Bachelor's Thesis Computing Science

Radboud University Nijmegen

Classifying TCP loss causes to optimize throughput

Author:
Danny Swart
s1083428

First supervisor/assessor:
dr. I.G. Bucur

Second assessor:
dr. ir. X.J.G. de Carné de Carnavalet

January 11, 2026

Abstract

TCP is the backbone of reliable connectivity. Without it, it is impossible to ensure that the data we send on our devices actually gets delivered to the recipient. However, a flaw is that it throttles throughput in the case of wirelessly lost packets. A wireless loss is considered a congestive loss, which means that the system believes the network is congested. This is because congestive and non-congestive losses are evaluated as one and the same in TCP. In this paper, we investigate the usage of machine learning classifiers to determine the cause of packet losses. By simulating various network conditions using ns-3, a network simulator, we produced a large dataset that reflects real network conditions. Then, by training various classifiers on this data and testing their performance, we have found that there is promise in using classifiers to improve TCP. Furthermore, we have implemented such a classifier in a running simulation to see whether its usage results in a improvement in throughput.

Contents

Chapter 1

Introduction

Internet usage is becoming extremely abundant in everyday life. In a paper by E. Kirci et al.[1] it is suggested that there is a large increase in internet traffic in recent years. It is also reported that people spend more and more time on their phone[2], and much of that time is likely to be spent on a service that requires internet connectivity. With so much data transfer going on in every moment of our lives, it is essential that these processes happen as efficiently as possible. Every delay could be discomforting for a user, costly for a professional, or wasteful to the environment.

This shows how important it is for our devices to work smarter and more efficiently. To reliably send data to another person, devices use the Transmission Control Protocol (TCP). In broad terms, this protocol governs the way data is reliably sent between two applications on different networks. Introduced more than 50 years ago, it is still in broad use today.

Considering its constant and widespread use, finding any improvement to it can be beneficial. One known issue facing it is that its throughput or sending rate is reduced when it thinks a packet has been lost regardless of the underlying reason. In such instances, the protocol determines that the network through which the data traverses is congested, which does not need to be the case. The protocol does not consider how a packet gets lost, which could be important information to determine whether to reduce this throughput or not.

This thesis presents evidence that suggests that classifiers can be used in real-time to deduce the origin of a lost packet to better inform the decision whether or not to reduce throughput during data transfer. It is organized as follows:

Chapter 2, Related Work: Highlights different works that are similar to this one and how they compare.

Chapter 3, Preliminaries: An overview of the knowledge required to understand this paper and our conducted experiments.

Chapter 4, Research: This chapter goes in-depth into what this research tries to accomplish, how our data is generated, how this data looks, and what tools are used.

Chapter 5, Results: Presents the performance of our machine learning classifiers. It shows various confusion matrices, along with some metrics to understand how well and how quickly the classifiers perform. Also shows the performance of our best performing classifier running in a simulation.

Chapter 6, Discussion: An analysis of our results, with further elaboration on limitations of the classifiers and the implications of its use in a real-world system.

Chapter 2

Related Work

Similar experiments have been performed that attempt to classify TCP losses. In the following sections, three papers that perform a similar experiment will be discussed, with their similarities and differences with our work highlighted.

2.1 Enhancement of TCP over wired/wireless networks with packet loss classifiers inferred by supervised learning

I. El Khayat et al.[3] performed an experiment where classifiers were trained to determine packet loss causes in a simulated network using ns-2. The usage of the simulator allowed them to label the two types of packet loss, non-congestive loss and congestive loss. The metrics used to inform their classifier are the one-way delay and the inter-arrival time collected on the sender's side. As inputs for their classifier, functions of the data collected from the simulator were used. For example, a classifying rule is whether the maximum inter-arrival times of the succeeding packets divided by the average inter-arrival times of the packets received during a set duration is less than a constant. This was done to make sure the model is not dependent on the absolute values of these metrics, such that it is as independent of the particular network conditions as possible. In addition, the paper discussed TCP-friendliness. It is a measure of how much a protocol affects the efficiency of other competing protocols in the same network. An example of an unfriendly protocol is when two clients send data over the same network but the client using the unfriendly protocol utilizes much more of the available bandwidth, even though both clients are capable of utilizing the same amount of bandwidth. It is important to keep this metric in mind when proposing new protocols as it is very undesirable for a new protocol to have negative effects on other flows it is competing with. In our proposed protocol, we will consider the TCP-friendliness.

2.2 Congestion or No Congestion: Packet Loss Identification and Prediction Using Machine Learning

I. Ali et al.[4] performed an experiment that focuses on classifying TCP loss causes. The goal was to find out which machine learning model is most capable of making these classifications and which metrics are most important as input. The metrics used for training were Round-Trip Time (RTT), congestion window (cWnd) and jitter. RTT is the time it takes for a sent packet to receive a corresponding acknowledgment (ACK) from the receiver, cWnd is the amount of data that the sender allows to be in-transit before an ACK is received and jitter is the variation in arrival time for a sent packet. The paper extensively tested different ML models to see which one performed the best. After computing the recall and F1-score for each of the classifiers, the paper concluded that their Random Forest and k-NN classifiers were the most effective. Following this, they performed an ablation study to find out what the relative importance was of the included features. This step showed that the congestion window is the most important feature, followed by RTT and lastly jitter. Based on these findings, we opted to mainly focus on RTT and the congestion window as main inputs for our classifiers. Compared to this work, this paper differs in which machine learning classifiers were used, and how the metrics from the sender were used as input.

2.3 End-to-end differentiation of congestion and wireless losses

To highlight a work that differs significantly in the approach to minimizing throughput loss, the following paper is discussed. S. Cen et al.[5] focus on finding algorithms that use some of the same metrics as the other two papers. Ultimately, the considerations these algorithms make are similar to the rules the classifiers in our work or those in the other papers generate. These algorithms are referred to as end-to-end loss differentiation algorithms (LDAs). By understanding the underlying procedures of TCP these algorithms are created to be used to determine whether or not to throttle throughput. These rules have parameters that can be tuned to reduce loss cause misclassification. This differs with our approach as our rules are generated from the classifiers that use our simulator data.

Chapter 3

Preliminaries

3.1 TCP

The Transmission Control Protocol (TCP) is a communication protocol that facilitates the exchange of messages between devices over networks. It is connection-oriented, meaning that sender and receiver stay in contact, with the goal of reliable connectivity. In this context, reliability means the following things. Packets will get delivered, data integrity gets ensured, and the packets arrive in-order.

The certainty that data gets delivered is achieved by using acknowledgments (ACKs). Each sent packet is expected to have a corresponding ACK, signed with the sequence number of the packet that was originally sent.

A recipient can indicate that they have not received the expected data by re-sending previous acknowledgments. This is known as a duplicate acknowledgment (DUPACK). If the sender receives multiple of these, then it re-sends the missing data. It is also possible that the sender receives no ACK for some previously sent packet at all within an expected timeframe, and in this case it can also re-send this data.

Data integrity is achieved by attaching a checksum to each sent packet. The sender calculates the checksum as a function of its payload. Then, the receiver can perform this same calculation to check whether it gets the same result as the sender included. If it is equal, then the data was unchanged during delivery, and if it is unequal, then it is corrupted. In the data is corrupted, the same mechanism described previously will trigger, in which the corrupt data gets re-sent by the sender.

Lastly, in-order delivery is ensured by attaching sequence numbers to every packet. The receiver can know the order of the incoming packets by arranging them according to these. If there are unexpected jumps, then the receiver can repeat previous ACKs to let the sender know that some data has gone missing. In later versions of the protocol Selective Acknowledgments (SACK) were introduced. This allows the recipient to send a window of sequence numbers corresponding to acknowledged packets such that the sender knows not to re-send more than it needs to.

3.1.1 TCP Congestion Control States

The protocol features ways to control the level of congestion in a flow. A flow is defined as a connection between two clients running TCP. This could span many different hops in-between. The main mechanism responsible for handling the congestion in a flow is the congestion window (cWnd). This is a value maintained at the sender that limits how many bytes can be sent in a certain duration. By adjusting the window size according to network conditions, the system can optimize overall throughput.

After a connection is established between two clients, the protocol first enters the Slow Start (SS) phase. In this phase, there is exponential growth of the cWnd such that the flow uses as much of the available bandwidth as possible. This phase continues until a stable threshold is reached, known as the Slow Start Threshold (ssthresh), or until some packet loss is detected by means of receiving multiple DUPACKs. In the former case, the protocol enters the Congestion Avoidance phase (CA). During this phase, the congestion window grows carefully, so that any additional freed up bandwidth can be used while not causing congestion in the flow. In either the SS or CA states packet loss can also be detected by the receiving the aforementioned DUPACKs. In that case the system enters the Fast Recovery (FR) state. In this state, the lost data gets retransmitted and the congestion window is lowered. FR is exited in the case that a new ACK is received acknowledging all lost data or if a timeout (RTO) occurs. In the first case, the system returns to congestion avoidance. In the last case, the sender did not receive any acknowledgment at all within a set timeframe and proceeds to enter the initial slow start phase again. This system is illustrated as a state machine in Figure 3.1.

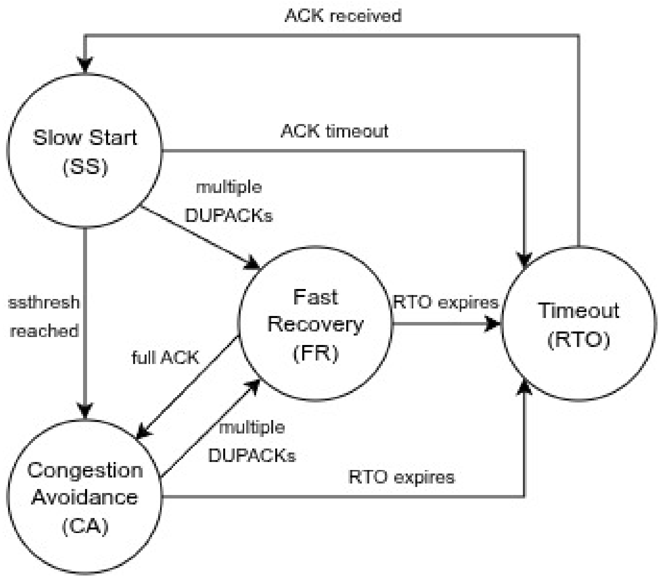

Figure 3.1: TCP state machine

3.1.2 Bandwidth under-utilization in classical TCP algorithms

In older protocols such as TCP-Reno and NewReno the congestion window only grows once per measured RTT in the congestion avoidance state. This is in accordance with the desired goal of this state, where additional freed up bandwidth can be used. However, this also means that window update speed is inversely proportional to RTT. In the paper by Sangtae Ha et al.[6] an example is given. If the bandwidth of a network is 10 Gbps and the average RTT is 100ms with packets of 1250 bytes, it could take 1.4 hours for the congestion window to grow to fully utilize the available bandwidth. This shows that it is quite possible that flows get under-utilized.

To solve this issue, many variants were proposed, one of which is BIC-TCP[7], and is the one that became the default TCP algorithm for Linux in 2004. It uses a binary search to optimize the congestion window. It takes the last window size where a loss occurred (maximum) and the last window size where a loss did not occur (minimum), then it takes the mid-point between these as a new window. After the window grows to this number, it then performs the previous check again. If there is no loss at this new window value, then it can take that point as the new minimum. Performing this repeatedly, the optimal congestion window is found.

3.1.3 TCP CUBIC

CUBIC[6] is the successor to BIC-TCP. In 2006, it replaced it as the default in Linux and has since become the default in MacOS and Windows. A key feature of this version was that it is independent of RTTs. The issue mentioned in the previous paragraph is one that is still somewhat present in BIC-TCP, namely that the timing for updating its window is still dependent on RTT. Although this does not mean that growth is significantly slower, it does have implications for flows competing for the same bottleneck. In a bottleneck, if one flow achieves a higher bandwidth earlier because it updates its window faster than another flow, then it is disproportionately favoured over a flow with a slower RTT. In simple terms, a low latency connection competing with a high latency connection over the same bottleneck will achieve a higher throughput regardless of their connection speeds because it updates its congestion window faster. This is the main issue that CUBIC addresses. By decoupling RTT from the rate of congestion window updates flows do not get favoured based on latency. This is done by making cWnd updates based on wall-clock time instead.

3.2 Classifiers

The process of mapping a certain input x, to an expected discrete output y is called classification. This input can also be a vector $x = (x_1, x_2, \ldots, x_n)$, as long as it still maps to a single y. When an input gets mapped to a continuous variable, it is no longer referred to as classification, but rather as regression.

3.2.1 Logistic Regression

Logistic Regression (LR) is a classification algorithm that tries to separate two classes of data by means of a line. This means that it can only capture linear relations in data. It does so by applying a sigmoid function to a linear combination of input features. It defines what values of the feature x map to a certain output possibility. Then, a threshold is defined by which we say that the event happens or does not happen. In our case, this could be a classification for non-congestive or congestive. This threshold is often configured by default to be 0.5. The boundary is optimized by means of maximum likelihood estimation.

3.2.2 Bagging

A bagging classifier is an ensemble method, which means that it combines multiple decision trees to create a better overall classifier. This works by training each model on a different bootstrap sample of the data in parallel and then combining their predictions to form a single classifier. A bootstrap sample is a resampling of the original dataset with replacement. Doing this creates diversity between the various decision trees, which leads to better classifications when the trees are combined. As an example, a bagging classifier might have a set amount of trees that all make a classification, but the majority of those trees have classified to a certain class. This means that that majority class is then the classification of the classifier as a whole. Because the classifier is an combination of the outputs of multiple models trained on many different bootstrap samples, the classifier will also be less prone to overfitting. In our experimentation, we will use a Random Forest classifier, which is a bagging classification model.

3.2.3 Boosting

Boosting is also an ensemble method. However, this one does not work by training multiple models separately and then combining them, but by progressively adding new models that address the errors of the model previously. In order to make sure a newly added part helps, the model gives more weight to misclassified samples, such that they are more likely to be classified correctly in the new part. For our experiments, we have opted to use the XGBoost boosting classifier.

Chapter 4

Research

4.1 Setup

The goal of this research is to see whether it is possible to use modern classifiers in real-time to accurately classify the losses that can occur when using TCP. Furthermore, we would like to see whether these classifiers implemented into TCP can improve the performance of a simulated network.

4.1.1 Design

We want to see how various machine learning models perform in differentiating between congestive losses and non congestive losses. To do that, we will need a large dataset with relevant metrics from the sender in a data transfer with the appropriate labels. To do so, we created a simple network topology in ns-3 (Figure 4.1). We then collected data at the sender's side and fed this as inputs for the classifiers we tested. Lastly, we evaluated the performance of these classifiers and analysed whether they could be used in real-time to improve the efficiency of a running simulation.

4.2 Software

ns-3 is a network simulator that is used to simulate internet systems. This practice is a form of discrete-event simulation. In essence, it is a library and event scheduler written in C++ that features a large array of functions with many abstractions that enable the developer to create their own network topologies and simulate various network configurations. Additionally, should the user prefer, they can also write their simulations in Python, though not all API calls are supported this way.

To gain an understanding of how the simulator uses abstractions to define simulations, we provide an example. Bob wants to send a file to Alice.

Bob and Alice's computers are defined as a node. A node is a representation of a device. Such a device could be a mobile phone, a server, or perhaps just a computer. Any device that can receive, send, or relay data could be an example. Onto such a node we can install applications, protocol stacks (such as TCP), and peripheral cards. This is just like the things you could do with an actual physical computer, but then abstracted in the simulator. In our example, Bob uses an application to facilitate sending his file. The connection between Alice and Bob is called a channel. This is a representation of the wire or wireless connection between them. To send data from one device to another, you need some kind of network card. In the simulator this is referred to as a network device, which abstracts this exact peripheral card. Each of the things discussed in this example has a corresponding class in the simulator, which can be used to tune and very precisely configure what kind of network topology the developer wants to simulate.

In summary, Alice and Bob's devices are nodes. Bob has an application on his node that he uses to send a file from his network device, over the channel between them, and lastly to Alice's network device, where the data gets picked up by her application.

4.2.1 Motivation

The choice of using ns-3 is due to its accessibility. In the domain of network event simulations, network research and protocol design, it is one of the most used simulators[8]. At this point in time, it is also still actively worked on, and is often cited as a prominent candidate for use in network research. It has a large library of predefined models, and comes with many example implementations to aid in understanding the framework. The developers also maintain very extensive documentation on its various classes and features.

4.3 Viability of machine learning models for this problem

In classical approaches to finding ways to differentiate congestive and non-congestive losses, it is clear that each of these techniques focuses on different metrics that could correlate with the congestive or non-congestive nature of packets. For example, non-congestion packet loss detection (NCPLD) takes in consideration RTT at the sender and compares it to the delay during known non-congestion, or TCP Westwood analyses the ACK stream to determine the necessity of throughput reduction. These are but a few examples that indicate that there are many different attributes that could be conducive to classifying packet loss.

Machine learning models have many traits that help in classification problems, especially when many attributes are present. A strength of many machine learning classifiers is that they are more suitable in finding non-linear relations in data.

4.4 Feature selection

The features we chose to include were RTT and cWnd. Based on our literature research, we found similar experiments to look at the these two metrics[4][5]. The RTT is a measure of how long it takes a packet the server sent to get a corresponding acknowledgment. If a network is more congested, then it might need more time to transfer a packet. This straightforward notion is why it makes a lot of sense to always include this feature. The cWnd governs how much data is being sent into the network, which in turn influences how congested the network is.

Another measure, such as jitter, does not seem to have a direct relationship with the amount of congestion in a network. This is a measure of the difference in latency over time. It can be thought of as a variance of the average latency over a set amount of time. I. Ali et al.[4] in their experimentation on TCP loss causes also found that jitter contributed very little to their classification performance. In their case, they found RTT and cWnd to be the greatest contributors.

4.5 Experiment 1: Classification performance

In ns-3 we have built the topology shown in Figure 4.1. It features two end-hosts named the Server and the Client, with some nodes (hops) in between. The Client is stationary and wirelessly connected to an access point and the server has a direct connection with the access point of the client. The aim is to have packets drop en-route at either the wireless connection between the client and the access point, or at the congested bottleneck link between the two middle hops. This gives us both congestive losses and non-congestive losses that we expect will have different corresponding metrics at the sender's side.

In a simulation run, the speed of the bottleneck link and the other links is randomized. The bottleneck speed varies between 90Mbps and 110Mbps, and the speed of the other links varies between 500Mbps and 1500Mbps. For each simulation, metrics are collected on the sender side in a `.pcap` export, which can be analysed using Wireshark. In addition, the ns-3 trace system is used to extract metrics that are not found in such an export. In a run data is sent continuously to the receiver. Throughout such a run the wireless

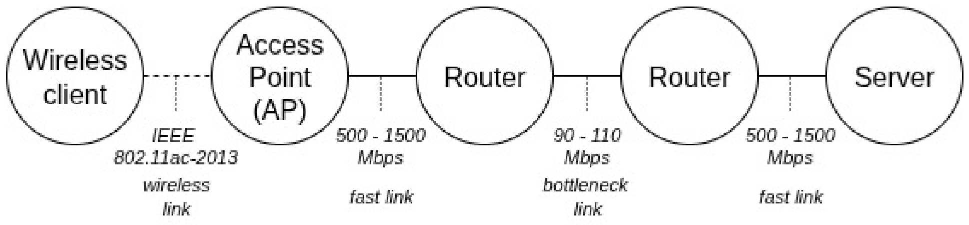
Wireless client
Access Point (AP)
Router
Router
Server
IEEE 802.11ac-2013 wireless link
500 - 1500 Mbps fast link
90 - 110 Mbps bottleneck link
500 - 1500 Mbps fast link

Figure 4.1: Network topology for our simulation

connection of the receiver varies in connection strength, which results in the wireless loss we intend to capture. These include sequence numbers corresponding to congestive losses to label our data points and the progression of the congestion window throughout a simulation.

4.6 Data preparation

We used one large dataset comprised of 300 of the aforementioned simulation runs. The majority of our data comes from our `.pcap` export. This is a simple output file that lists every single packet sent and retrieved from a node of our choosing, in our case the sender.

packet type	sequence number	RTT (ms)
ACK	422233	0.132
ACK	423673	0.147
ACK	425113	0.145
ACK	426553	0.158
ACK	427953	0.173
DUPACK 1	430793	-
DUPACK 2	430793	-
DUPACK 3 (LOSS EVENT)	430793	-
Retransmission	430793	-
ACK	432213	0.203

Table 4.1: Example loss event

This file allows us to compute the RTT for each packet. A loss event is characterized by the presence of multiple duplicate ACKs followed by a retransmission. By collecting the RTT's for the packets before such a loss event, we can construct a part of our data point. This is illustrated in Table 4.1. In this table a subset of the captured packets at the sender's side is shown. At the third DUPACK, a loss event is recognized and by default the congestion window would be reduced. In our experiment we then capture the preceding RTT's from that point, which are $(0.132, 0.147, 0.145, 0.158, 0.173)$. For each of these preceding RTT's, we are also interested in what the congestion window size is at that time. As stated above, we were able to use the ns-3 tracing system to know what the cWnd is at each of these points. By combining these, we have constructed another part of our data point. Lastly, again by means of ns-3's tracing system, we have extracted the sequence numbers from the simulator that correspond with packets dropped due to congestion. This allows us to label all of our data points. The reason this allows to do all of our labelling is because there are only two types of drops. This means that we label drop events that have a corresponding sequence number that the simulator reports as dropped due to congestion as congestive and all other drop events as non-congestive. This leaves us with the following data point:

[5 preceding RTT's, 5 preceding cWnd's, label]

For reference, all simulation and data processing was conducted on a machine with the following specifications:

CPU: AMD Ryzen 5 5600X
OS: CachyOS x86_64 (Arch Linux)

Of note is that ns-3 is deterministic, so no matter the used hardware, if the simulation runs with the same configuration (script, random number generator seed), the output will be the same.

4.7 Experiment 2: TCP CUBIC OSB

To evaluate the performance of the classifier in a more representative setting we have implemented the classifier in ns-3 into the CUBIC protocol. This means that whenever a loss event happens, the classifier takes the recent RTT and congestion window metrics and classifies the loss event as either non-congestive or congestive. In the case that it is ruled as non-congestive, then the congestion window does not get lowered at all. In the other case, the congestion window gets lowered as normal. This gives an updated state machine given in Figure 4.2. Note the additional state to the right of CA and under FR. The addition of the classifier does not necessarily create a new state, and can be considered to be part of the Congestion Avoidance (CA) phase. It is shown as a different state for clarity. The adapted protocol is named "TCP CUBIC OSB", where "OSB" is the abbrevation of the author's hometown, Oost-Souburg.

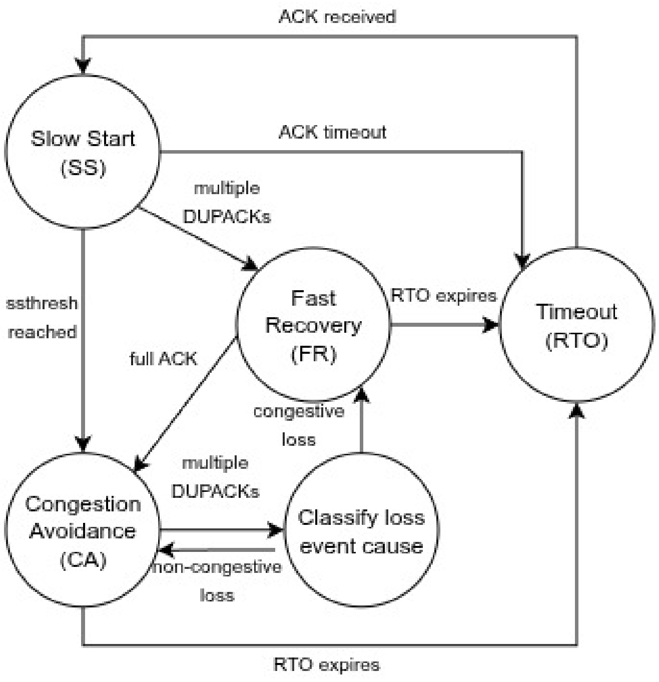

Figure 4.2: TCP amended with classifier to infer loss cause state machine

Similar to experiment 1, the adapted protocol and the original protocol will be ran in 10 sets of 300 simulation runs, where in each set the average throughput, average amount of wireless drops and average amount of congestive drops will be evaluated. Doing many sets of these simulation runs allows us to account for the variation in between runs, as there is a certain amount of randomness present. The throughput gives a simple estimation of whether the protocol with the classifier is better or not. The average amounts of wireless drops and congestive drops can indicate if the addition of the classifier has other effects on the efficacy of the protocol.

Chapter 5

Results

In this section, the performance of our trained classifiers will be discussed. Afterwards, we will evaluate the performance of the best performing classifier implemented in a real running simulation in ns-3, and discuss whether it is an improvement by means of an average throughput comparison.

5.1 Classifier performance

In the following, the performance of our trained classifiers will be presented using our generated dataset. For each of the trained models, a confusion matrix will be presented along with a short discussion of its performance.

5.1.1 Logistic Regression model

In Figure 5.1 we see the confusion matrix of the LR model. Out of all three models, this one has the worst performance by far. Below, in Table 5.1 you can find the classification report. Although it is clear that the congestive case gets classified correctly most of the time, non-congestive, is oftentimes misclassified. This is made clear by the low recall score for the non-congestive class. Furthermore, the AUC score is not much higher than 0.5, meaning the classifier is only slightly better than random guessing.

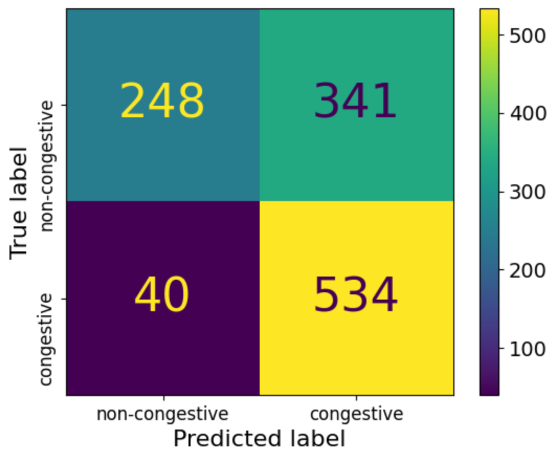

Figure 5.1: Confusion matrix for the Logistic Regression classifier

class	precision	recall	f1-score	instances
non-congestive	0.8611	0.4211	0.5656	589
congestive	0.6103	0.9303	0.7371	574
AUC	0.5987			

Table 5.1: Classification report for the Logistic Regression classifier

It is more than likely that this means that there are non-linear relations in the dataset that logistic regression cannot capture.

5.2 Random Forest

In Figure 5.2 we see the confusion matrix for the LR classifier. From here we see that both classes are being classified more often properly. This is supported by the scores and the AUC score in Table 5.2, where the AUC score is nearly 0.9. Again, the minority class still has worse performance but is significantly better than the performance seen in the previous.

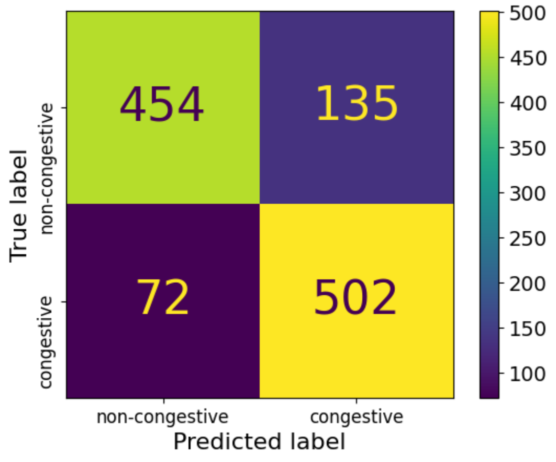

Figure 5.2: Confusion matrix for the Random Forest classifier

class	precision	recall	f1-score	instances
non-congestive	0.8631	0.7708	0.8143	589
congestive	0.7881	0.8746	0.8291	574
AUC	0.8878			

Table 5.2: Classification report for the Random Forest classifier

5.3 XGBoost

Lastly, we have our best performing classifier. The confusion matrix is shown in Figure 5.3. Once again, especially clear from the classification report in Table 5.3, it is a modest improvement over the previous classifier. As this classifier is our best performing classifier, it will be used and implemented in the CUBIC protocol to see whether its addition can be beneficial to the performance of the protocol.

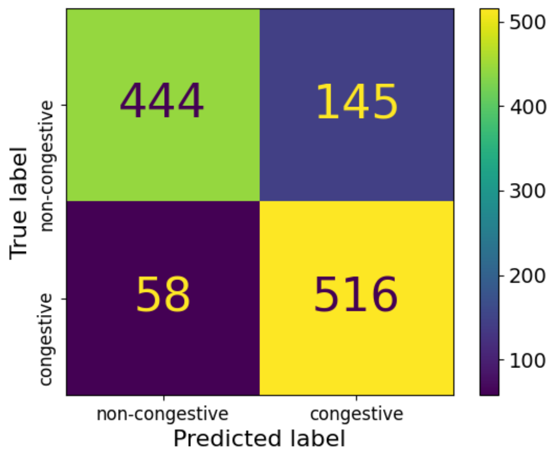

Figure 5.3: Confusion matrix for the XGBoost classifier

class	precision	recall	f1-score	instances
non-congestive	0.8845	0.7538	0.8139	589
congestive	0.7806	0.8990	0.8356	574
AUC	0.8929			

Table 5.3: Classification report for the XGBoost classifier

5.4 TCP CUBIC OSB performance

In this section, the performance of our proposed protocol with the classifier built-in will be evaluated against the original protocol. The three metrics that are used to do so are the average throughput, the average amount of wirelessly dropped packets and the average amount of packets dropped due to congestion. For each, a short discussion will be given. The addition of the drops metrics can indicate whether a change in performance can partly be explained by more or less wasted bandwidth.

5.4.1 Throughput

Given is a box plot showing the average throughputs across various runs compared against the original protocol. Also a table is given, showing statistics such as the average and standard deviation for additional comparison.

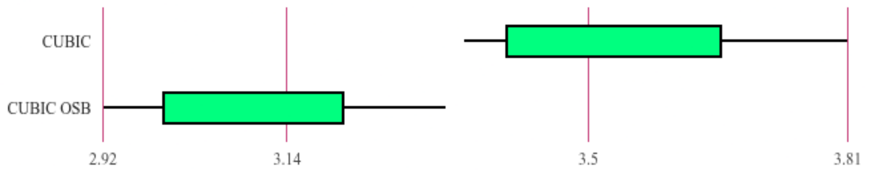

Figure 5.4: Box plot showing the difference in throughput

Statistic	CUBIC	CUBIC OSB
Average	3.54	3.11
std.dev	0.17	0.14
Absolute difference	-0.43	
Relative difference	-12.14%	

Table 5.4: Throughput comparison

It is clear that the proposed protocol performs significantly worse in terms of throughput than the original. The average throughput is 0.43 lower across all runs, and even the best performing run is outperformed by the original's least performing run.

5.4.2 Wireless drops

The comparison laid out in this section will indicate whether the introduction of the classifier also results in a difference in the amount of wireless drops.

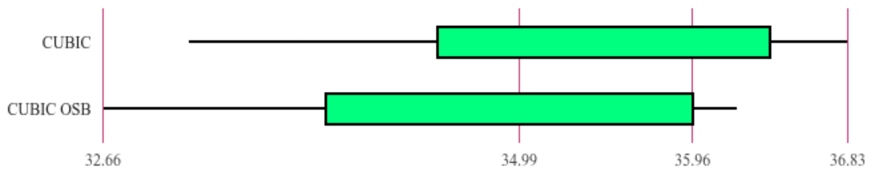

Figure 5.5: Box plot showing the difference in average amount of wireless drops

Statistic	CUBIC	CUBIC OSB
Average	35.39	34.84
std.dev	1.23	1.30
Absolute difference	-0.55	
Relative difference	-1.56%	

Table 5.5: Wireless drops comparison

Although there is a small difference in favour of the original protocol, there does not seem to be a significant difference in the amount of wireless drops. This effect would make sense, as the amount of wireless drops is completely dependent on the wireless network conditions, and should thus not be affected by the introduction of a classifier.

5.4.3 Congestive drops

The following analysis will indicate whether the proposed classifier wastes more or less bandwidth due to congestive drops.

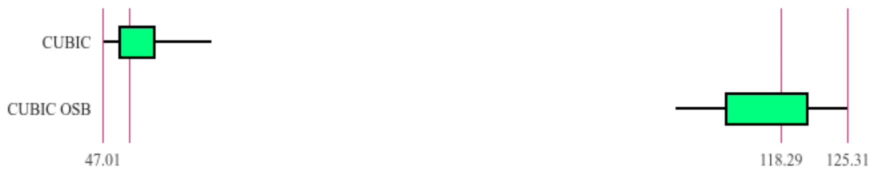

Figure 5.6: Box plot showing the difference in average amount of congestive drops

Statistic	CUBIC	CUBIC OSB
Average	50.80	116.95
std.dev	3.30	6.24
Absolute difference	66.15	
Relative difference	130.22%	

Table 5.6: Congestive drops comparison

Again, we see a large reduction in performance, especially made clear by the gap between the two box plots. In a run, there are about 66 more congestive drops on average, which is more than double the usual amount. It is likely that the classifier classifies some loss events as non-congestive, even though they are congestive. This means that cWnd is not reduced, so the rate of sending stays the same, even though the bottleneck link cannot handle the current rate of sending. This likely results in more dropped packets at the bottleneck link, and culminates in wasted bandwidth. Additionally, because the sender keeps the cWnd higher longer, there might even be more loss events total, which triggers more cWnd reductions. This would reduce the overall throughput even more.

Chapter 6

Discussion

In general, the performance of the random forest classifier and the XGBoost classifier showed promise. With these two usually making the correct classifications. Clearly, the logistic regression classifier had poor performance, and is likely unusable. Further analysis showed, however, that employing the XGBoost classifier in a real simulated system did not improve the overall performance.

6.1 Implications of misclassification

Although the aforementioned classifiers seem usable, it is important to recognize what could happen in case of misclassification. If the system classifies a congestive drop as a non-congestive drop, then it will not lower the congestion window even though it is necessary. This means that it would continue sending data at the same rate even though the network is likely unable to handle it. What then happens is that even more congestion occurs at the bottleneck link, leading to even more wasted bandwidth. This in turn could reduce overall throughput, thereby reducing or even nullifying the benefits of using the classifier at all.

An interesting case is when a non-congestive drop gets classified as a congestive drop. In a normal system, there is no distinction made between the two types of drops, so the cWnd size always reduces in such cases. This means that for this type of misclassification, the system would perform as normal.

6.1.1 TCP-Friendliness

TCP-Friendliness, as explained before, is the trait of a TCP congestion algorithm to not favour one flow over another competing over the same link. Two flows with equal bandwidth competing over the same link should not have vastly different cWnd's, as that would be unfair. Although no explicit friendliness testing was performed, we can still reason about it. Because the congestion response is significantly altered, it is likely that the friendliness will also be reduced. In the case of non-congestive loss events, CUBIC OSB will not reduce its cWnd even though competing protocols over the same flow will. Therefore, it will take a larger share of the available bandwidth, even though it would not necessarily have more bandwidth capacity. This results in our protocol being favoured, and is therefore less TCP-friendly. On the other hand, our testing has also shown that CUBIC OSB is less performant, which could indicate that the cWnd is on average lower. As explained before, this could have occured due to misclassifications, leading to packets being sent at the same rate during congestion, triggering more congestive loss events. This would mean that our protocol would yield too much bandwidth to other flows, therefore becoming overly friendly.

6.2 Improvements

The following are various possible improvements to our experiment. They vary not only from practical things concerning our model training and our simulation design, but also concern ideas on how to create a new simulation that could create a dataset that could create a classifier that can be used in a new version of TCP.

6.2.1 Experiment coverage

In our experiment, we have a single link with 5 nodes, a server sending data to a client wirelessly connected to an access point. To get the data to this client, the data must pass through two hops, one of which is the access point the client uses. For such a classifier to be included in a version of TCP, it should be able to cover most network topologies. Examples of usable topologies could be a user having a local server to store his photos, but also someone sending data from his phone to another person's phone on the other side of the globe. It might be necessary then to drastically increase the topology's size and randomness. For example, you could create a large number of nodes, and have them randomly connected, making sure there is at least one usable path.

Our simulation also made the link speeds static at the start of a simulation run, but these can also be varied over time. However, the connection rate can be dynamic in the real world. Adding this randomness to the simulation could mimic the way speed across certain connections can vary.

In addition, ns-3 has many ways to simulate connection instability. Not only can you reduce the signal strength of certain wireless links, you can have it fade in certain places on a 2-d grid or have the connection obstructed by objects. These are known as propagation loss models[9] and can readily be used. Their usage could aid in creating a classifier that covers more scenarios.

6.2.2 Congestion window size input

The way cWnd was used as input for the classifier could perhaps bring better results if it did not check the size of the cWnd at times of the preceding RTT's, but rather if it were just a vector of the preceding changes overall. That could paint a better picture of whether this value in recent time was decreasing, staying the same or increasing. If this value was steadily increasing leading up to the loss event, it is possible that it became too large. This means that we are overloading the network, leading to a congestion drop. Checking at the times of the set amount of preceding RTT's might lead to a too small window to judge this. Many datapoints in our dataset had no changes in the cWnd leading up to a loss event, which could suggest that the way we captured this metric makes it not very valuable for our classification.

6.2.3 Context size

The number of packets of context surrounding a loss event was set to 5 in our experiments. This means that for every loss event we use metrics about those 5 preceding packets. Naturally, the performance of the classifier is dependent on this context value. Increasing this value could result in adding more noise, so the classifier has inputs that have little to do with the current loss event. However, the amount of context necessary could shift depending on the network conditions. It is unknown whether a relatively fast network could benefit from having more or fewer context.

6.2.4 Hyperparameter tuning

Our analysis suggests that false negatives (congestive loss classified as non-congestive loss) can be a cause for the worse performance of the proposed protocol. Tweaking the threshold at which the classifier classifies to a certain class such that false negatives appear less could possibly improve the performance. This would make the classifier more careful, such that congestive cases do not (or less often) get classified as non-congestive loss events. This insight also means that the recall score is very important for a classifier such as this, as this score is dependent on the amount of false negatives.

6.3 Future work

In this thesis, we have shown that classification of loss causes in simulated networks is possible. As explained in the previous section, there are still many improvements to be made and questions to be answered. As mentioned in the previous paragraph, the context size is a hyperparameter that could greatly determine the classifiability of the loss events. For this research, it was chosen to be 5. Furthermore, another research could try and use CUBIC along with CUBIC amended with a classifier, such as the one trained in this work, to see whether the protocol is still TCP-friendly. In addition, as the poor quality of our proposed protocol can possibly be attributed to false negatives, the threshold of the classifier to evaluate to non-congestive can be made stricter.

A larger question that is unanswered is whether it is possible to train a classifier that is able to distinguish non-congestive and congestive loss events for any type of network. A flow could be simple, such as one within a single house. However, flows can also pass through every continent on the planet, with a huge RTT. Whether a single classification model can tell apart the non-congestion and congestion in such varying conditions is unknown.

6.4 Retrospective analysis

CUBIC was introduced in part to tackle the complexity its predecessor[6], BIC TCP. This protocol, as mentioned before, features a complex strategy to find the optimal cWnd. It had explicit phases with binary search growth, concave growth, convex growth, where each had different updating rules. This results in complicated state transitions, and therefore convoluted state machines. CUBIC had replaced all of this logic with a simple cubic function, which features much of the same phases, such as concave and convex growth, in a single formula. Given that the idea was to reduce complexity, adding trained classifiers to this system can be considered as an idea that goes against the design philosophy of CUBIC in the first place.

Chapter 7

Conclusions

In this thesis, we have explored the usage of classifiers to identify loss causes in simulated networks. We simulated a simple network that represented a wireless client connected to an access point, that received data from a server a few hops away. This produced a large dataset that served as the input for our classifiers. The results show it is possible to identify the loss cause quite effectively. Further analysis showed, however, that the usage of such a classifier within the CUBIC protocol did not yield better performance.

These results serve as a proof of concept that showcase that a classifier can distinguish between a non-congestive and a congestive drop. A future research could look into using such trained classifiers in a real network. Such a research would not only evaluate the effectiveness of the classifiers in a realistic scenario, but could also be a benchmark for how effective research within the simulator translates over to the real world.

Bibliography

[1] Ege Cem Kirci, Ayush Mishra, and Laurent Vanbever. Five blind men and the internet: Towards an understanding of internet traffic, 2025.

[2] Xiongfei Cao, Mingchuan Gong, Lingling Yu, and Bao Dai. Exploring the mechanism of social media addiction: An empirical study from wechat users. *Internet Research*, 30(4):1305–1328, 2020.

[3] Ibtissam El Khayat, Pierre Geurts, and Guy Leduc. Enhancement of tcp over wired/wireless networks with packet loss classifiers inferred by supervised learning. *Wireless Networks*, 16:273–290, 2010.

[4] Inayat Ali, Seungwoo Hong, and Taesik Cheung. Congestion or no congestion: Packet loss identification and prediction using machine learning. In *2024 International Conference on Platform Technology and Service (PlatCon)*, pages 72–76. IEEE, 2024.

[5] Song Cen, Pamela C Cosman, and Geoffrey M Voelker. End-to-end differentiation of congestion and wireless losses. *IEEE/ACM Transactions on networking*, 11(5):703–717, 2003.

[6] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[7] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM 2004*, volume 4, pages 2514–2524. IEEE, 2004.

[8] Jose Gomez, Elie F Kfoury, Jorge Crichigno, and Gautam Srivastava. A survey on network simulators, emulators, and testbeds used for research and education. *Computer Networks*, 237:110054, 2023.

[9] Mirko Stoffers and George Riley. Comparing the ns-3 propagation models. In *2012 IEEE 20th international symposium on modeling, analysis and simulation of computer and telecommunication systems*, pages 61–67. IEEE, 2012.