset. This technique was repeated by the study for both datasets. When using the ANOVA test, the null hypothesis is that there is no statistically significant difference between the means of all distributions tested. An ANOVA test allows for the three prediction distributions from the models to be tested in a combined way to avoid interpreting the results from two or more comparisons of the distributions. The ANOVA test produces an F value with an associated p-value. That p-value may be analyzed as rejecting the null hypothesis to be true if p is below 0.05. When p is close to one, the distributions are very similar. A low ANOVA p score between all three prediction distributions is needed to support the claim that a model performs differently than the other two models.

## 3.8 Computational Delay

Computational delay or delay of results incurred by computation time is a crucial assessment metric. If a model can predict one second into the future with perfect accuracy but takes 2-seconds to produce that prediction, the model would be very remarkable but ultimately useless. The motivation for this study lies in the ability to make performance-enhancing decisions based on a prediction, so it stands to reason the forecast should be produced with enough time left over to take action based on that decision. In this study, the computational delay of a model is measured as the average time taken to predict. Given each sample from the test set, the time a model takes to predict will be collected and averaged. The study measures the average execution time of all models while they are running on the CPU.

Furthermore, the execution time of the machine learning models are measured while running on the GPU. This second GPU measurement is not meant to be compared with the algorithmic approach's CPU execution time. Instead, this metric should be used to understand how effective a machine learning model may be when used inside a data center or packet routing hub.

# Chapter 4

# Results

The methods discussed in this study were developed to answer specific questions of interest surrounding the performance of algorithmic and machine learning attempts to predict network flow throughput. In this chapter, the results of this study's methods are reviewed.

## 4.1 Execution Time

Table 4.1: Computational delay performance data

| **Model** | CPU Time | GPU Time | CPU Max Flows | GPU Max Flows |
|---|---|---|---|---|
| ARIMA | 0.000745 | NA | 1343 | NA |
| DNN | 0.000424 | 0.000101 | 2360 | 9930 |
| LSTM | 0.000519 | 0.000104 | 1927 | 9592 |

Table 4.1 provides data surrounding the average execution time of each model. It lists both CPU and GPU execution times along with a rough estimate of maximum supportable flows for each model. All models were found to have an average execution time ranging between 0.000101 and 0.000745 seconds. This range is about a seven

times increase from best to worst. The range also accounts for GPU execution times. GPU execution times were tightly grouped with a difference of 3.54452E-06 seconds. The average CPU execution results are more distributed, with ARIMA having the slowest execution time at 0.000745 seconds and DNN having the fastest execution time at 0.000424 seconds. The difference between the slowest and fastest CPU execution time was about 0.000321 seconds. These execution times are for predicting a single input sequence or flow.

Dividing one second by the average execution time of each model gives a rough estimate of the maximum amount of concurrent flows a model can support with this study's hardware and environment setup. The most amount of flows capable of being supported by the GPU is 9930 and 2360 on a CPU. ARIMA specifically can support 1343 concurrent flows.

A hardware monitoring application, Performance Monitor (built into Windows 10), measured CPU utilization during testing. CPU utilization during DNN and RNN testing was 95.736% on core zero, while core one averaged 91.559% utilization. The neural networks utilized no other CPU cores. GPU utilization, measured with MSI Afterburner, averaged 40% utilization. ARIMA is a CPU-bound application in this study's setup and was measured to have 89.710% and 100% core utilization on cores zero and one, respectively.

## 4.2 Review of Datasets

As a reminder, the first dataset in this study contains the network traffic data associated with playing a multiplayer game and streaming music. Music streaming, file upload, and web browsing were all occurring in the second dataset.

The characteristics of each dataset may be found in Table 4.2. The maxima,

Table 4.2: Characteristics of datasets

| **Datasets** | **Max** | **Min** | **Mean** | **Std** | **Median** |
|---|---|---|---|---|---|
| First | 0.066193 | 0.000000 | 0.011399 | 0.014994 | 0.000000 |
| Second | 0.348852 | 0.000000 | 0.035227 | 0.092890 | 0.000000 |

minima, average, standard deviation, and median are listed for each dataset. The first dataset can be said to have the smallest scale, while the second dataset is progressively bigger in scale. The maxima of each dataset increases by an order of magnitude of one. The maximum order of magnitude of the first dataset is negative one, while the second dataset is zero. All datasets share a minimum of zero. The median of the two datasets is zero. Finally, the datasets' average and standard deviation follow a similar trend to the maxima column.

## 4.3 First Dataset Predictions

Table 4.3: Characteristics of model prediction on the first dataset

| **Model** | **Max** | **Min** | **Mean** | **Std** | **Median** |
|---|---|---|---|---|---|
| ARIMA | 0.073655 | 0.000000 | 0.011206 | 0.014877 | 0.000000 |
| DNN | 0.061891 | 0.000038 | 0.011101 | 0.014252 | 0.000046 |
| LSTM | 0.066120 | 0.000127 | 0.012046 | 0.015330 | 0.000364 |
| Dataset 1 | 0.066193 | 0.000000 | 0.011399 | 0.014994 | 0.000000 |

Table 4.3 contains the characteristics of the prediction set produced by each model using the first dataset as input. Differencing Table 4.3 with the expected set's characteristics in 4.2, a few behaviors of each model are shown. Within the Max column, ARIMA produced a significant over-prediction. At the same time, the DNN underpredicted the maximum expected value, and the RNN very closely (within 0.000073MB of error) predicted the maximum value of the expected set. The

DNN and RNN overpredicted the expected set's minimum, while ARIMA perfectly predicted the minimum at zero. The average produced by all models was relatively close to the true average, with ARIMA being the closest and the RNN being farthest from the true average. The standard deviation behaved similarly to the average, but the DNN was farthest from the true standard deviation. The median and minimum are the same in dataset one, so the models performed similarly between these two metrics.

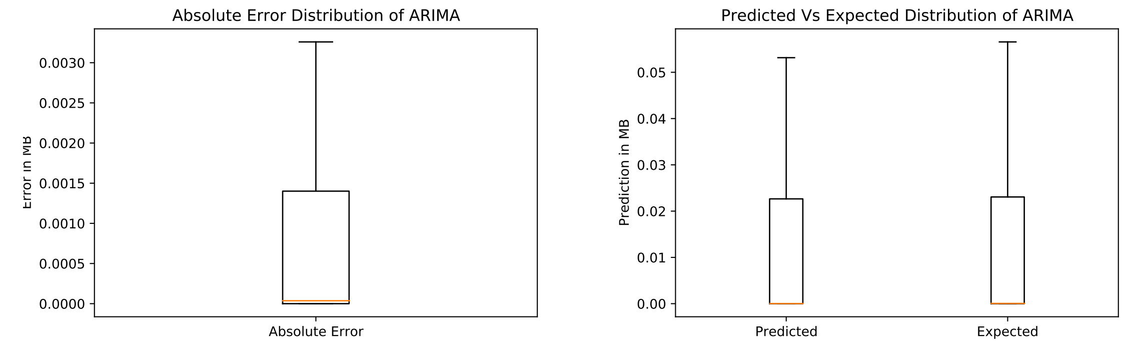

Figure 4.1: ARIMA on dataset 1, absolute error distribution (left) and predicted v. expected distribution (right)

The box and whisker plots for each model give further details and interpretable information. On the left side in Figure 4.1, the absolute error distribution is shown for ARIMA on dataset one. The upper quartile range does not exceed 0.0015MB of error, and the upper extreme is at 0.0030MB of error. The box plot agrees with the reported **MAE of 0.00101MB**. On the right in 4.1 are the distributions of both the predicted and expected data. Both plots look identical with some minor differences at the upper extremes.

Next in dataset one is the DNN found in Figure 4.2. On the left side in the figure, the absolute error distribution looks similar to the ARIMA error plot. However, the median and lower quartile do not lie as close to zero as in the ARIMA

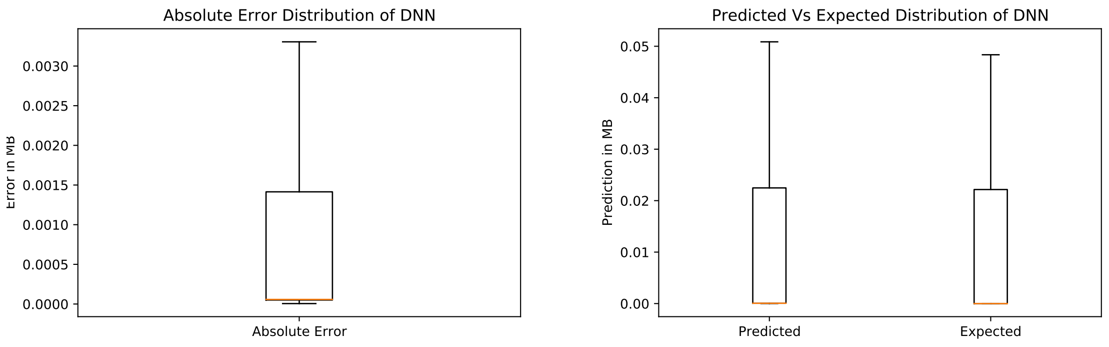

Figure 4.2: DNN on dataset 1, absolute error distribution (left) and predicted v. expected distribution (right)

plot. The MAE of the DNN confirms this with a reported **MAE of 0.00114MB**. On the right in 4.2 are the distributions of both the predicted and expected data. Both plots look identical with some minor differences at the upper extremes. This is similar to the ARIMA model.

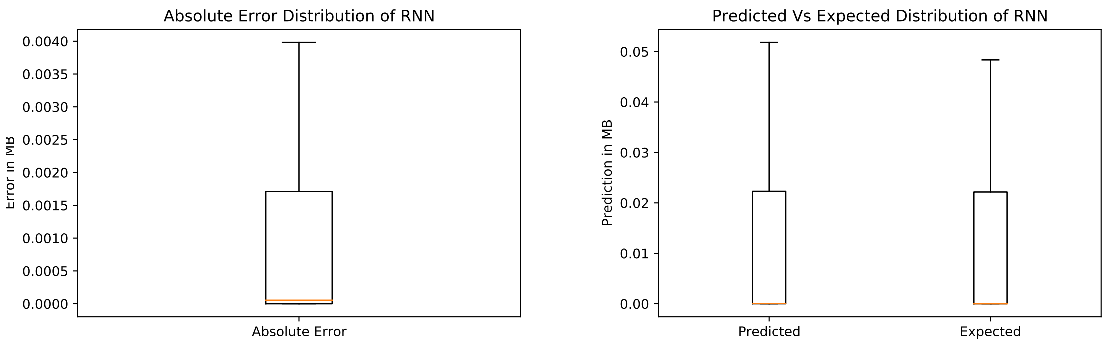

Figure 4.3: RNN on dataset 1, absolute error distribution (left) and predicted v. expected distribution (right)

Last in dataset one is the RNN found in Figure 4.3. On the left side in the figure, the absolute error distribution looks similar to both the ARIMA and DNN error plots. The median and lower quartile do not lie as close to zero as the ARIMA plot. The MAE of the RNN affirms this with a reported **MAE of 0.00168MB**. On the right in 4.2 are the distributions of both the predicted and expected data. Both plots look identical with some minor differences at the upper extremes. Removing outliers from dataset one does not produce any significant change in the MAE of all three models.

## 4.4 Second Dataset Predictions

Table 4.4: Characteristics of model prediction on the second dataset

| **Model** | **Max** | **Min** | **Mean** | **Std** | **Median** |
|---|---|---|---|---|---|
| ARIMA | 0.561002 | 0.000000 | 0.038412 | 0.100727 | 0.000002 |
| DNN | 0.349408 | 0.000028 | 0.033331 | 0.093964 | 0.000028 |
| LSTM | 0.066120 | 0.000127 | 0.012046 | 0.015330 | 0.000364 |
| Dataset 2 | 0.348852 | 0.000000 | 0.035227 | 0.092890 | 0.000000 |

Table 4.4 contains the characteristics of the prediction set produced by each model using the second dataset as input. Differencing Table 4.4 with the expected set's characteristics from 4.2, some behaviors of each model are shown. ARIMA produced a significant over-prediction within the Max column similar to its performance in dataset one. The DNN was the closest (within 0.000556MB of error) to the true maximum expected value. The RNN significantly underpredicted the maximum value of the expected set. The DNN and RNN overpredicted the expected set's minimum by a negligible amount (almost the same values as dataset one). ARIMA perfectly predicted the minimum at zero. The average produced by the ARIMA and DNN models was relatively close to the true average, with DNN being the closest. The RNN was farthest from the true average by 0.023181MB. The standard deviation had ARIMA and the DNN close. The RNN was farthest from the true standard deviation by one order of magnitude. The median and minimum are the same in dataset two,

so the models performed similarly between these two metrics.

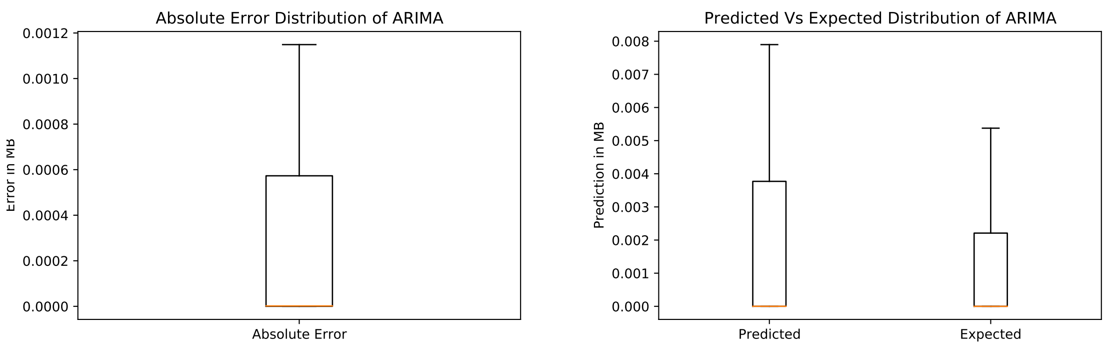

Figure 4.4: ARIMA on dataset 2, absolute error distribution (left) and predicted v. expected distribution (right)

On the left side in Figure 4.4, the upper quartile range does not exceed 0.0006MB of error, and the upper extreme does not pass 0.0012MB of error. The **box plot does not agree** with the reported **MAE of 0.006933MB**. This is due to extreme outliers (not displayed on the plot) shifting the mean error higher than if the outcome was binary. The box plot's upper quartile better measures ARIMA's prediction capabilities in this instance. On the right, in Figure 4.4 ARIMA's upper extreme is 0.002MB higher than the expected upper extreme. This is another indicator that outliers in the dataset and ARIMA's prediction set caused a skewed representation of its error.

On the left side in Figure 4.5, the upper quartile range does not exceed 0.00075MB of error, and the upper extreme does not pass 0.00175MB of error. The **box plot does not agree** with the reported **MAE of 0.004836MB**. This is again due to extreme outliers shifting the mean error by a disproportional amount. The box plot's upper quartile better measures the DNN's prediction capabilities in this instance. On the right, in Figure 4.5, the DNN's upper extreme is about 0.002MB higher than the expected upper extreme. This indicates that outliers in the dataset

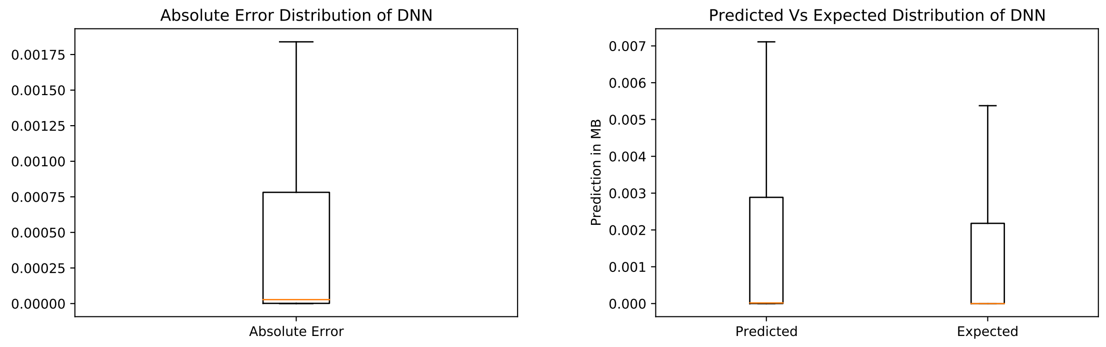

Figure 4.5: DNN on dataset 2, absolute error distribution (left) and predicted v. expected distribution (right)

and DNN's prediction set caused a skewed representation of its error.

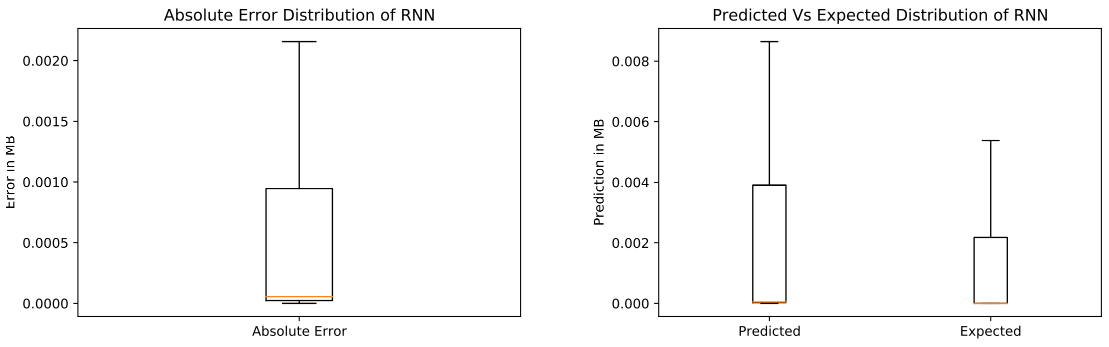

Figure 4.6: RNN on dataset 2, absolute error distribution (left) and predicted v. expected distribution (right)

Finally, on the left side in Figure 4.6, the upper quartile range is about 0.001MB of error, and the upper extreme is just over 0.002MB of error. The **box plot does not agree** with the reported **MAE of 0.004095MB**. This is from extreme outliers shifting the mean error by a disproportional amount. The box plot's upper quartile better measures the RNN's prediction capabilities in this instance. On the right, in Figure 4.6, the RNN's upper extreme is about 0.008MB higher than the expected upper extreme. The upper extreme of 0.008MB is the largest of the three models. This indicates that outliers in the dataset used to test and train the RNN have caused a skewed representation of its error.

### 4.4.1 Outliers

When outliers are filtered out of the input training and test data (the expected set is not filtered directly) using the interquartile range method, the RNN and DNN models outperform the ARIMA model. When filtering the input for outliers, the ARIMA model's MAE is unchanged compared to when outliers exist in the input data. When comparing the filtered RNN and DNN MAEs to the ARIMA, the new filtered RNN and DNN have better performance. All MAEs produced using the filtering methods discussed are in the list below.

- ARIMA Unfiltered Dataset Two MAE: 0.006933MB
- DNN Unfiltered Dataset Two MAE: 0.004836MB
- RNN Unfiltered Dataset Two MAE: 0.004095MB
- ARIMA Filtered Dataset Two MAE: 0.006932MB
- DNN Filtered Dataset Two MAE: 0.000364MB
- RNN Filtered Dataset Two MAE: 0.000513MB

## 4.5 Statistical Tests

### 4.5.1 Coefficient of Determination

Table 4.5: Coefficient of determination, $r^2$ values, for assessment of the goodness-of-fit between predicted and expected distributions

| **Dataset** | **RNN** | **DNN** | **ARIMA** |
|---|---|---|---|
| **1** | 0.970928 | 0.976824 | 0.983093 |
| **2** | 0.758740 | 0.963230 | 0.855665 |

In Table 4.5 the coefficient of determination, calculated by squaring the Pearson correlation coefficient r values for each model's prediction distribution and the associated dataset's expected distribution, are listed. In this study, the coefficient of determination is used to measure goodness-of-fit for each model. The value is abstracted away from metrics like MAE, so the interpretation is less subjective when defining what good and bad errors are considered. In dataset one, all three models received a $r^2$ score between 0.97 and 0.98. These scores heavily support the idea that all models fit dataset one well. The RNN scored the lowest $r^2$ value in dataset two at 0.759. This statistic should not discount the RNN as a well-fitting model for dataset two as it scored the best in terms of MAE. The discussion section will discuss further reasons for this low $r^2$ score, but it is essential due to the training set size. ARIMA also scored lower with a $r^2$ score of 0.856. Taking into account that the RNN with the lowest MAE and ARIMA with the highest MAE in dataset two both scored lower than the DNN, it can be interpreted that all three $r^2$ scores are considered good fits.

### 4.5.2 Analysis of Variance

Finally, when applying an ANOVA test between all model prediction distributions for dataset one, a **p-value of 0.983331** was scored. This p-value fails to reject the null hypothesis that any predicted distributions are statistically different for models predicting on dataset one input. In dataset two, the same ANOVA analysis was performed and produced a **p-value of 0.994318**. Again, this p-value fails to reject the null hypothesis that any predicted distributions are statistically different for models predicting on dataset two input.

# Chapter 5

# Conclusions and Discussion

The Internet has a growing need for low latency networking. Dual-queue AQM is a technique that was developed to support low latency networking. For dual-queue AQM to be truly useful on the open Internet, it requires accurate prediction of future flow throughput. Machine learning and algorithmic time series forecasting are the two most popular approaches to solving this problem. With the popularity of machine learning compared to traditional algorithmic approaches, the former technique is often used without empirical support. This study aimed to address the lack of empirical evidence by testing the efficacy of machine learning against classical algorithmic time series forecasting techniques for network per-flow throughput prediction to support dual-queue AQM congestion (flow) control. It was hypothesized that deep and recurrent neural networks would surpass the accuracy of an autoregressive integrated moving average algorithm when predicting future network per-flow throughput as measured by the mean absolute difference between the actual and predicted values of two independent datasets created by sampling network traffic.

## 5.1 Answering the Research Question

### 5.1.1 Statistical Findings

According to the coefficient of determination used to judge the goodness of fit, all models fit well to their associated expected distributions. This goodness of fit means that when encountering new but similar data to that of the test or train sets, the models have a good chance of predicting the new expected throughput with low error. All models had negligible differences in error on the first dataset. The machine learning approaches had a slightly better MAE on dataset two.

An ANOVA test was performed on the models ' prediction distributions because of the minor differences in error between the models on dataset two. This test was to see if there were any statistically significant differences in the means of the distributions. The ANOVA test failed to reject the null hypothesis that there is no difference in means for both datasets. This failure to reject the null hypothesis means no significant difference between the predicted distributions exists. A finding of no significant difference between the prediction distributions means that **there is no evidence to support the hypothesis** that machine learning is more accurate at predicting per-flow network throughput.

## 5.2 Levels of Error

An example is helpful to put the scale and amount of error occurring in these models into perspective. In this example, a small packet switch is placed somewhere on the open Internet. This packet switch uses a dual-queue AQM with one of the models tested in this paper. This dual-queue is actively supporting low latency flows. The low latency queue can support a threshold of 1MB/s of combined throughput from all of its low latency flows. If this threshold is broken, the queue cannot guarantee low latency times for all flows it contains. During operation, it experiences 80% threshold utilization caused by 80 flows, each averaging 0.01MB/s of throughput. The model that is predicting for this dual-queue AQM has an MAE of 0.001MB (a finding similar to the current study). To cause problematic slow down for flows inside the low latency queue, a simultaneous underprediction of 0.001MB by the model would have to occur for over 200 flows. This is a highly improbable event and would be required to continuously occur for an extended network slowdown.

## 5.3 Interesting Findings

### 5.3.1 ARIMA

The ARIMA algorithm had slightly more error when predicting on dataset two. An explanation for this requires a look at dataset two. In dataset two, both upload and download flows were occurring simultaneously. It is possible that machine learning algorithms like LSTM can recognize and distinguish upload and download flows from each other. This recognition could likely lead to slightly better performance by tailoring the model's weights to predict differently based on the flow type. This behavior leads to the topic of input features discussed in the limitations section.

### 5.3.2 Outliers and Datasets

Dataset one contained very few outliers, and because of that, the resulting MAEs from each model were also agreed with the visual box and whisker plots. Specifically, all MAEs were well within the bounds of the upper quartile. This is interpreted to mean that no extreme single data point error values significantly increased the MAE of the models.

In dataset two, this was not the case. Dataset two contained many extreme outliers. When predicting on dataset two, all models experienced MAEs well outside the upper extreme of their respective box and whisker plots. This indicated that the extreme outliers increased the MAEs by a disproportional amount. When filtering by interquartile range multiplied by 1.5, the machine learning models had a significant reduction in error. Interestingly, the ARIMA model's error did not change significantly. One explanation is that ARIMA is not affected by outliers like the DNN and RNN models. The problem lies within every point of error being given weight equal to its magnitude. The amount of error for a single data point should only be so significant. A further explanation of this phenomenon can be found in the limitations section, with a proposed solution to the problem.

### 5.3.3 Execution Time

As shown in the results section, CPU execution times for all models were fast enough to support over a thousand flows a second. If needed, this process could be parallelized to any number of CPU cores to increase performance when the number of incoming and outgoing flows increases. Alternatively, in critical infrastructure where the number of flows exceeds the limits of CPU-bound computation, a GPU may be used to support close to ten thousand flows. GPU utilization during testing showed there was possible room for a second machine learning model to run in parallel, giving a theoretical maximum of close to twenty thousand flows.

At this time, the GPU implementation of ARIMA available on Python is slower than its CPU counterpart when predicting single time series. It would be interesting to test the batch prediction capabilities of both ARIMA and the machine learning models on the GPU. Another interesting study would be to test the capabilities of each model when using hardware like field-programmable gate arrays (FPGAs) or Nvidia Jetson.

## 5.4 Limitations

### 5.4.1 Dataset Collection

When collecting the datasets for this study using Wireshark, some issues were encountered. Wireshark, when capturing packets, stores the whole packet in main memory. The storage methods Wireshark utilizes are not efficient, and it takes more than the size of a packet to store the packet. This inefficient storage results in main memory running out faster than the amount of data being recorded. For short or low throughput captures, this is not a problem. The system's main memory would run out in just a few minutes when attempting to record high throughput traffic. This resulted in very short captures for high throughput traffic. The resulting failed captures had many outliers compared to datasets one and two. The number of outliers in high throughput captures decreases the longer the capture is allowed to run. In a repeat of this study, it would be better to use a Linux interface with tcpdump or a similar program.

### 5.4.2 Data Features

Another limitation of the study involves dataset features. The datasets were made to be univariate time series to maintain an "apples to apples" comparison of the prediction methods. This was done because ARIMA cannot take additional input features. In a different scenario where DNNs and RNNs are given more input features, like the average time between packets in a flow or the type of protocol a flow was using, the machine learning approaches may outperform multi-feature forecasting algorithms.

### 5.4.3 Measuring Effective Error

The last limitation experienced in this study is the previously mentioned issue with assigning the weight of an error based on the magnitude. The magnitude of prediction error only matters up to a certain point. Also, the type of error, underprediction or overprediction, should be weighted differently. An overprediction only affects a single flow, while an underprediction affects all flows. Further, the magnitude of an over- or underprediction should only matter until the flow is kicked out of the low latency queue or the low latency queue experiences increased latency. Any error penalty past these points is misleading and skewing the amount of average error in the upwards direction. Zuo proposed a solution to this problem [19] where a piecewise function was used to control the threshold of the maximum acceptable error for under and over predictions, essentially bounding the error function.

## 5.5 Implications

### 5.5.1 Future Research

As mentioned previously, the scale of error is hard to assess. A more definitive evaluation of model error would be changing the throughput prediction problem into a classification problem. This can be done by creating a threshold of throughput. Flows below this threshold are considered low latency flows, and flows above this threshold are considered classical flows. In this setup, there is less information to process and evaluate as the models directly tell the dual-queue AQM to which queue to send packets. The measuring of error also becomes more straightforward as accuracy can be measured as a binary outcome for each data point.

Further studies may be done with a multi-feature forecasting algorithm compared against similar DNN and RNN with extra features as input. One of the main selling points of machine learning algorithms is their ability to process multi-feature data and find obscure correlations in data to produce better results. Additionally, reproducing this study on datasets with more variability and higher throughput levels may show more variable error and influence the behavior of the models and thus yield different results.

### 5.5.2 Recommendations

The algorithmic model took one day of setup and tuning to see the performance metrics stated in this study, while the machine learning approaches took about five days each to achieve similar performance. When the development time is critical, and the requirements for accuracy are not strict, an algorithmic approach has a much smaller cost of failure as compared with machine learning models.

When deploying prediction models to the field, it is best to use the concept of "on-line" training suggested by He et al. [7]. This training scheme allows for heavy optimization of a model to the type of flows it will encounter during operation. Generalizability is a problem for models needing to be deployed to a wide range of applications. "On-line" training solves this generalizability issue by training the model on the traffic it is deployed to predict. It is unnecessary to test generalizability if a model can be trained on the data it encounters.

Based on this study's findings, the data obtained recommends that an algorithmic approach should be attempted before using machine learning if the maximum amount of acceptable error is known. If an algorithmic approach fails to achieve the required performance and accuracy, a machine learning approach may be considered for a better fit. Another specific advantage of the algorithmic approach being used first is because of the small amount of development and tuning needed compared with machine learning approaches.

# Bibliography

[1] Rahul Amin, France Jackson, Juan E. Gilbert, Jim Martin, and Terry Shaw. Assessing the impact of latency and jitter on the perceived quality of call of duty modern warfare 2. In Masaaki Kurosu, editor, *Human-Computer Interaction. Users and Contexts of Use*, pages 97–106, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[2] Dejene BoruOljira, Karl-Johan Grinnemo, Anna Brunstrom, and Javid Taheri. Validating the sharing behavior and latency characteristics of the l4s architecture. *ACM SIGCOMM Computer Communication Review*, 50(2):37–44, 2020.

[3] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

[4] Bob Briscoe, KD Schepper, Marcelo Bagnulo, and Greg White. Low latency, low loss, scalable throughput (l4s) internet service: Architecture. *Internet Engineering Task Force, Internet-Draft draft-ietf-tsvwg-l4s-arch-01*, 2018.

[5] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Controlling queuing delays for real-time communication: the interplay of e2e and aqm algorithms. *ACM SIGCOMM Computer Communication Review*, 46(3):1–7, 2018.

[6] Liang Guo and I. Matta. The war between mice and elephants. In *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pages 180–188, 2001.

[7] Qing He, Georgios P. Koudouridis, and György Dán. A comparison of machine and statistical time series learning for encrypted traffic prediction. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 714–718, 2020.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[9] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA, 1988. Association for Computing Machinery.

[10] James F.. Kurose and Keith W Ross. *Computer Networking: A top-down approach*. Pearson., 2021.

[11] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[12] Akbar Majidi, Nazila Jahanbakhsh, Xiaofeng Gao, Jiaqi Zheng, and Guihai Chen. Dc-ecn: A machine-learning based dynamic threshold control scheme for ecn marking in dcn. *Computer Communications*, 150:334–345, 2020.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[14] Kadangode Ramakrishnan and Sally Floyd. A proposal to add explicit congestion notification (ecn) to ip. Technical report, RFC 2481, January, 1999.

[15] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[16] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

[17] Greg White, Karthik Sundaresan, and Bob Briscoe. Low latency docsis: Technology overview. *Research & Development*, 2019.

[18] Hong Zhong, Jinshan Xu, Jie Cui, Xiuwen Sun, Chengjie Gu, and Lu Liu. Prediction-based dual-weight switch migration scheme for sdn load balancing. *Computer Networks*, 205:108749, 2022.

[19] Xuechen Zuo. A deep learning approach to downlink user throughput prediction in cellular networks, 2020.