Network Factors Influencing Packet Loss in Online Games

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Philip Dueck

©Philip Dueck, August 2021. All rights reserved.
Unless otherwise noted, copyright of the material in this thesis belongs to the author.

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Disclaimer

Reference in this thesis to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the University of Saskatchewan. The views and opinions of the author expressed herein do not state or reflect those of the University of Saskatchewan, and shall not be used for advertising or product endorsement purposes.

Requests for permission to copy or to make other uses of materials in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan S7N 5C9 Canada

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

Abstract

In real-time communications it is often vital that data arrive at its destination in a timely fashion. Whether it is the user experience of online games, or the reliability of tele-surgery, a reliable, consistent and predictable communications channel between source and destination is important. However, the Internet as we know it was designed to ensure that data will arrive at the desired destination instead of being designed for predictable, low-latency communication.

Data traveling from point to point on the Internet is comprised of smaller packages known as packets. As these packets traverse the Internet, they encounter routers or similar devices that will often queue the packets before sending them toward their destination. Queued packets introduces a delay that depends greatly on the router configuration and the number of other packets that exist on the network. In times of high demand, packets may be discarded by the router or even lost in transmission. Protocols exist that retransmit lost packets, but these protocols introduce additional overhead and delays - costs that may be prohibitive in some applications.

Being able to predict when packets may be delayed or lost could allow applications to compensate for unreliable data channels. In this thesis I investigate the effects of cross traffic and router configuration on a low bandwidth traffic stream such as that which is common in games. The experiments investigate the effects of cross traffic packet size, bit-rate, inter-packet timing and protocol used. The experiments also investigate router configurations including queue management type and the number of queues. These experiments are compared to real-world data and a mitigation strategy, where n previous packets are bundled with each new packet, is applied to both the simulated data and the real-world captures.

The experiments indicate that most of the parameters explored had an impact on the packet loss. However, the real world data and simulated data differ and would require additional work to attempt to apply the lessons learned to real world applications. The mitigation strategy appeared to work well, allowing 90% of all runs to complete without data loss. However, the mitigation strategy was implemented analytically and the actual implementation and testing has been left for future work.

Acknowledgements

I would like to thank my supervisors Dwight Makaroff and David Callele, for without their incredible patience, support, and guidance this thesis would not be possible.

Contents

List of Tables

List of Figures

List of Abbreviations

ARX	Auto-Regressive Exogenous
BOINC	Berkeley Open Infrastructure for Network Computing
CDF	Cumulative Distribution Function
CSMA	Carrier-sense Multiple Access
CSMA/CA	Carrier-sense Multiple Access with collision Avoidance
CSMA/CD	Carrier-sense Multiple Access with collision Detection
CRAWDAD	Community Resource for Archiving Wireless Data At Dartmouth
DT	DropTail
FCFS	First-Come First-Served
FDM	Frequency Division Multiplexing
FMLP	Feed Forward Multi Layer Perceptron
FPS	First Person Shooter
IoT	Internet of Things
ISP	Internet Service Provider
MLP	Multi Layer Perceptron
MTBF	Mean Time Between Failure
MWU	Mann-Whitney U-test
QoS	Quality of Service
RED	Random Early Detection
RNN	Recurrent Neural Network
RR	Round Robin
RTS	Real Time Strategy
RTT	Round Trip Time
RUDP	Reliable UDP
SFQ	Stochastic Fair Queue
SP	Strict Priority
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UDP	User Datagram Protocol
WFQ	Weighted Fair Queue
WRED	Weighted Random Early Detection

1 Introduction

1.1 Motivation

With the increased demand placed on networking infrastructure, it has become more important than ever for network providers to provide acceptable Quality of Service (QoS) for their subscribers. Network operators often employ traffic shaping algorithms to attempt to ensure that subscribers maintain an acceptable throughput level [41, 47]. Traffic shaping and the stochastic nature of current packet networks cause individual packet delivery times, or latency, to vary widely. Although achieving guaranteed end-to-end data transfer rates is usually the primary concern for Internet Service Providers (ISP), achieving sufficient and consistent end-to-end delivery time is important to some applications such as video games, media streaming and on-line communications.

The variations in delivery time, or jitter, can often be ignored under normal situations. For time sensitive applications such as distributed virtual environments [34, 63], media streaming [24], real-time communications [59], gaming [5, 50] and time synchronization [39] small variations in delivery time have a larger overall affect than latency and can lead to errors or poor user experience.

Network congestion can lead to packet loss that can increase the time it takes for data to be transferred to the destination. In heavily utilized routers packets are dropped under certain conditions and according to configured rules due to the inability to queue all incoming data. The effect of dropped packets depends on the protocol and application but typically will introduce at least a Round Trip Time (RTT) delay from the source to destination. The increased delay contributes to the variation in latency thus contributing to degraded user experience.

Network stability and reliability has a large impact on the user experience of multiplayer games. Armitage identifies First Person Shooters (FPS) and other similar reaction-based games as being primarily affected by poor network conditions [3]. Armitage also suggests that sports and racing games can also be intolerant to high latency, but their tolerance is largely dependent on the game.

For example, increased round trip time between players in a real-time strategy game can alter the perceived positions of players in the game. This difference between game clients can be the difference between hitting your target or being hit. The visual representations of other players in the game may vary slightly between clients depending on how long it takes for new information to be disseminated to all clients. This means that one client may register a shot they take as hitting their target, but would be forced to rewind time and replay the event if the server had different positions for the players. Similarly, you may appear to have dodged an attack from another player but would become teleported back in order to replay the events as seen by the server.

Similar effects can occur with interactive items in games. For example, opening or closing doors or viewing the contents of chests. In these cases the client will typically perform the action preemptively assuming that the current state of the player and the interactive item is correct. However, if confirmation from the server indicates the current information such as player position or item state is incorrect, the game would be forced to rewind the interactions, in some cases teleporting players back behind closed doors or causing items to disappear from inventories.

Massively Multiplayer Online Role Playing Games (MMORPG) such as World Of Warcraft and Black Desert Online have similar problems. One common solution is to eliminate real-time player to player interactions as much as possible and either lock items to a specific player or provide each player with their own copy of the item with which to interact. This allows gameplay to proceed relatively smoothly with a RTT as high as a second [16]. However, some gameplay such as Player vs Player (PvP) combat and instances where multiple players are engaging in Player vs Environment (PvE) can experience similar teleportation or missed attacks as a result of other player's interactions. MMORPGs often mitigate missed attacks by providing players with Areas of Effect (AoE) and rules stating ranged attacks always hit their target. However, the problem with out-of-date information is nearly impossible to eliminate.

Real Time Strategy (RTS) is another popular online game genre. RTS involves controlling many small military and civilian units around a map in order to gain a strategic advantage over an opponent. One might expect that this control over many small units would result in significant network traffic, but in reality the network coordination for RTS games consists of packets on the order of tens of bytes at a frequency of 10 packets per second or fewer. Excessive latency can have an impact on the game; the result typically appears only as a delay in the movement of troops and the construction of buildings. Some RTS games attempt to overcome latency by reducing the number of packets sent and increasing the number of user actions into each packet [56].

Racing games are also a genre of games that may suffer from increased latency due to packet loss [66]. However much work has been done in mitigating the affects of latency in racing games, such as predicting player actions, time dilation, and dead reckoning [35]. However when such attempts fail, the players' perception of the game state, such as car position and race position may differ.

In recent years, multiplayer games have become popular on mobile platforms utilizing 3G, LTE, and now 5G networks. However, the wireless medium can introduce additional instability into the network, increasing packet loss, jitter, and latency [40]. Recent developments in cloud computing have started shifting gaming to the cloud, with mobile devices (and computers) acting as thin clients rendering only a video stream and transmitting periodic control updates. This cloud-based gaming is susceptible to degradation due to packet loss and latency [29].

Disruptions in the timely transmission of data between devices is a significant cause of poor user experience in multiplayer games. While game play time varies greatly between games, players, and genres, it is important to create an immersive and seamless experience for the player. It may not be possible to completely eliminate the problem, but by reducing data loss on the network, reducing latency in data transmission, introducing client side character movement prediction, or implementing similar techniques, it may be possible to provide a positive user experience despite an unstable network. If the Mean Time Between Failures (MTBF) or noticeable disruptions in game play is long enough, a player may be willing to forgive the occasional hiccup.

A better understanding of the influence of network congestion on game packet loss may enable earlier network degradation recognition and in-game logic compensation. Analyzing packet loss over a variety of network conditions would provide further insight into the feasibility of potential general or specific solutions to the problem of packet loss in classes of network conditions. Machine Learning tools could be applied to help predict network instability or packet loss to better aid in the packet loss mitigation One simplistic mitigation strategy would be to retransmit every packet an adaptive number of times, so that the information is received at a latency cost of a multiple of the game update interval (16 ms or 33 ms), which typically is smaller than the round trip time. The RTT can vary between 30 and several hundred milliseconds depending on the network, hardware/software configuration and physical distance from the server [57].

1.1.1 Networks

There are many different network technologies with differing characteristics that pose a challenge to understanding packet loss. Interference from other devices on the network or environmental conditions differ greatly depending on the medium used to transmit signals. Different protocols are used to combat interference depending on the transmission medium and standards used. Traffic shaping and policing take several forms and can be configured according to corporate policy and therefore can differ greatly from one ISP to another.

Cellular phone technologies are prone to wireless interference. To combat the interference, several techniques are implemented on transmitters. One such technique, called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), detects if the channel is too noisy to transmit or is currently in use by another transmitter and will wait increasingly longer random amounts of time if unable to transmit [61]. A related technique, called Carrier Sense Multiple Access with Collision Detection (CSMA/CD), detects when the data being transmitted collides with another transmission or has been obscured by noise; once detected the transmitter will again wait a random amount of time, increasing with each failed attempt, before attempting to transmit again [61]. The response to collisions or noise on the network can introduce significant delay on busy or noisy links. Since the amount of delay before retransmission is randomly chosen, albeit with known constraints, the overall delay increase is random and may be impossible to predict. CSMA/CD and CSMA/CA will be discussed in greater detail in later sections.

Often, mobile networks will also divide data into smaller pieces that are then encoded with error detection or error correction techniques [21]. Error correction allows the receiver to detect, and in many instances correct, transmission errors; this added layer of complexity adds computational time on both the transmitter and receiver as well as increasing the total amount of data transmitted. While this time would generally be constant for encoding, in the event of errors additional processing may be required to correct the data, thus increasing the transmission time. In the event that the receiver is unable to correct the data, many mobile protocols implement a variant of a method called Auto Retransmit reQuest (ARQ) [12] to request that data be retransmitted. This retransmission can also cause increased delays that are difficult to model.

Wireless technologies such as WiFi, Bluetooth, and LTE suffer from interference between devices that operate in the shared wireless spectrum allocated for communication. These technologies implement similar protections against noisy links. Bluetooth and older WiFi networks operate on widely used unlicensed frequencies that are even more prone to interference then cellular networks thus increasing the random variation in delay. In prior work, Bluetooth is shown to have fewer random delays than WiFi but more than cellular networks [22].

Physical links, like those comprising wired networks and the backbone of the Internet, do not suffer as much from electrical interference or traffic from other nodes. Unlike WiFi and other wireless technologies, physical links typically are shielded from external noise and are typically point-to-point networks, meaning the only data transmitted is between two devices eliminating the possibility of data collision. Despite these protections, physical networks still implement the same interference and traffic detection mechanisms. Cable networks and some older Ethernet networks do have multiple clients on the same physical link so they will occasionally collide and create additional delays thus increasing jitter.

Due to the complicated nature of wireless networks and the relatively low interference seen by wired networks, this thesis will take the first step by using a wired network to simplify the simulation and analysis. Future work would include expanding the simulation into larger more complicated networks including WiFi and LTE.

1.1.2 Routers

Routers are used to direct traffic between networks but sometimes routers are required to fulfill other roles such as traffic policing and traffic shaping. Traffic policing is the enforcement of rate limits determined by ISP subscription tiers or corporate policy. Policing typically uses a token bucket or similar mechanism [31]. Token buckets allow short bursts of bandwidth, over the limit, after periods of inactivity. At a sustained transfer rate that meets or exceeds the configured rate, packets will begin to be dropped depending on the size of the incoming buffer. Traffic shaping and QoS attempt to prevent any one data stream from consuming all the available bandwidth. Traffic shaping is typically accomplished by assigning data streams to packet queues based on classification rules defined by the network administrator. Packets are then drawn out of queues according to an algorithm known as a packet scheduler [61].

For this research, the generated application traffic is assumed to be from a game with a relatively low data rate requirement. It is further assumed that the internet connection to the ISP is not saturated and therefore does not encounter traffic policing by the ISP. These assumptions remove one source of jitter from the system, thus simplifying the analysis. While these assumptions should hold true under most circumstances, it is possible that some games may transmit more data or the ISP link may be saturated with other traffic during game play.

1.1.3 Evaluation Methodology

Due to the complexity of controlling real hardware, a simulation environment was developed using *NS-3* [53] and *The Click Modular Router* [32]. The *NS-3* network simulation software was selected because of its popularity and large networking component library. *Click*, a software package that allows developers to create custom router configurations, was selected because *NS-3* does not include a configurable router but does include integration with Click. With these packages, an environment was parameterized in which attributes such as Link capacity, traffic rates, and router queueing algorithms to allow for simulation of a vast number of network configurations and conditions.

1.2 Thesis Statement

Network games require real-time communications to maintain synchronization between clients [22]. Other internet applications such as collaboration software [19], live media streaming [10], and teleoperation [51] have also benefited from real-time communication technology. With the increased use of mobile devices as a platform for these types of applications, the transmission characteristics of packets on wireless networks such as WiFi and LTE have become an important topic. In many cases, game state updates are small packets sent to clients at a fixed interval; for example, a 128 byte packet every 33ms or 17ms for frame rates of 30 and 60 frames per second respectively [13].

This thesis determines what factors in competing network traffic and bottleneck routers have the most influence on low bandwidth game-traffic. Data is collected via a simulation environment, and an analysis technique based on run-length encoding is developed and used to determine the impact of each of the parameters chosen to study which informs a mitigation technique. The work is then compared with real-world data to test validity.

1.3 Outline

The rest of this thesis is laid out as follows. Chapter 2 covers the background and related work for modeling and predicting loss. Chapter 3 describes the experimental configuration and experiments. Chapter 4 presents results and analyses. Chapter 5 contains conclusions and future work.

2 Background and Related Work

This chapter provides the background information and related work that demonstrates the need for additional research. First, there is a brief discussion about the IP protocol used on the Internet focusing on packet loss sources, most of which are outside application control. Next follows a discussion of several studies investigating and identifying traffic shaping due to network policy. After that existing mitigation techniques for packet loss and latency are discussed. Finishing off the chapter is a look at previous work relating to packet loss prediction.

2.1 Data Transmission Overview

Packet networks, such as those supported by the IP network layer protocol, are complicated systems with many discrete components. At the top layer of the Internet protocol architecture, the application communicates using an Application Protocol Interface (API) to send blocks of data to the operating system directly or to an intermediate library which then forwards the data to the operating system. The operating system uses its network stack implementation to generate packets from the data provided by the application. These packets are sent on the network through modems, routers, and switches until the packet finally reaches its destination. At the destination, the packet is retrieved by the operating system's network stack and the payload is then forwarded to the application that processes the data. There are many complicating factors within the Internet and subsystems that cause differing amounts of delay depending on environmental conditions.

The exchange of data between applications starts in the application; for games, this is generally during the processing of game state and before the rendering of the actual frame. Scheduling when the application sends the data depends on many factors including thread scheduling, and for games, the rendering frame rate. The packet is not actually transmitted on the network when the application creates the packet; instead, the application queues the data, along with meta-data such as the destination, with the operating system through the provided API and continues processing. Once the operating system has received the data, the data is wrapped in either User Datagram protocol (UDP) or Transmission Control Protocol (TCP) headers inside Internet Protocol (IP) and Ethernet headers and then queued for transmission via the physical hardware.

In many point-to-point networks such as data centers, ADSL links, and wired home networks where computers are plugged directly into a router or switch, the transmission protocol does not need to be aware of other traffic on the network. In these cases, each machine has a dedicated connection to a switch or router that coordinates the forwarding of packets to the destination.

However, in unswitched networks where more than two devices share the physical medium (or more than one device transmit and receive on the same medium), such as most cable networks and wireless technologies, the Network Interface Card (NIC) must wait until the physical medium is free to transmit. One mechanism to implement this access protocol is known as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). If the physical medium is currently being used by another host on the network, determined by first listening for existing traffic, the network card will wait a random number of time steps where the random number will be between 0 and $2^i - 1$, where i is the number of times the network has been sensed as busy [27]. If, during a transmission, the NIC senses that another host is also transmitting, or there has been a burst of interference, it will cease transmitting and perform the same exponential back-off as when waiting for the network to be clear; this is known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD). However, in modern wired networks the need for CSMA/CD is greatly reduced due to isolation of a link by restricting access to only two devices, such as the computer and the switch [61].

In some networks, for example wireless networks and power line networks, additional processing must be done to ensure reliable reception of data. Due to the limited frequencies available, multiple devices may be attempting to communicate on the same channel, prompting the need for protocols other than CSMA/CA and CSMA/CD [25]. Additionally, to ensure reliable delivery of data, packets are often broken up into smaller blocks and encoded with error correction codes and error detection codes to attempt to improve channel reliability and quality. In many cases, if a packet or segment has been detected as damaged then the physical layer protocol utilizes an Automatic Repeat reQuest (ARQ) mechanism to request that a block be retransmitted.

In order to share a frequency band between multiple clients, several multiplexing techniques can be utilized. Multiplexing takes two forms: Time Division Multiplexing (TDM) and Frequency Division Multiplexing (FDM). TDM partitions transmission time into discrete time units and clients are assigned one or more of the time units to transmit data. Frequency Division Multiplexing involves dividing a larger frequency band into multiple smaller frequency bands that are then assigned to clients.

Duplexing is similar to multiplexing but only creates two divisions. Duplexing is often employed to govern when or how NICs can transmit and receive. For example, Frequency Division Duplexing divides the allocated frequency band in two, one band for transmitting and the other for receiving. The goal of Multiplexing and Duplexing is to provide the clients with a pseudo-guaranteed share of total bandwidth; this guarantee greatly depends on the number of clients and the types of traffic on the network. This division means that on some FDM networks with fewer clients than divisions, some of the available bandwidth will be left unused. However, barring interference, clients will enjoy a relatively constant throughput. On the other hand, on networks with many clients, the interference between clients will cause significant degradation in the QoS.

As a packet travels from source to destination, the packet may encounter one or more devices that temporarily stores the packet and then retransmits the packet. These devices can be *switches*, which typically redirect packets to another device on the same network, or *routers* which act as bridges between two different networks. It is in the router that most of the QoS and traffic shaping occur. In a typical router, packets may be classified according to one or more metrics such as strings embedded in the payload, port, destination, protocol, etc. [4] and placed into a buffer that corresponds to the classification. The router then chooses the next packet to send, using a scheduler algorithm configured specifically for the current application. It is through the classification/scheduling mechanism that network administrators control the quality of service for customers or application types.

Common classification and scheduling algorithms include Weighted Fair Queue (WFQ), Strict Priority (SP), and Stochastic Fair Queue (SFQ). WFQ and SFQ are variations of the Fair Queue (FQ) which typically classifies packet flows based on source/destination pairs with a Round Robin (RR) scheduler where a packet is drawn from each queue in turn [44]. WFQ adds a weight to each queue and the scheduler algorithm draws a number of packets from each queue proportional to the weight assigned to the queue [48]. This weight allows the network administrator to prioritize traffic flows and helps guarantee throughput for flows that may otherwise have packets dropped. SFQ also uses the RR scheduler, but modifies the a classifier to use a hashing algorithm with random seed that is perturbed periodically to ensure that any two packet flows are not continuously queued together [42]. SP, however, takes a different approach, emptying queues with the highest priority before moving on to lower priority queues. This allows the administrator to guarantee throughput for some classifications of traffic, but may ultimately starve low priority traffic. Variations of these algorithms and similar algorithms are sometimes used as well, giving network administrators control over how traffic is forwarded.

Queues, however, can only hold a finite number or packets, or bytes, depending on the configuration. In the simplest case, referred to as DropTail, when a queue is full and a packet arrives, the packet is dropped. Under high load DropTail does not perform well, maintaining a full buffer and preventing congestion control from properly adjusting. To combat the issues with DropTail, other queue management algorithms, such as Random Early Detection (RED) and Controlled Delay (CoDel) were developed to start dropping packets before the queue is full to trigger congestion control earlier and keep the buffer available for bursty traffic.

Once a packet has reached its destination, the data is extracted from the packet and the application is notified of the arriving data. Similar to what occurs when sending data, the application may be unable to process the data as soon as it arrives. The receiver must wait until the appropriate thread is scheduled or the application is in a particular state before processing the incoming data.

2.2 Transport Layer Protocols

There are two main transport layer protocols in use on the Internet today, Transmission Control Protocol (TCP), and User Datagram Protocol (UDP) [61]. TCP has features to ensure reliable transfer and congestion control built into its protocol and is often used for applications such as web browsing, file transfers and media streaming. The reliable nature ensures that all data gets transferred and the congestion control ensures that all streams have an appropriate share of the network. Reliability of the network comes at a cost of both latency and variation in latency due to the retransmission of lost data or throttling of the transmission rate. On the other hand, UDP is a minimalistic protocol that has neither congestion control nor reliable delivery. The simple nature of UDP allows for quick transfer of data from source to destination but suffers from packet loss over congested networks. The reduced latency has made UDP the first choice for real-time applications such as video games and communications.

While UDP may seem like an attractive choice for applications where latency is most important, there are issues that can affect the end-user experience of an application that uses UDP as its transfer protocol. Some applications can tolerate the occasional loss of data, but often applications implement their own reliability mechanisms such as UDP Data Transfer (UDT) [20] or Reliable UDP (RUDP) [6, 62] that can cause increased jitter in the event of packet loss.

2.3 Measurement Studies

BADABING [58], developed by Sommers *et al.*, was created to estimate packet loss along a route on the Internet. Like similar active measurement tools, *BADABING* worked by sending small probe packets to a destination server and calculating the resulting loss statistics. Instead of sending probe packets in a Poisson distribution like standard packet loss measurement tools, *BADABING* sent a sequence of packets at fixed intervals with a variable probability. The application could then look at packet loss in each sequence to determine if the network was starting to drop packets, continuing to drop packets, or finished dropping packets. They were then able to estimate the packet loss for other packet streams on the network. They also showed that using Poisson distribution for probe packets was ineffective for measuring loss episode frequency and loss episode duration. They demonstrated that their method not only produced more accurate results, but with a smaller number of probes than previous estimators. This thesis explores the possibility of embedding a similar mechanism into a game using its own update mechanism for estimation of packet loss. The game could then use this information to compensate for expected packet loss.

Carullo *et al.* [8] described a simulation using *NS-3* to study the performance of WebRTC over LTE. WebRTC is a technology that allows web applications to communicate in a peer-to-peer manner and uses UDP to traverse NAT firewalls. In their experiment, they built the simulation environment using physical machines for LTE clients and the web server, enabling them to generate real traffic from a real application. Their experiment consisted of transferring video from one client to another and then analyzing the packet traces. The *NS-3* simulator was configured with four scenarios representing the ideal case, a more realistic configuration with fading and channel error, an increasingly more realistic scenario with fading, errors, and finite queues, and finally a configuration with fading errors, finite queues, and interfering cross-traffic. The simulator was also configured to simulate client movement. Their results indicated that with increasingly realistic configurations the throughput decreased and jitter and packet loss increased. The introduction of cross-traffic created an increase of 1 ms (34%) in the jitter median and a 19 kb/s (13%) decrease in the throughput median. The packet loss median also increased with slightly under 1% more packets lost, an approximate 14% increase over the similar scenario without cross-traffic. Similarly this thesis explores the effect of the network configuration on an application's data stream. However, instead of analyzing WebRTC over LTE with varying degrees of realism, this work focuses on game-traffic through a simple router, varying factors such as cross-traffic and router configuration.

2.4 QoS and Traffic Shaping Modelling

Weinsberg *et al.* introduced a framework called *Packsen* [65] that was able to detect when a network path was being shaped and the approximate parameters used to shape the traffic, assuming that a WFQ or SP scheduler was being used. The *Packsen* framework consists of a measurement server that sends the data to the end client and an experiment server that contains scripts and parameters required to coordinate and operate an experiment.

To detect if an ISP is actively shaping traffic, two short data flows are simultaneously sent to the client and the distributions of packet inter-arrival times are compared using the Mann-Whitney U-test (MWU) [37]. The MWU test is a nonparametric test to determine if one of two distributions is stochastically greater than the other. If the packet inter-arrival time distributions differ, the ISP is assumed to be shaping the traffic. Once shaping has been detected, *Packsen* then attempts to calculate the weights or priorities given to each flow to create this behavior. Assuming that the two packet flows were sent at just below link capacity, the actual bandwidth would be expected to be determined by the queue weights.

Cross-traffic is particularly troublesome when attempting to detect traffic shaping and determining weights. To combat cross-traffic, *Packsen* [65] repeats the experiment multiple times until the variance in the results drops below an arbitrary threshold. While *Packsen* may work well as a standalone test to determine if a link or path is being shaped, the methodology used by *Packsen* (the experiment and measurement server interactions) make real-time detection of traffic shaping and prediction of future behavior in a mobile application difficult. The methodology *Packsen* uses for comparing inter-arrival times could be explored as an alternative to analyzing packet loss. However, the effectiveness of inter-packet arrival times may be reduced in low loss environments.

DiffProbe was introduced by Kanuparthy and Dovrolis [30] to allow researchers to test if their ISP is treating different traffic flows differently by either dropping packets or increasing the latency of packets. *DiffProbe* assumed that the ISP is classifying IP packet flows into high and low priority streams. It also assumed that the ISP is using SP or WFQ scheduling and either Weighted Random Early Detection (WRED) or DropTail (DT) queue management. If the ISP does not perform differentiation, it is assumed that the ISP uses a First-Come First-Served (FCFS) queue with DT management. *DiffProbe* simultaneously generates two packet flows, an application stream that is intended to be classified as low priority and a probe stream that is intended to be classified as high priority.

During the test, the probe stream was increased to just below the customer's link rate limit. Differentiation was determined by comparing the relative end-to-end delay statistics and packet loss statistics of the two packet streams using Kullback-Leibler divergence [33]. Once it has been detected that the probe and application streams have been treated differently, the queue type is determined by examining the bursts of probe packets received, immediately after application packets, for variability in the end-to-end delay. Packet bursts that exhibit large variability are assumed to be WFQ, while low variability is assumed to be due to SP queues. The type of queue management is determined by analyzing the packet loss rates of the application and probe streams.

ShaperPrope [31], an extension of *DiffProbe*, was developed with a slightly different approach than *Packset*. Instead of using the inter-packet time to determine if a packet stream was being shaped, *ShaperPrope* used the throughput curve over a set period of time to determine if there was any throttling being done. The method assumed that traffic shaping was done through a token bucket of a fixed size and fixed token generation rate. Token buckets work by incrementing a token counter by some rate up to a set value (the bucket size), and then as packets are forwarded, the token counter is decremented. If the token counter is at zero, the remaining packets are queued until a token is available. The size of the bucket and the token generation rate correspond to the burst rate and the throttled traffic rate respectively.

Assuming data was being sent as fast as possible in the presence of a traffic shaper, the recipient would observe high throughput until the token bucket became empty and the packet stream was throttled. Using the measured throttled rate and the duration and rate of the un-throttled traffic, an estimate of the bucket size could be made. The authors note that cross-traffic could have an impact on the results by artificially reducing the received data rate and suggested removing outliers from the rate calculations. The direct application of this method of detecting traffic shaping to data generated by applications that create low to moderate traffic would not work due to the requirement for *ShaperPrope* to saturate the network. While *ShaperPrope* could be used to detect current shaping parameters on application start, the knowledge of shaping parameters may not be sufficient to create an accurate model of the network.

Zhang *et al.* [68] developed a methodology for identifying links treating traffic differently on the path between two points. To determine if a link was being throttled, they systematically probed multiple paths using different types of traffic. For each type of traffic, they calculated packet loss statistics for each path between the source and the destination. Using a complete network graph, they use boolean algebra as described by Nguyen and Thiran [45] to identify individual links that are congested. If different traffic types have different performance on a link, one could conclude that the given link is treating the traffic differently.

Molavi Kakhki *et al.* [43] investigated traffic differentiation on mobile networks. They employed a VPN to record packet flows that they then used as a baseline for determining whether traffic was throttled. They replayed the recorded traffic over the mobile network while maintaining the inter-packet spacing as closely as possible. They compared the recorded traffic with the replayed traffic to determine if the traffic was throttled for the given path. Several different types of streams, such as HTTP, YouTube and P2P, were tested to see if different traffic was handled differentially. To develop the system, they employed commercially available hardware to perform real world traffic shaping. Additional experiments indicated that stream classification was based on destination port, keywords embedded in the first data packet, or a combination of the two. In their experiments using several American ISP they found that YouTube, Netflix, and Spotify were most impacted by the ISP traffic shaping policies. In some cases the traffic was simply throttled, in other cases proxies or middle boxes manipulated the data providing cached copies of data or changing the network connection parameters. With the increase in encrypted traffic, the ISP-controlled proxies and middle boxes become less likely to have an impact on network traffic due to their inability to manipulate the contents of a stream undetected.

2.5 Mitigation

An alternative mechanism for concealing jitter and latency from a user is through in-game mitigation techniques where the game hides the lag with in-game elements. For example, animation sequences that allow for the game to synchronize when interacting with items. However, hiding lag and jitter in-game may not work for all games.

One example of this technique is demonstrated by Gao *et al.* [17] where they replaced network players with bots when game updates were late. In their paper, they developed a game of pong that incorporated the bot replacement technique. In their implementation, the game would ignore input from a networked player if the latency became too high and substituted a bot in their place. When network conditions improved, the user's inputs were once again used to control the character. While the game was receiving inputs from the player, the game would train the bot using regression analysis so it would behave like the player. The authors had previously built a dead-reckoning algorithm to handle network lag, but they found the predictive algorithm produced significantly lower error. With this approach packets are intentionally discarded once the bot takes over. It is possible that the bot would not be adequately trained by the time it is required to take over the role as player. It would be beneficial if the incoming packets were used to train the bot further, even if they arrived late. With the technique described in Section 4.3, more information would be received in each packet; even if the data was too late to act on, the additional information could be used to further enhance the bot's predictive abilities.

Although intended for industrial environments with remotely operated robots, Hou *et al.* developed a framework that incorporated movement prediction and packet loss probabilities that could achieve a zero latency experience [23]. The authors used a Kalman Filter [28] to predict a position in the future and a k-repetition packet scheme (k packets are repeated) to achieve reliability. They then created an optimization problem to determine how far into the future to predict and the minimum amount of bandwidth required to achieve the desired latency. Most of their calculations were done considering wireless networks such as 5G, and as such they also considered other factors such as increased error correction encoding and multiple antennas. Despite this, many of the principles involved could be applied to most on-line games. Additionally, if the network state were constantly monitored, for example through feedback from the recipient, the optimization problem could be recalculated on the fly adjusting the k value as needed. With an additional feedback mechanism this type of framework could easily be incorporated into the work provided in the next chapters.

2.6 Game Streaming

Game streaming services now are becoming popular to gaming enthusiasts. With game streaming services the user uses a thin client to render the output (visuals and audio) and register inputs from the user (button presses, finger swipes, etc). In this scenario, small on-demand packets would be sent from the user to the server and the server would send back each frame as it is rendered. This increase in the reliance on the network also increases the potential for network conditions to play a part.

Some mitigation techniques have been proposed such as *Outatime* [36]. With *Outatime*, the game server renders multiple potential scenes that could result from user input. The thin client would then choose which scene to display depending on the input detected and potentially blend scenes together to achieve the desired outcome. The server employed worker instances that would execute predicted inputs and return back the rendered frames including 3-D positioning information and additional viewpoints. The 3-D information and additional images allow the client to render scenes even if the outcome was not accurately predicted. The downside to this approach is that not only is the server subject to packet loss from the client, but the client may also lose packets coming from the server. The synchronous packet loss means that the client may not receive any predicted data from the server in a timely fashion. Additionally, the increased data load from the server means more packets that potentially could be lost. Additional techniques such as the one described in Section 4.3 may help mitigate packet loss in both directions.

2.7 Packet Loss Prediction and Machine Learning

Immich *et al.* built a novel approach to combating packet loss for streaming videos [26]. In their system they assign each frame an importance value that when combined with the predicted packet loss adjusts the amount of Forward Error Correction (FEC) that is applied to each packet. Packet loss prediction was done through statistics of reports of packet loss sent by the recipient back to the server. The recipient would keep track of the periods of time where packets were received without errors and the periods that packets were lost or had errors and periodically report to the server. The server would then calculate a probable packet loss percentage for the next batch of packets to be transmitted using statistics sent by the recipient. This probability was then combined with the importance level of the frame using Ant Colony Optimization (ACO) where simulated ants randomly navigate the search space. The importance level of a frame was determined using several factors including Motion Intensity, Frame type (I or P frame) and frame size. The Motion Intensity was determined using a Recurrent Neural Network that was trained off-line with labeled data. They found that their system was able to increase the QoE of the video while reducing network bandwidth requirements as compared to other methods. Their approach has a similar method of analyzing the packet loss where statistics of the periods of received and lost packets are calculated to determine the state of the network. With a similar feedback mechanism, it would be possible to have the server dynamically adjust the redundancy of data being sent to the client. In some instances this could be error correction as is done here, but in other cases where future states are unknown it may simply be redundant data.

With the goal of improving the performance of scientific data transfers, Giannakou *et al.* developed a Machine Learning (ML) framework to predict packet loss percentages in TCP streams [18]. Their approach involved training a Random Forest Regression (RFR) [7] using existing datasets including metrics such as average RTT, Source, Destination, initial TCP congestion window, throughput, duration, and file size. To improve performance of the ML algorithm they applied a smoothing algorithm to reduce short term noise in order to expose the long-term trends. Their experiments indicated that the methodology worked reasonably well with an approximate 97% accuracy rate on the training datasets, but much worse ($> 66\%$) when trained and tested on unrelated data. They concluded that the relative importance of each feature used varies over time, however they indicate their algorithm performed well for datasets with similar statistics.

Verma *et al.* presented an approach to hide packet loss in real-time music streams [64]. In their paper they propose a solution where a MLP network was trained on similar audio as the music that was to be transmitted and the recipient device would use that MLP network and the past two seconds of received audio to construct the data for missing packets. They compared their work to similar AR solutions and found that their approach resulted in fewer errors for almost all configurations of the AR algorithm. However, they did note that their algorithm was unable to run in real time on their test CPU, so current implementations would need to rely on other hardware to achieve real-time prediction of lost data. Although their work was focused on predicting missing audio, depending on the application a similar method could be used to fill in the gaps for missing game state or user input.

Cheng *et al.* attempt to improve the Forward Error Correction (FEC) on media streams using Long Short-Term Memory (LSTM) ML network to determine the number of Reed-Solomon (RS) encoded packets to send [11]. In their solution they divided the data sent into blocks and used feedback from the recipient to determine the number of lost packets per block. The LSTM was trained on existing datasets, with a RTT-sized gap between the historical blocks and predicted blocks to account for the time it takes for feedback from the client. This does indicate that in order to train the LSTM the RTT of the route must be known in advance. In both of their experiments with simulated data and then existing captured data, they found that the LSTM accurately predicted the packet loss 70% of the time. This accuracy allows the algorithm to automatically adjust the redundant packet data sent, improving performance while ensuring that most of the data is received in a timely fashion. This paper describes an algorithm that could be easily applied to similar packet loss mitigation techniques such as the one presented here.

Another approach to minimize packet loss by directing traffic more efficiently was explored by Mao *et al.*[38]. In their paper they implemented a mesh network in the lab and used traffic at the edges to train a Tensor based Deep Learning model to properly route traffic on the edge nodes. They found that the new Tensor based approach nearly eliminated packet loss in their experimental environment. However, their findings require changes to network infrastructure and therefore could not be applied directly to mobile games. Additionally, their experiments don't necessarily represent the real world and they themselves indicate that more work needs to be done to consider other factors such as packet size in the routing algorithm.

Roy and Ghosh compared two separate ML approaches (Decision Tree (DT) and Logistics Regression (LR)) to predicting the class of packet loss (high or low) for use in edge routers to determine whether a packet stream was accepted [54]. From a pre-analysis of their sampled datasets, they found that download speed, download throughput, upload throughput, ISP, and technology (ADSL, Satellite, Fiber, etc) were the highest predictors of packet loss. Using 10-fold cross validation with 80% of their data to train the two ML models and the remaining 20% to test against they found that the DT algorithm had 89% accuracy outperforming the LR algorithm with only 86% accuracy. If enough information was available during the startup of a mobile game, it may be possible to utilize a similar method to determine the class of packet loss that would be seen and compensate and/or inform the user they may experience degraded performance. Under most circumstances the amount of information available to a game client at startup is limited, but a similar idea could be used throughout the game play to inform the mitigation strategy to either increase or decrease the number of packets to resend. Investigation into whether a ML algorithm could be used to improve the performance of the mitigation strategy has been left for future work.

2.8 Chapter Summary

The Internet is not well-suited for real-time communications due to the variability in the packet trip times and packet loss. Most Internet components were built with the idea of reliable data transfer from source to destination. This reliability, when used on a congested network, can cause jitter in the packet trip times. If some of the reliability is removed, for instance replacing the TCP protocol with UDP, the destination may not receive all the packets. Packet loss and jitter can create a poor user experience if applications are not designed to handle poor network conditions or require real-time synchronization.

Most research into compensating for packet loss and jitter has been from media streaming and similar situations where future information is already known, allowing for the use of FEC. In these instances machine learning has been used with some success in predicting the future state of the network, allowing for adjustments to the FEC algorithm. For real-time applications, the two areas predominately explored are end-to-end trip time prediction and traffic differentiation detection. Trip time prediction using machine learning techniques such as MLP, ARX and Auto-Regression achieved some success in related work and may warrant additional research. Research into identifying when packet streams are being manipulated by network policy have been shown to work well in identifying when throughput is being reduced by network policy, but has done very little in attempting to compensate for or bypass such restrictions. Some research has also been done in client side or server side prediction of either user inputs or server outputs in order to mask delays in the network traffic.

3 Experiment Configuration and Design

This chapter contains descriptions of the simulation software, simulation parameters, and experiments. The simulation software is described first, followed by the simulation environment and the parameters used for the experiments; then the experiments and their designs are described in detail. Section 3.4 focuses on describing the run length analysis that is the basis for all of the analysis in Chapter 4. Finally, Section 3.5 describes a proposed mitigation strategy that is then explored later in Chapter 4.

3.1 Simulation Environment

All simulation is done with *NS-3* [53], a discrete event-based network simulator often used for research and protocol development, and the *Click Modular Router* [32]. Discrete event simulators model activity as events that are then sequentially processed by the system rather than modeling the system as a continuous simulator that constantly updates element state. An event-based simulator can jump from one discrete event to another, potentially enabling a simulation to execute faster than real time. *NS-3* has many built-in modules such as CSMA links, routers, and computer nodes. All elements in *NS-3* are written in C++ and the framework allows for the development of new elements that can be easily integrated into simulations.

NS-3 comes with built-in integration for the Click Modular Router allowing sophisticated routing to be performed inside of an *NS-3* simulation [60]. The *Click Modular Router* is a software router designed to ease router development and other network devices through a custom scripting language. The *Click Modular Router* has many built-in components such as queues, classifiers, schedulers, and packet redirection. *Click Modular Router* elements are written in C++ and the framework provides a simple mechanism for defining custom elements for use in router design.

To distribute the simulation to multiple computers, the *Berkeley Open Infrastructure for Network Computing* (*BOINC*) [2] was used. *BOINC* allows any number of computers to be used as compute nodes, all coordinated by a central server. Each compute node, by default, is configured to process tasks only when idle, making *BOINC* a perfect candidate to utilize unused computers in the campus computer labs. Although *BOINC* has mechanisms for issuing parameters and for collecting and analyzing results, the size of the results for these experiments made it easier to upload the results to a large storage facility for later analysis.

3.2 Experimental Parameters and Performance Measurements

All simulations follow the same pattern (Figure 3.1). Two computers, one generating simulated *game-traffic* and another computer generating simulated *cross-traffic* at rates up to 200 Mbps, are both connected by a common link to a router. This router acts as a gateway that routes the traffic to a second network where two computers are simulated, one that receives the game-traffic and another that receives the cross-traffic. The logical flow of data is denoted with a dotted line, the solid lines indicate a physical connection between devices and actual data flow occurs within these physical connections.

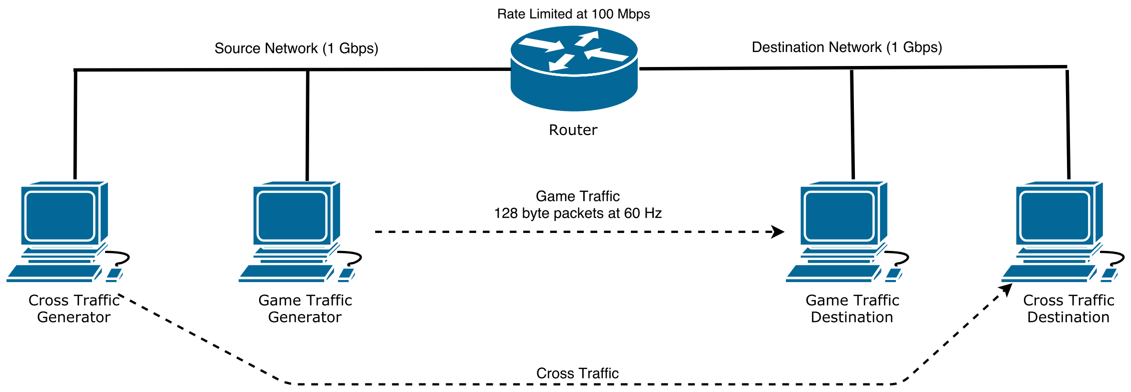


Figure 3.1: Diagram of the simulated network.

Both the source and destination networks are modeled with gigabit links to minimize collision probability. The router includes a token bucket rate limiter, limiting the outgoing traffic to 100 Mbps to introduce a bottleneck in the traffic flow; the queue length was fixed at 200 packets. The 200 packet buffer size is larger than the minimum recommended using the *small-buffer model*, but small enough to quickly drop packets under load instead of increasing the latency of the packets [55].

3.2.1 Experiment Configuration

Each experiment consists of a node sending simulated game-traffic consisting of 128-byte packets, at 60 packets per second, for 120 seconds, while a second node simultaneously sends packets at a specified rate, inter-transmission time and packet size. Packet Capture (pcap) files are generated from the traffic stream as packets reach the router, as they leave the router, and as they reach their destination, thereby facilitating stream characteristic comparison and analysis. Game packets contain a sequence number so that, using only the packets captured on the receiving node, we could easily identify and quantify dropped packets. Unlike the simulation, real world traffic may deliver UDP packets out of order, or too late to be used; these packets would also be considered dropped. The captured game-traffic at the receiver is then run-length encoded as a series of counts of (received, dropped) packets, to further facilitate the analysis. For example, a run of 20 successful packets followed by 2 lost packs is encoded as (20, 2). The run length encoding is further described in Section 3.4.

3.2.2 Initial Experimental Factors

The experimental factors are the cross-traffic characteristics and the router policies. The remainder of this section describes the chosen configurations. Table 3.1 outlines the cross-traffic parameters and selected values.

Table 3.1: Cross-Traffic Parameters

Parameter	Values
Cross-Traffic Streams	TCP, UDP
Mean Bitrate (Mbps)	25, 50, 75, 100, 125, 150, 175, 200
Mean Packet Size (Bytes)	128, 256, 439, 512, 1024, 1500
Packet Size Distribution	Constant, Uniform, Gaussian, Realistic
Packet Size Standard Deviation (Bytes)	1, 2, 4, 8, 16, 32, 64, 128
Packet Size Range (Bytes)	1, 2, 4, 8, 16, 32, 64, 128
Inter-Packet Time Distribution	Constant, Uniform, Gaussian, Realistic
Inter-Packet Time Range (percent)	10, 20, 40

The *packet size distribution* settings include a) constant, b) uniform random, c) Gaussian, and d) realistic (drawn from trace data) distributions. The uniform packet size distribution created randomly-sized packets, uniformly distributed between $PacketSizeMean \pm PacketSizeRange$. The Gaussian distribution generated a random packet size with a specified mean and standard deviation. Realistic packet sizes were determined using the pdx/vwave dataset [49] downloaded from CRAWDAD [67], consisting of traces gathered from sniffing packets on public hot-spots around Portland, OR in July, 2009. The traces were combined to create an empirical, histogram-based packet size distribution. As the resulting distribution had a mean packet size of 439 bytes, 439 bytes was added to the *Mean Packet Size* parameter to allow for direct comparison with the other distributions.

The *inter-packet transmission time* settings are the same as those for packet size. The constant inter-packet time setting calculates an inter-packet time from the mean bitrate and packet size. For uniform distributions, the *inter-packet time range* parameter is a percentage of the calculated inter-packet time. The Gaussian inter-packet distribution used a standard deviation of 10% of the calculated inter-packet time. The 10% value was chosen to keep the percentage within ±40%, similar to the uniform distribution. The realistic inter-packet spacing was generated using a modified version of the synthetic traffic generating code created by Ammar *et al.* [1]. Modifications were made to accommodate variable-sized packets.

Table 3.2: Router Configuration Parameters

Parameter	Values
Type of Queue	DropTail, RED
Number of Queues	1, 2, 3
Type Of Classifier	Shared, Dedicated, Random

Table 3.2 provides the router configuration factor values. The *Type Of Classifier* and *Number of Queues* determined the queue into which the cross-traffic and game-traffic packets were placed. In all cases, all game-traffic was placed in the first queue as all packets in a data stream would typically be treated in the same manner. For the shared classifier, all traffic was placed in the first queue. The dedicated classifier evenly split the cross-traffic between the remaining queues, leaving the first queue dedicated to game-traffic. The Random classifier used a uniform distribution to assign the cross-traffic to one of the available queues.

The *Type Of Queue* (or router queue management policy) parameter is either Random Early Detection (RED) or DropTail. RED [15] is an algorithm that was designed to increase fairness between flows by introducing randomness to determine which packets are queued. The RED algorithm drops packets according to configured probability that is roughly proportional to the queue size once the number of packets in the queue reaches a specified threshold. DropTail, unlike RED, does not drop packets with a configured probability but instead drops packets that cannot be queued when the queue reaches capacity. The queues were configured using the defaults, or recommended settings, from the Click documentation. RED queues were configured to begin randomly dropping packets once there were more than 5 packets in the queue, with a linear increase in the packet drop probability up to 2% when there were 75 or more packets in the queue. These numbers were based on Click documentation examples and only slightly modified to prevent the degenerative case (where the queue becomes full and behaves much like a DropTail queue).

Due to the large number of parameters, a full-factorial experiment would have resulted in a prohibitive number of configurations. Fortunately, many combinations were illogical and were removed (for example, experiments that would result in zero or negative payload sizes). The resulting number of configurations to evaluate was over 86,000.

3.2.3 Extended Experimental Factors

In order to explore additional parameters without significantly increasing the number of runs, a reduced parameter-set was created by eliminating some of the values from the initial experimental parameters (Section 3.2.2) and adding some additional values to explore.

Table 3.3 contains the parameters and values used for the second set of experiments. Of the Packet Size Distributions, only the realistic and constant values remain, chosen because the artificial distributions

Table 3.3: Extended Cross-Traffic Parameters

Parameter	Values
Cross-Traffic Streams	TCP, UDP, 2 TCP, 3 TCP, 1 UDP 1 TCP, 1 UDP 2 TCP, 1 UDP 3 TCP
Mean Bitrate (Mbps)	75, 100, 125, 150, 175, 200
Mean Packet Size (Bytes)	439, 512
Packet Size Distribution	Constant, Realistic
Inter-Packet Time Distribution	Constant, Realistic

performed similarly and to provide contrast between artificial and empirical configurations. Most of the Mean Packet Size values were eliminated, leaving only the 439-byte and 512-byte configurations, chosen to remain because the 439-byte configurations were required for the Realistic Packet Size Distribution and 512 was the closest of the remaining values. The Inter-Packet Time Distribution was also reduced to the realistic and constant configurations, again to provide contrast between the random and non-random distributions. The 25 Mbps and 50 Mbps configurations were removed from the Mean Bitrate because they were deemed unlikely to have any significant impact on the results being so far below the bottleneck link capacity.

The largest addition to the Extended Experimental Factors is the inclusion of multiple Cross-Traffic Streams. While in the previous experiment only a single TCP or a single UDP stream comprised the entire cross-traffic, the new configuration parameters included additional configurations that included multiple simultaneous cross-traffic streams at a time. For the second set of experiments, up to 3 TCP streams could be present in the cross-traffic with or without a single UDP stream. The decision to only include a single UDP stream was because it is believed that a single UDP stream behaves the same as N UDP streams provided the net bitrate of the streams is the same, while TCP has additional overhead for congestion management and flow rate control that makes multiple TCP streams more interesting to observe.

Table 3.4: Extended Router Configuration Parameters

Parameter	Values
Type of Queue	DropTail, RED, CoDel, AdaptiveRED
Number of Queues	1
Type Of Classifier	Shared

For the queue related parameters, most of the previous configurations were removed, leaving only Number of Queues: 1 and therefore only one valid Type Of Classifier: shared. These parameters were primarily kept to reduce the complexity in the router during this experiment. In this experiment, two additional values were added to Type of Queue; CoDel and AdaptiveRED. CoDel [46] is a queue management algorithm designed to reduce the amount of time a packet is in a queue. The CoDel algorithm measures the minimum amount of time a packet has been in the queue over a configured sliding window and drops packets from the beginning of the queue if the minimum delay exceeds the target delay for at least the sliding window duration. AdaptiveRED [14] extends RED by automatically adjusting the probabilities to minimize fluctuations in the queue length. The AdaptiveRED queues were configured with a target queue length of 100 packets and a maximum drop probability of 5%. The parameters for AdaptiveRED were chosen with the desire to keep the queue half-full on average with a probability that prevented the queue from degenerating into the DropTail state. CoDel was configured with the defaults of a 5 ms target latency through the router over a 100 ms sliding window of packet forwarding activity.

3.3 Real World Data Collection

In addition to simulating various network conditions, data was collected over the Internet using a simulated game. For these experiments, a small server application was configured on DigitalOcean[1] to accept UDP packets and record the success/loss characteristics. Several devices were then configured similarly to the game packet generator in the simulation used in the first two experiments, with a client application that transmitted 7200 128-byte packets at 60 packets per second to the server. The configuration details for the devices are in Table 3.5. The LTE device primarily captured data in a residential area and in a business park, but also included data from the university campus and the commute between locations. WiFi and wired data was collected using a business level ISP as well as a residential Cable and ADSL provider. At the university a single wired device was used to generate data.

Table 3.5: Real World Device Configurations

Device	A	B	C	D	E	F
OS	WIN 10	WIN 10	CentOS 6	WIN 10	WIN 10	Android
Network Type	WiFi	Wired	Wired	Wired	WiFi	LTE
ISP	Business Park	University	Business Park	ADSL/Cable	ADSL/Cable	Telco

3.4 Run-Length Analysis

User gameplay experience is greatly affected by the length of time the game can be played without experiencing interruptions due to network lag. Network lag can be caused by network conditions delaying packets,

[1]`https://digitalocean.com`

or by the loss of packets causing data retransmission. Network lag interruptions can pull the gamer out of the immersive experience [9, 50, 52]; however, lag may not always be an issue. For example, if the network can transmit data without lag for a minimum period of two minutes, than any game that could be played in under two minutes is less likely to experience lag, therefore likely increasing the user experience.

This leads to the assessment of whether or not a game is playable. For the purposes of this thesis, *playable* has been defined as being able to play for a sufficient amount of time to complete a level or run of a game without data loss. How long the game needs to run without data loss, and the impact of data loss on the user experience, depends greatly on the game. For example, twitch games that involve fast reactions like FPS games would be most impacted by late or lost data, potentially causing glitches in the game like missed shots and teleporting players. However, FPS games tend to run until an objective or time limit is reached so the run times can be longer than the two minutes explored here. The racing game genre operates mostly with a deterministic physics engine allowing games to remain in sync, however late or lost inputs could cause clients to desynchronize. Fortunately, racing games tend to have short levels, reducing the length of time (or duration) for which data loss needs to be avoided. Board games or RTS games can be more resilient to delays in data transmission. The game requirements would be used to inform the bounds of the run length required for a playable game.

Therefore, if we can determine on average how long a game can play uninterrupted and how long these interruptions last, we may be able to derive mitigation techniques that will provide the user with the experience associated with longer periods of gameplay. With longer gameplay, the possibilities for a successful network gaming experience are increased. Additionally, if we can identify the conditions that lead to network instability, game developers could implement mitigation techniques that pre-emptively alter networking parameters or gameplay to reduce the effect of packet loss.

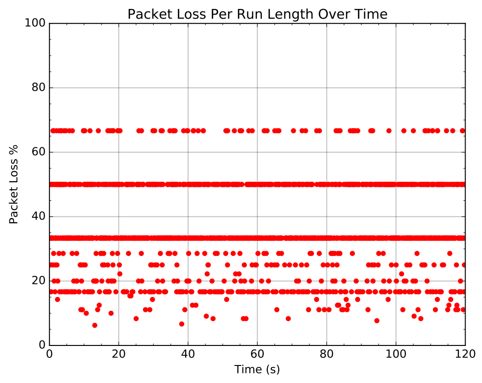
To characterize the relationship between lost and received packets and to aid in understanding when a network is reliable and unreliable, a run-length encoding of received/lost packets was employed to observe the stability of packet reception. Put another way, network communications can be described as a sequence of (Received, Lost) tuples (which we refer to as success/loss intervals, or just intervals). *Received* is the number of sequential packets sent that were received by the game client that met the gameplay requirements and *Lost* is the number of sequential packets sent that failed to be received by the game client or failed to meet the gameplay requirements such as being received out of order or too late to be processed in a timely fashion.

These individual tuples are elements of a sequence of tuples that represent a single simulation or data capture period (also referred to as a run). New intervals are created with the first packet received that meets the gameplay requirements. For example, ten successive packets that met the gameplay requirements followed by two packets that did not meet the gameplay requirements would be represented as the tuple (10, 2). It follows that an arbitrary period or run could then be described as a sequence of these tuples; for example, ((10,2), (100, 12), (2, 2)). An example taken from one of the experiments is ((919, 1), (380, 1), (3299, 1), (2599, 0)). One thing to note, however, is that any run losing packets at the beginning of the run will have the first interval encoded as (0, x) where x is the number of packets lost; similarly any run ending with successfully received packets will have the last interval encoded as (x, 0) where x is the number of packets received.

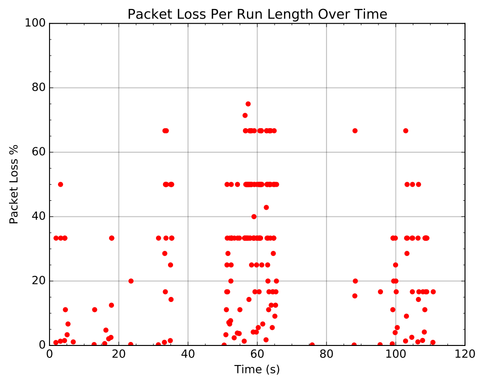
This technique is a form of run-length encoding. This encoding enables us to quickly calculate loss statistics and identify if packets are lost in long sequences or in short sequences while maintaining the temporal relationship of the packets in the data received.

Figure 3.2a is an example run of the simulation. The router and cross-traffic parameters for this run are the following:

- UDP Cross-traffic,
- Two DropTail Queues,
- Random Classifier,
- 512-byte Packet Size,
- Uniform distribution with two bytes standard deviation,
- Uniform inter-packet time distribution with +/- 20% spread, and
- 125 Mbps bitrate.



(a) Frequent and Consistent Packet loss



(b) Bursty Packet Loss

Figure 3.2: Example Runs Demonstrating Run-Length Encoding

Given the run-length data for a run, we can plot the start of each interval as a percentage loss; for example the interval (10, 2) would be 2 / (10 + 2) = 16.67% packet loss. With this visualization, we can see the percentage of packets lost over time; however, the magnitude of each interval must be determined by the spacing between each plotted point. Ideally, we would see only a few points (all of which are at or near zero) representing long sequences of received packets followed by only a few lost packets. This plot can be used to determine the volatility of the network over time.

In the run in Figure 3.2a, there appear to be heavy concentrations (almost solid bands) at 50% and 33% and lower concentrations (slightly less intense bands) at 66% and 16%. The percentages are a result of integer math, resulting in bands at common fractions such as 1/2, 1/3, 2/3, 1/4, etc. In this chart, most of the time the number of packets lost is less than or equal to the number of packets received. There are some exceptions though, the most noticeable is the line at 66%, that indicates occasionally twice as many packets are lost than received. The figure shows how the ratio changes over time; the magnitude of each tuple however, is difficult to determine. Even though the horizontal space between dots on this chart represents the time between each interval (and, therefore, the total number of packets represented by each dot), the number of plotted intervals tends to obscure this information. This is a weakness of this visualization technique that is difficult to overcome without very (physically) large visualizations or very low packet transmission rates.

We can overcome this weakness, in part, by counting the number of each unique (received/lost) value tuples. As shown in Table 3.6, a clearer picture of the relationship between successfully received packets and the subsequently lost packets emerges. Table 3.6 allows us to see the frequency of the success/loss ratio while also showing the maximum number of packets lost in an interval. These two data representations, used together, assist in the interpretation of the number of packets lost and the relationship between packet loss and time.

Table 3.6: Success/Loss Interval counts (frequent and consistent)

	Packets Received															
Lost		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	324	954	57	44	163	9	8	22	2	2	6	1	0	1	1
	2	88	252	0	8	44	0	5	8	0	0	2	0	0	0	0

Table 3.7: Percent Loss Interval Counts (frequent and consistent)

Percent	6.67	7.69	8.33	9.09	10	11.11	12.5	14.29	15.38	16.67	20	22.22	25	28.57	33.33	50	66.67
Count	1	1	6	2	2	22	8	9	2	163	52	5	57	44	962	576	88

Using the data in Table 3.6, no more than two packets are ever consecutively lost during this example configuration. Despite the poor network conditions and high packet loss, the network appears to be stable with very little variation in the number of packets lost over time. However, in both representations it is not easy to determine the distribution of the packet loss. Table 3.7 contains the same data, grouped by percent loss, giving a better indication of the overall distribution of percent loss. Figure 3.2b is another example run of the simulation. The router and cross-traffic parameters for this run are the following:

- UDP Cross-traffic,

- Two DropTail Queues,
- Random Classifier,
- 512-byte Packet Size,
- Realistic packet-size distribution,
- Realistic inter-packet time distribution, and
- 75 Mbps bitrate.

The counts of each success/loss pair for this configuration can be seen in Table 3.8. In this configuration, fewer packets were lost than in the previous example. However, the network appears to be relatively unstable, having periods of little to no packet loss, punctuated by network instability, with up to five consecutive packets lost. The bursts in the packet loss indicate that the network likely experienced several short periods of congestion.

Table 3.8: Success/Loss Interval counts (infrequent and bursty)

		Packets Received																							
		1	2	3	4	5	6	7	8	10	11	12	13	14	17	20	23	24	26	29	39	41	47	50	56
Lost	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	38	59	8	5	17	3	3	7	2	0	1	1	3	3	1	3	1	1	2	1	1	1	0	1
	2	22	20	1	2	4	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	3	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

		Packets Received																			
		63	65	74	75	77	89	90	101	103	109	143	173	204	327	361	428	473	550	616	726
Lost	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	1	1	2	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
	2	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.9 shows the percentage loss breakdown for the second example run. In this table, the frequent single packet losses produce numerous low percentage counts. Like Table 3.7, in Table 3.9 most of the intervals have 33%, 50%, or 66% packet loss. However, Table 3.9 has two instances where the packet loss was greater than 70%.

Table 3.9: Percent Loss Interval Counts (infrequent and bursty)

Percent	0.0	0.11	0.14	0.16	0.21	0.23	0.28	0.3	0.49	0.57	0.91	0.96	0.98	1.1	1.11	1.32	1.33	1.38	1.52	1.56	1.75	2.08	2.38	2.5	2.53	3.33	3.7
Count	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1

Percent	3.85	4.0	4.17	4.76	5.56	6.67	7.14	7.69	9.09	11.11	12.5	14.29	15.38	16.67	20.0	25.0	28.57	33.33	40.0	42.86	50.0	66.67	71.43	75.0
Count	1	1	3	1	3	3	1	1	2	7	3	3	1	17	7	8	4	61	1	1	58	23	1	1

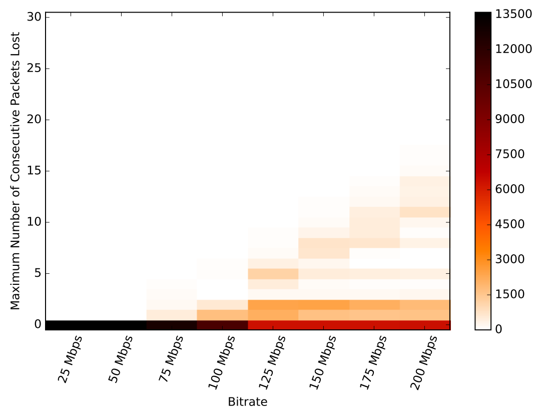
The previous charts and tables are useful for visualizing individual runs, but statistical techniques were used to compare runs. Table 3.10 contains the aggregated received/lost measurements generated for the configurations used in Figure 3.2a and Figure 3.2b. The six columns on the left are the values for *received packets* including Standard Deviation, Mode, Median, Mean, Minimum and Maximum packets received in a row. The five columns in the middle are the values for *lost packets*, including the Standard Deviation, Mode, Median, Mean, and Maximum number of packets dropped per run-length. The right side of the table contains the statistics for the *percentage of packets* lost for each run-length in a configuration.

Table 3.10: Example Run-Length Statistics

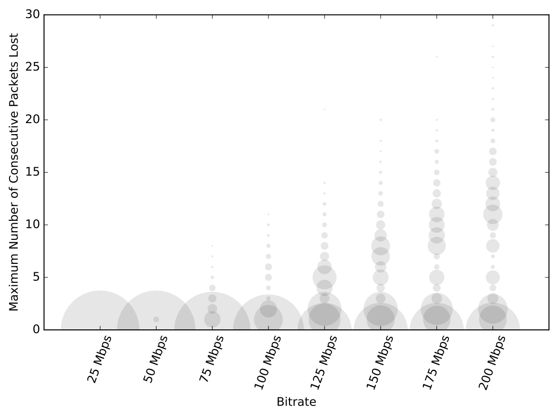
	Received Packets						Lost Packets					Percent Lost Packets in an Interval				
	stdev	Mode	Median	Mean	Min	Max	stdev	Mode	Median	Mean	Max	stdev	Mode	Median	Mean	Max
frequent and consistent	1.6	2	2	2.4	1	15	0.40	1	1	1.2	2	13	33	33	37	66
infrequent and bursty	105	2	2	29	1	905	0.54	1	1	1.3	5	20	33	33	32	75

Comparing the two configurations, we see that the greatest differences between the two configurations are the *Standard Deviation of consecutively received packets* and the *Maximum number of consecutive packets* received and lost. The Standard Deviation could be used as a measure of the stability of the network and could be used by applications to adapt to, or alert upon, changing network conditions. The Maximum number of packets lost provides us with the value of the number of packets, or updates, for which an application would be required to compensate under the given network conditions. The *mean packets received* can be thought of as the *Mean Time Between Failure* (MTBF), representing the expected time the application can run before experiencing a packet loss. The right side of the table gives an indication of the ratio between successful and lost packets per interval. Similar to the lost packets metrics, the Standard Deviation and the Maximum values provide us with the most insight; the Standard Deviation could be used to indicate the stability of the network and the Maximum value indicates the maximum period of time the game could run uninterrupted. Throughout the following sections and chapters, the *maximum number of packets lost* (*Maximum Packets Lost*), the *mean number of received packets* (*Mean Success Length*), and *maximum percent lost packets* (*Maximum Percent Loss*) will be used for comparing different configurations to help identify network conditions that are unusable and those that may be able to be overcome with mitigation techniques.

While the previous charts and tables are useful for analyzing the data for individual configurations, they are unwieldy for determining the effect of small changes in the cross-traffic when comparing hundreds of configurations at a time. Using the statistics calculated for each configuration and then grouping the configurations by a configuration parameter enables analysis of the impact of changing that parameter. Figure 3.3a is an example of taking the maximum number of packets lost for each configuration and generating a heat map. Each cell corresponds to a single integer value of a parameter's measurement. The shading of each cell corresponds to the number of configurations that resulted in a output value that fit into that cell. The darker the cell, the more configurations it contains. Each column contains the same number of configurations, allowing direct comparison between the distributions.



(a) Heatmap



(b) Bubble Chart

Figure 3.3: Maximum consecutive packets lost vs. bitrate

Due to the loss of granularity, and the difficulty of identifying subtle shading differences, a bubble chart can also be used to display the same information. Figure 3.3b contains the same information as Figure 3.3a, but uses different-sized bubbles (centered on the number of packets lost) where the area of the bubble is proportional to the number of configurations that had that value. The visualization shows us that the 75 Mbps set of configurations resulted in losses that are not noticeable in the heat map. The bubbles in the charts are translucent so that the overlapping bubbles can be seen, and darker overlapped areas indicate the close proximity of two or more data points (but has no other meaning in the analysis). Unfortunately, the overlapping bubbles can make it difficult to see individual points.

3.5 Proposed Mitigation Technique

In online games, packet loss can lead to degraded performance or produce unwanted visual or game play artifacts. While an increase in latency is also detrimental to the game-play experience, minor increases in latency can be tolerated by most games. With this in mind, one possible mitigation technique to reduce latency due to packet loss that could be deployed involves combining a number of previous packet payloads with every new packet transmitted. Assuming that the application can tolerate a short increase in delivery latency, the application would have all the information it needs to survive a small number of consecutive lost packets.

A simple “what if we retransmitted the previous x packets” scenario was deployed. Since rerunning the experiments with the mitigation technique would be prohibitive, an analytic approach was taken. Giving the existing run-length data, any packet loss run-length that preceded at least one successfully received packet was reduced by the mitigation amount and the successful region increased by the same number. For example, if the single packet mitigation analysis was applied to a run where one packet was received, two packets were lost, and one packet was received, the results would be one packet received, one packet lost, and two packets received. Successful run-lengths without packet loss run-lengths between them were then merged. However, this analysis does not take into account the impact of increasing the game packet size, which may indicate a possible threat to validity and has been left to future work.

3.6 Summary

Real world analysis of the effects of cross-traffic and network configuration on game-traffic is difficult due to limited access to the internal workings of the Internet. To better understand the interaction between the cross-traffic, router configuration and game-traffic, a simulation environment was created that enables each component to be individually configured, thereby allowing examination of their complex interactions.

For each experiment, one set of parameters was selected from a pool of available configurations and then simulated with varying quantities of cross-traffic (an empty network, an underutilized network, a saturated network and an over saturated network) to characterize the effect the simulation configuration has on the game-traffic under different cross-traffic rates.

4 Analysis and Results

This chapter reviews the results of the experiments and comments on the effectiveness of the proposed mitigation strategy. Section 4.1 looks at the impact, on the game-traffic, of manipulating individual simulation parameters. The experiments quantified the conjecture that packet loss is difficult to predict; however, the length of successive losses is less than four packets in our domain of interest. An example mitigation technique for these communications challenges is presented in Section 4.3 wherein recent packets are retransmitted along with the current packet. The proposed mitigation technique is then analytically applied to the captured data to evaluate its effectiveness. Section 4.4 investigates real-world data, compares the real data to the simulated data, and evaluates the effectiveness of the proposed mitigation technique within the real-world scenarios. Finally, in Section 4.5, the results of the previous sections are discussed.

4.1 Single Parameter Analysis

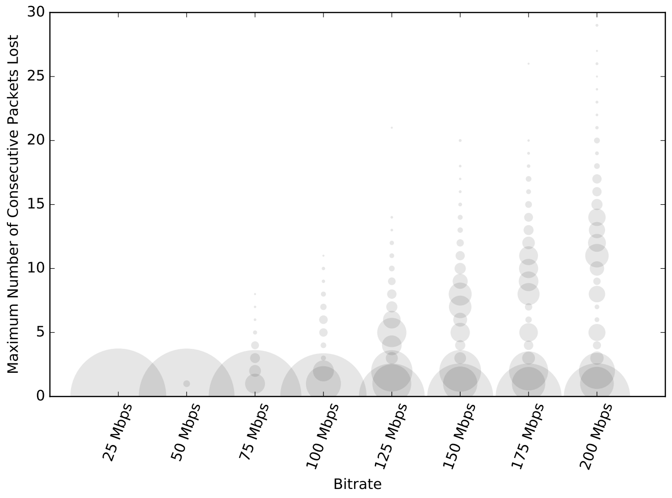
In this section, each parameter is analyzed to determine the effect that the parameter values have on the overall packet loss characterization. For each parameter, the packets lost and packets received are analyzed in an attempt to determine if there are any patterns that can be exploited by a mitigation technique.

4.1.1 Cross-Traffic Data Rates

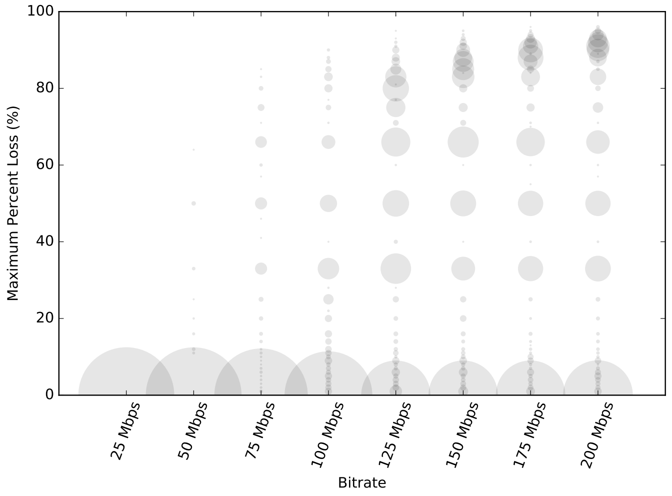
In this section the data is grouped by the configured cross-traffic bitrate to determine what impact, if any, the cross-traffic bitrate had on the game-traffic packet loss. Packet loss starts early, around 50 Mbps, suggesting additional factors contribute to packet loss, and increases as the bitrate increases. The details of the analysis and results can be found in the next section.

Results

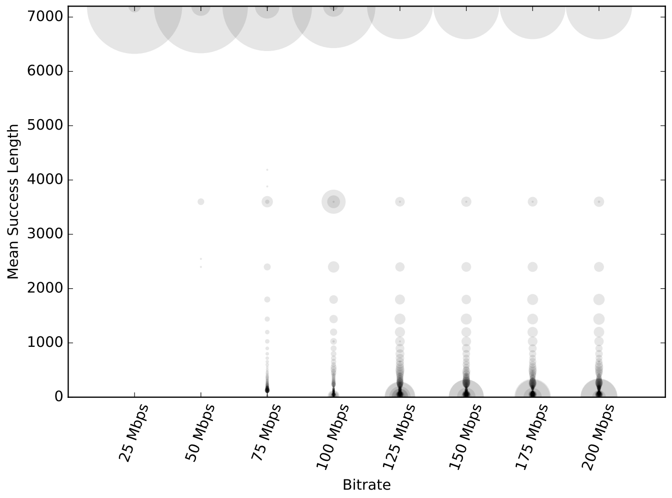
The bubble chart in Figure 4.1a shows maximum consecutive packet loss from the run-length encoding of each configuration. Each configuration used to generate Figure 4.1a is run with the same pseudo-random number seed to ensure that the pseudo-random numbers generated have a minimal impact on the results, allowing direct comparison between runs. The first observation from Figure 4.1a is that very few game packets were being dropped when the cross-traffic was configured at 25 Mbps or 50 Mbps. This is expected, as the outbound link is well below the saturation point. For cross-traffic bitrates over 100 Mbps, the effect on the game-traffic was similar in each case. The router dropped a significant number of packets without a



(a) Maximum consecutive packets lost



(b) Maximum percentage of packets lost



(c) Mean consecutive packets received

Figure 4.1: Bitrate Distribution

discernible pattern. A small number of experiments with a bitrate of 75 Mbps cross-traffic, experienced game packet drops. For cross-traffic bitrates under 75 Mbps, there is little or no packet loss; however, the packet loss increases as the bitrate increases.

Overall, across all configuration groups, 89% of the runs had fewer than five consecutive packets lost. For configuration groups with high bitrates, the number of runs with fewer than five consecutive packets lost drops to as low as 26%. This could present a problem for mitigation techniques that proactively retransmit small number of packets every frame time, such as those discussed in Section 3.5.

Comparing Figure 4.1a to Figure 4.1b, where the same data is used to create a bubble chart of the maximum percentage lost for each configuration, there appears to be a similar trend. Most configurations experience a per-interval packet loss under 20%, but at bitrates beginning at 50 Mbps, there are runs/configurations that experience a higher percentage of packets being lost in an interval. Clear bubbles of packet loss appear at 33% (integer multiples of two successes followed by one loss), 50% (integer multiples of one success followed by one loss), 66% (integer multiples of one success followed by two losses) and 80% and greater (integer multiples of one success followed by four or more losses). As the bitrate increases, configurations with maximum percent loss over 80% increase at a seemingly linear rate similar to the max packet loss seen in Figure 4.1a. For configurations with a bitrate over 125 Mbps, the number of configurations with no packet loss remains almost constant with values ranging from 6,335 to 6,419.

Figure 4.1c represents the distribution of the Mean Success Length for the configurations when grouped by bitrate. Similar to Figure 4.1a and Figure 4.1b, most runs experienced little or no packet loss and therefore had a Mean Success Length at or near 7200. At low bitrates, occasionally a single packet is lost, causing the Mean Success Length to drop slightly to just below 7200. These dropped packets mostly occurred at the beginning or end of the run, therefore not causing the Mean Success Length per run to drop in half. It is important to note that halving would occur any time a run had only two run-lengths of received packets separated by at least one loss. This halving happens on all bitrates higher than 25 Mbps but it is most noticeable at 100 Mbps. The distribution of the Mean Success Length, when the bitrate is over 100 Mbs, appears to be very similar; however, given the results of Figure 4.1a and Figure 4.1b, it is likely the similarities are due to the many small data points in close proximity causing the distributions below 1000 to blend together. There are, however, noticeable differences around zero Mean Success Length, and magnifying the area below 1000 would reveal more differences, confirming that the distributions are different.

4.1.2 Influence of Packet Size Distribution

The combination of the Packet Size Distribution, Mean Packet Size, Packet Size Range, and Packet Size Standard Deviation parameters resulted in 85 valid combinations to be examined to determine if the packet size distribution of the cross-traffic has any effect on the game-traffic packet loss. After grouping the simulation data by packet size and distribution configuration, it appears as if packet size and distribution of the cross-traffic does not have a significant impact on the game-traffic packet loss. However, it does appear that the 128-byte configurations and the realistic distribution did cause slightly more game-traffic packet loss. This was verified with a statistical analysis; investigation as to why this is occurring has been left for future work. The full analysis and charts are in the following section.

Results

Figure 4.2 through Figure 4.4e contains the visualizations of the Maximum Packet Loss parameter distributions. There were too many datapoints to fit onto a single chart, so the results are presented in eleven charts.

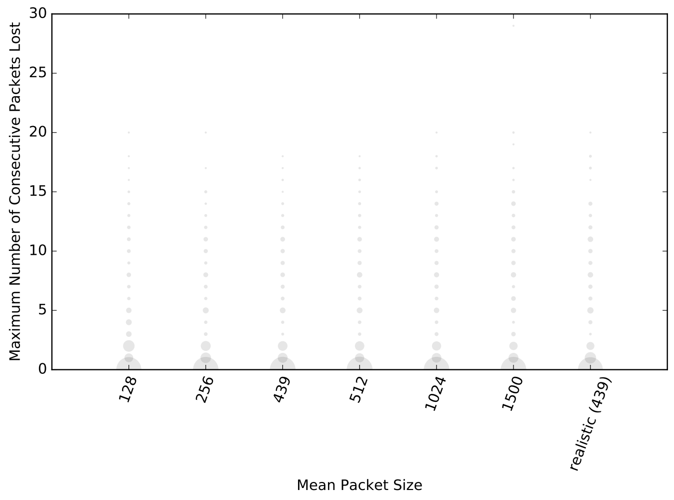
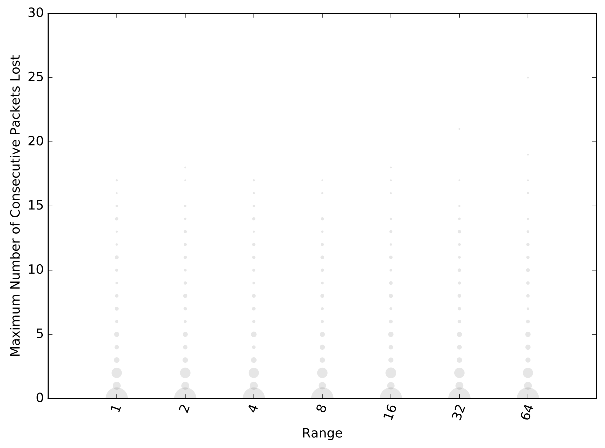
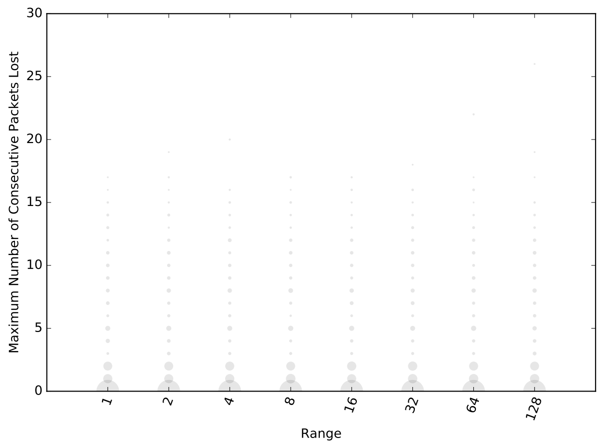


Figure 4.2: Maximum number of packets lost for a Constant and Realistic per packet size distributions

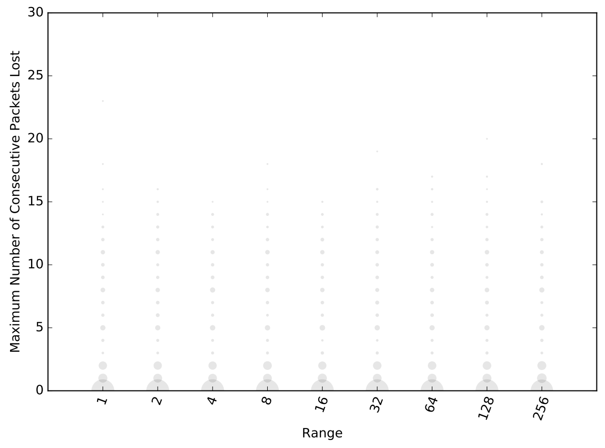
Figure 4.2 shows that the 128-byte configuration has a slightly higher number of runs with a Maximum Packet Loss between two and four than other configurations. A similar pattern emerges in Figure 4.3 and Figure 4.4 where a slight increase in the number of 128-byte configurations with a maximum packet loss of two, three, and four packets when compared to the other configurations for uniform and gaussian distributions of cross-traffic respectively. However, possibly due to the sparse nature of the charts, the results are difficult to see. Looking at the raw data, there is a 2.6% increase in the number of 128-byte configurations with a maximum of four or fewer packets lost (4 packets lost was chosen to match the mitigation analysis in Section 4.3). This would indicate that having smaller packets in the cross-traffic reduced the number of packets lost in the game-traffic. This seems counter-intuitive as one would assume the smaller packets would fill up the router queue quicker (the router queue was configured for 200 packets), reducing the probability that game packets make it into the queue. A possible explanation could be beat frequencies in the generated data, where packets from the two streams would occasionally be generated simultaneously, creating additional packet loss or delay; this would suggest that different random seeds or bitrates may produce different results. However, there is also a negative trend in the number of configurations with four or fewer maximum packets lost as the size of the packet increases. It should also be noted that, even though the 128-byte configurations



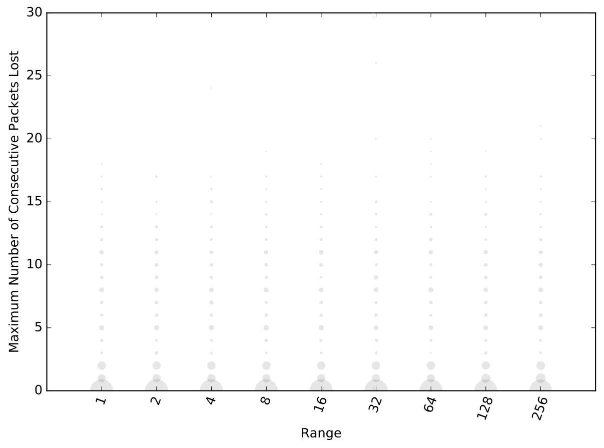
(a) 128-byte configurations



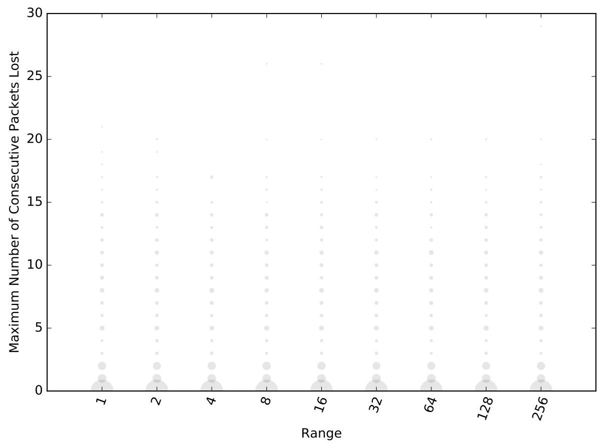
(b) 256-byte configurations



(c) 439-byte configurations

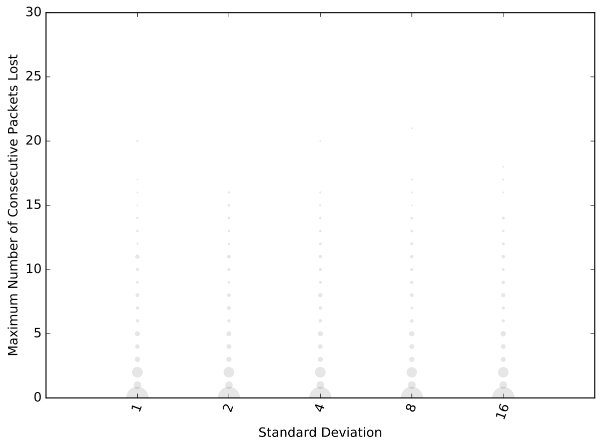


(d) 512-byte configurations

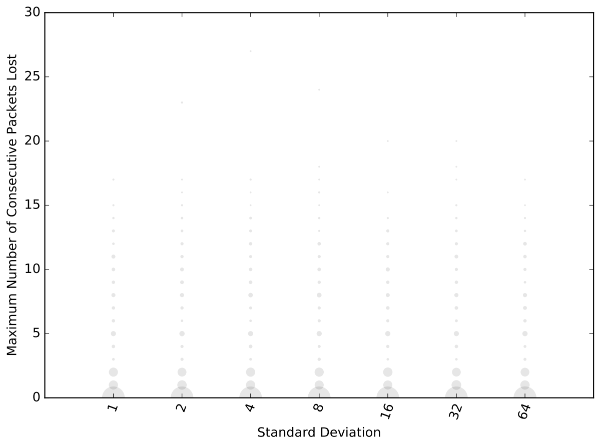


(e) 1024-byte configurations

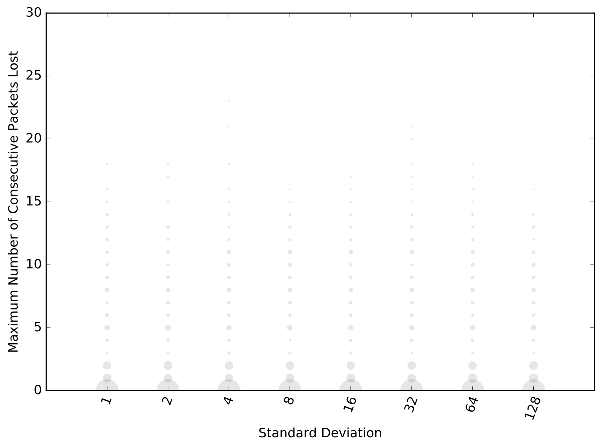
Figure 4.3: Maximum number of packets lost for Uniform configurations per packet size and range



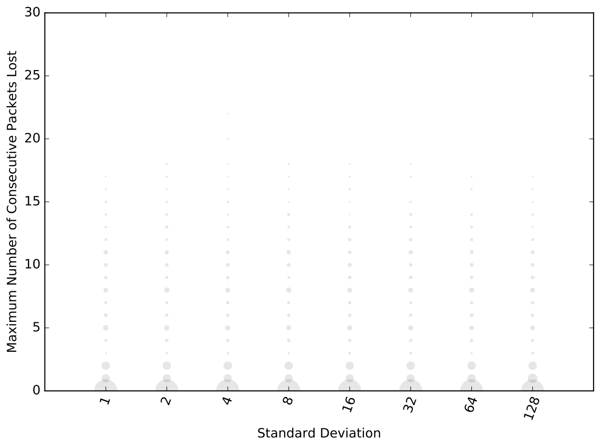
(a) 128-byte configurations



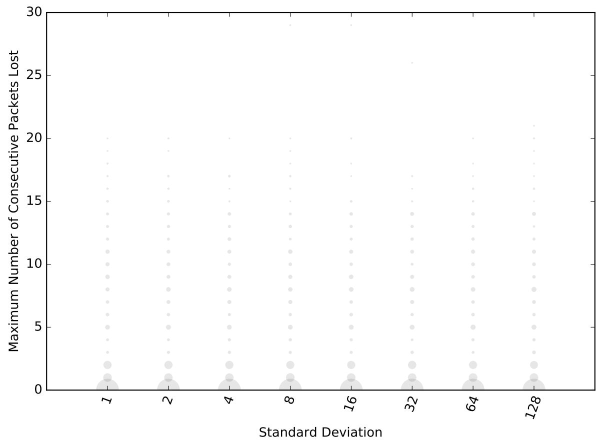
(b) 256-byte configurations



(c) 439-byte configurations



(d) 512-byte configurations



(e) 1024-byte configurations

Figure 4.4: Maximum number of packets lost for Gaussian per-packet size distributions

appeared to have reduced the number of high loss runs, the number of zero loss runs were also slightly reduced, indicating that, under certain conditions, the smaller packets also *increased* the number of packets lost in the very low packet loss scenarios.

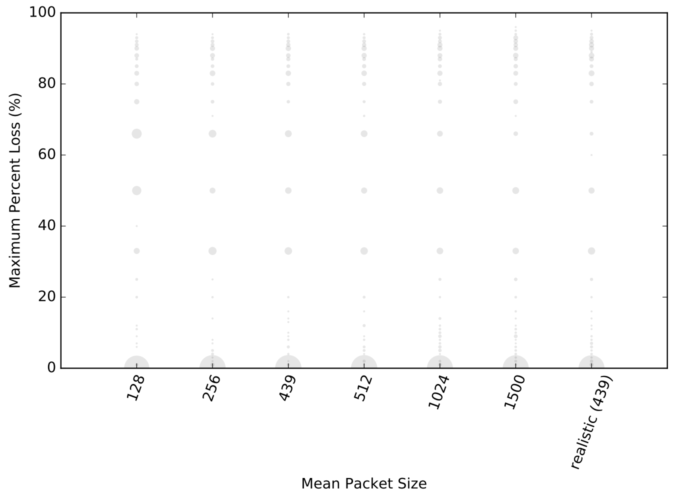
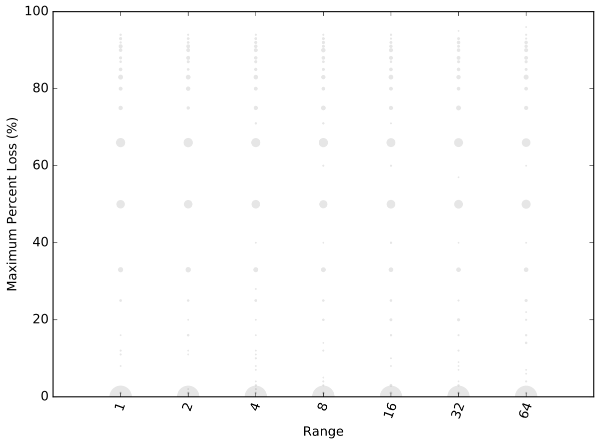


Figure 4.5: Maximum percent packets lost for Constant and Realistic configuration per packet size distribution

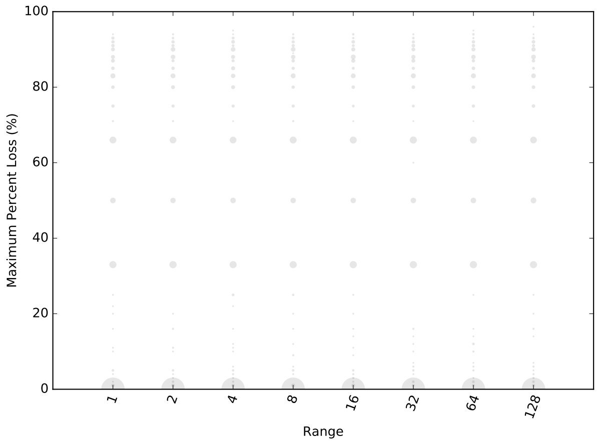
The distributions of the remaining configurations vary slightly, but no other noticeable differentiations appear in these visualizations. Other than the configurations with 128-byte mean packet size, it would appear that the packet size and distribution have little impact on the game-traffic packet loss. There are, however, other patterns visible in these charts that need to be explored further. For each configuration, the largest concentration of runs had no packet loss, followed by two packet losses and then one packet loss. Concentrations of packet losses of 5, 8, and 11 packets occur as well. The packet loss concentration patterns do not appear to be dependent on the packet size, but other configuration groupings may yield different results.

A similar pattern emerges when we look at the distributions of the Percent Packet Loss in Figure 4.5, Figure 4.6 and Figure 4.7. Once again, the 128-byte configurations have a noticeably different distribution than the other configurations. This time, the 128-byte configurations have an increase in runs that have 50% and 66% percent packet loss and slight reductions in the upper and lower percentage losses. However, unlike the Maximum Packet Loss charts, the Maximum Percent Loss charts have more variation between configurations. The 256-byte configurations have a slightly elevated number of runs with 66% packet loss when compared to the larger packet sizes and the 1024-byte and 1500-byte configurations appear to have the fewest numbers of runs with 66% and 33% packet loss. The realistic configuration also appears to have a slightly reduced percent packet loss compared to the other configurations.

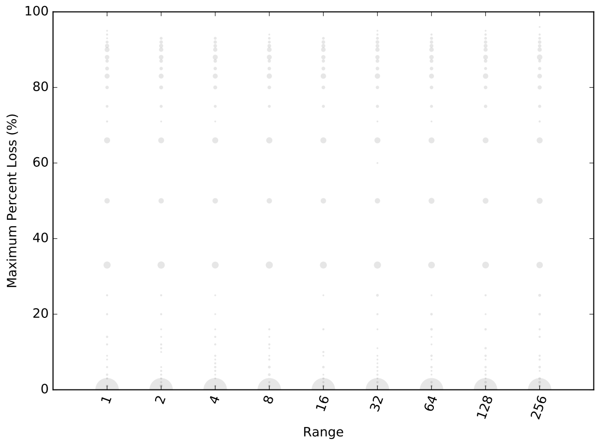
There is a general pattern in which the larger the cross-traffic packet size, the lower the percent packet loss in the game-traffic; however, it is difficult to be certain from these charts. This result seems more intuitive, as



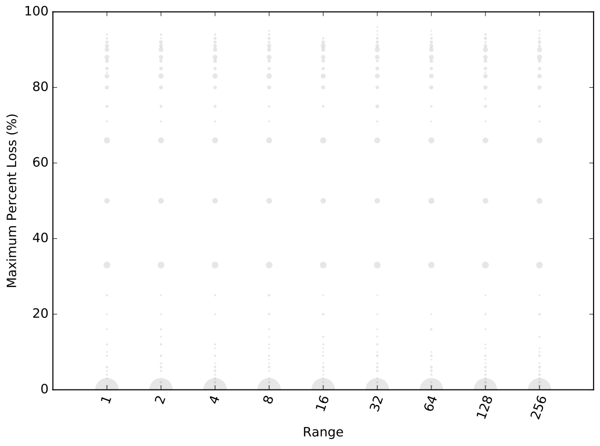
(a) 128-byte configurations



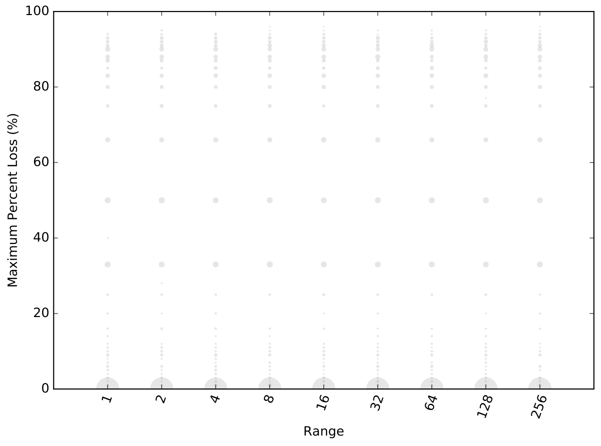
(b) 256-byte configurations



(c) 439-byte configurations

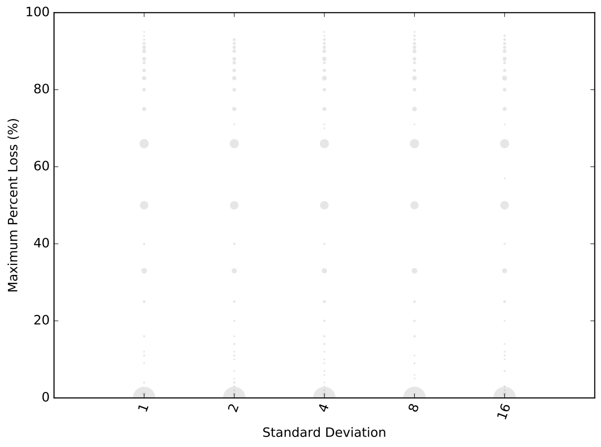


(d) 512-byte configurations

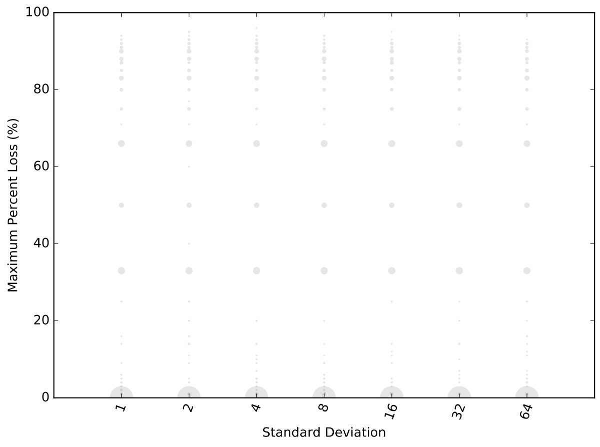


(e) 1024-byte configurations

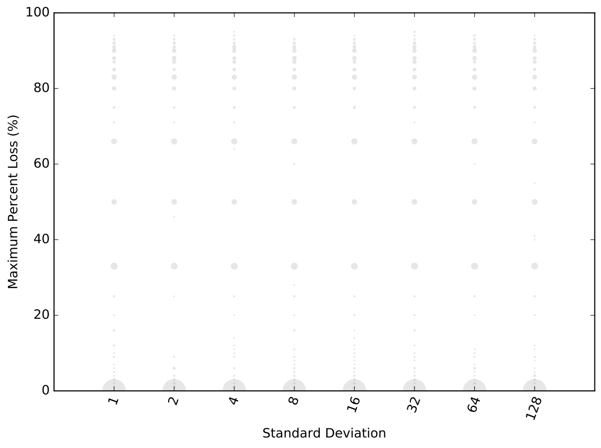
Figure 4.6: Maximum percent packets lost for Uniform configurations per packet size distribution



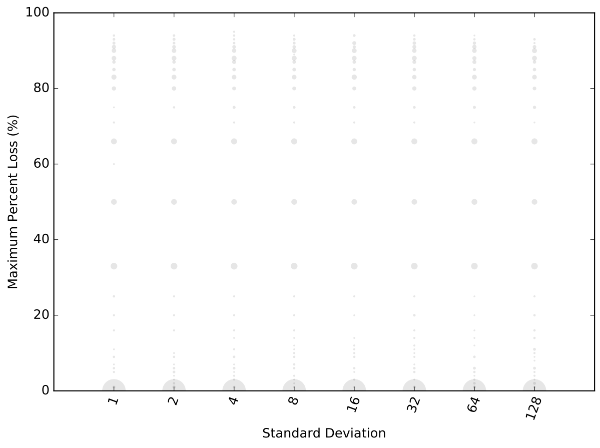
(a) 128-byte configurations



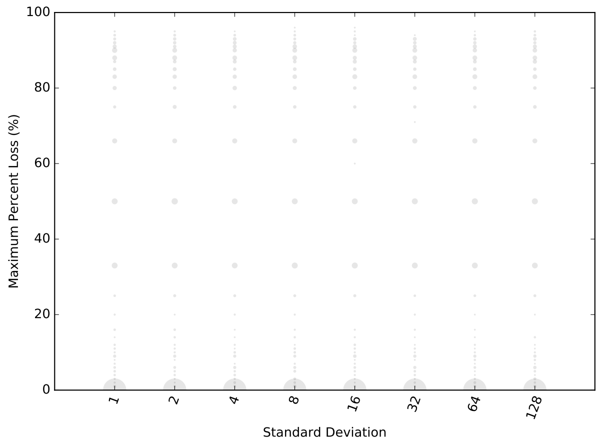
(b) 256-byte configurations



(c) 439-byte configurations



(d) 512-byte configurations



(e) 1024-byte configurations

Figure 4.7: Maximum percent packets lost for Gaussian configuration per packet size distribution

we would expect the larger packets to take up less space in the 200-packet queue than the equivalent number of bytes in smaller packets. Since there is not a similar pattern in the packets lost in the Maximum Packet Loss charts, it appears that the maximum packet loss remains more or less constant (with the exception of the 128-byte packet configurations), but the larger packets in the cross-traffic reduced the overall packet loss in the game-traffic (resulting in slight differences in the percent loss). This would indicate that the packet size and the packet size distribution could have a larger impact than would be indicated by the Maximum Packet Loss results.

Once again, additional patterns occur that cannot be used to distinguish between configurations, but are interesting nonetheless. All configurations have a large concentration of runs with 0% packet loss. Concentrations at 33%, 50%, and 66% are seen which are due to integer loss: success ratios of 1:2, 1:1, and 2:1 or multiples thereof. Similar distributions would be expected in the other percent packet loss charts in this chapter.

Using the Mean Success Length metric (Figure 4.8, Figure 4.9, and Figure 4.10), a clear differentiation for the 128-byte packet configurations can be seen. When the mean packet size is 128 bytes, there is a sharp decrease in the number of packets received in a row with a cross-traffic packet size distribution of Gaussian, Uniform and Constant. This corresponds to similar increases in the packets lost in Figure 4.2, Figure 4.3, and Figure 4.4 and maximum percentage loss in Figure 4.5, Figure 4.6, and Figure 4.7. Unlike the Maximum Percent Loss results, there does not appear to be any significant pattern with any of the other configurations. With these three charts, the number of packets sequentially lost is only slightly increased, while the average number of packets sequentially received significantly decreased indicating that, in some cases, the 128-byte configurations caused short (alternating) runs of successfully received packets and runs of packet loss. This would suggest that the number of run intervals (a run of received packets followed by a run of lost packets) would be increased for the 128-byte configurations. This hypothesis is verified by looking at the Cumulative Distribution Function (CDF) heatmap for the number of intervals per configuration group, as illustrated in Figure 4.11. In this heatmap, each vertical band represents the CDF for one configuration group and the color shade at each point on the vertical axis represents the percentage of runs in the configuration group that has, at most, that number of intervals.

Groupings for each packet-size distribution combination are visible, with only minor differences within each grouping. For example, all of the configurations with a uniform distribution and 128-byte mean packet size have a similar CDF while the next larger packet size (256-byte) has a distinctly different CDF. There appears to be pattern where an increase in the packet size causes a decrease in the number of intervals. While there appear to be slight differences between the distributions, and ranges within distributions, the differences are so small that they are likely due to random fluctuations in the simulations.

As expected from the previous charts, the 128-byte configurations have distinctly different CDFs from the other configurations. All of the 128-byte configurations groups had approximately 70% of the runs with only a single interval (indicating a perfect run). The remaining configurations, however, had approximately

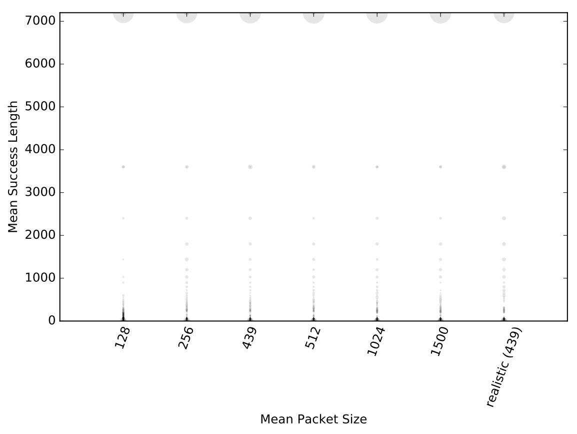
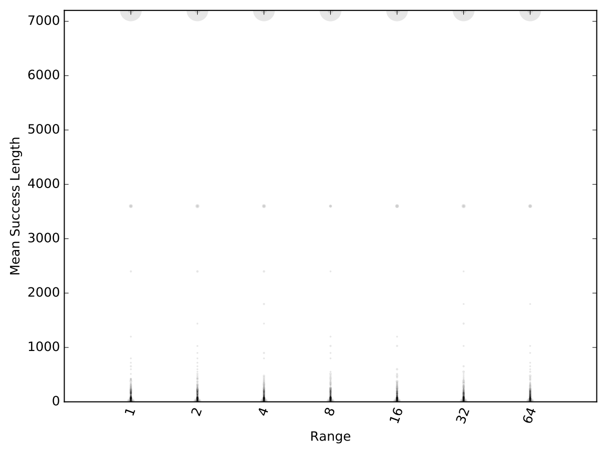


Figure 4.8: Average number of packets received for Constant and Realistic per packet size

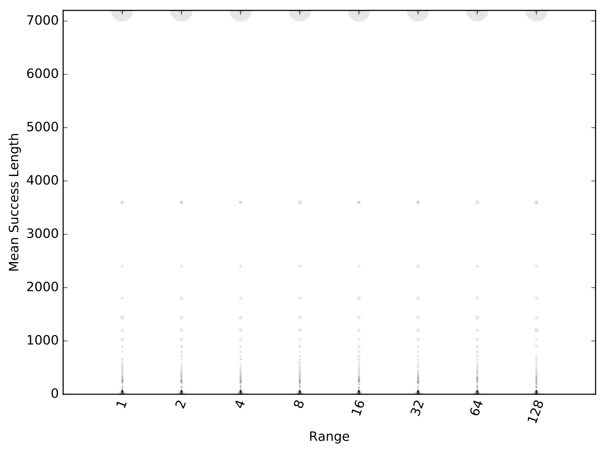
80% perfect runs. Despite the dramatic differences in the starting position for the different CDFs, all of the CDFs reach 85% by 15 intervals per run. After this, the CDFs rise at a similar rate and then diverge significantly when the CDFs reach approximately 90%. The 128-byte configurations appear to diverge more quickly, indicating a significant increase in the number of intervals for these configurations.

An exception to the previous observations occurs for the highest packet size distribution range for the 256-byte, 439-byte, and 512-byte Gaussian distributions. In these instances, the CDF rises fairly quickly, indicating that a larger number of runs for these configurations had six or fewer intervals per run. We can also see that, for these configurations, the CDF rises at a slower rate than the other configurations. However, at the upper end (around 65 intervals) the CDF appears to be similar to the CDF for other packet size ranges within the same packet size-distribution groups. This exception could possibly be explained by the Gaussian distribution reducing some of the packet sizes below the minimum threshold, forcing the simulation to clamp the packet size to the minimum allowable for TCP or UDP protocol. This would artificially increase the number of small packets being generated for the cross-traffic data and decrease the overall bitrate. However, one would expect these instances to rise slower, similar to the 128-packet configurations, unless the decrease in bitrate was sufficient to reduce the packet loss in the game-traffic significantly.

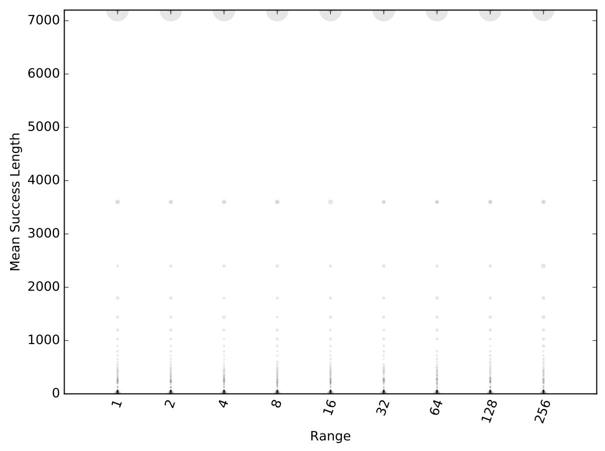
Figure 4.11 also points out differences that do not show up in previous visualizations. Since Figure 4.11 is directly related to Figure 4.8, Figure 4.9, and Figure 4.10, a decrease in the MTBF in the Mean Success Length results as the mean packet size increased would be expected. However, distinguishing between small changes in values in the bubble charts can be challenging, and the CDF chart does not consider packet losses. This would, for example, mean that a run that had one interval and one packet loss would have the same contribution to the CDF as a run that had one interval and 7199 packets lost. Figure 4.8, Figure 4.9, and Figure 4.10, on the other hand, would represent these two runs as 7199 and one respectively. Figure 4.11 does, however, suggest that the packet size is impacting the network differently, and further investigation into the cause of this differentiation may lead to improved network performance or enhanced network classification models.



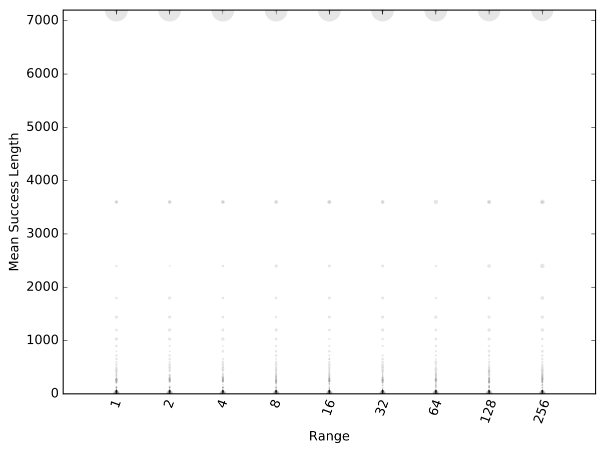
(a) 128-byte configurations



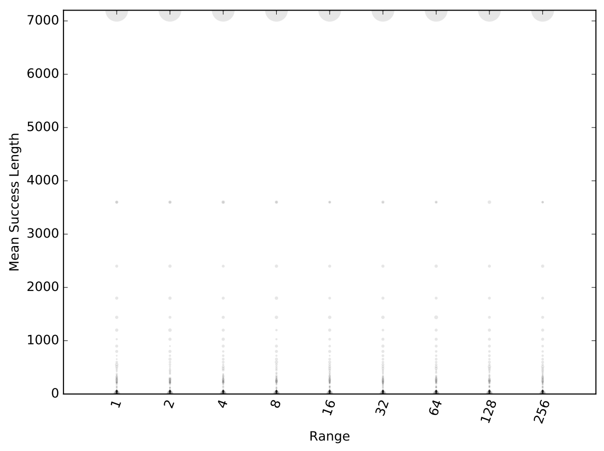
(b) 256-byte configurations



(c) 439-byte configurations

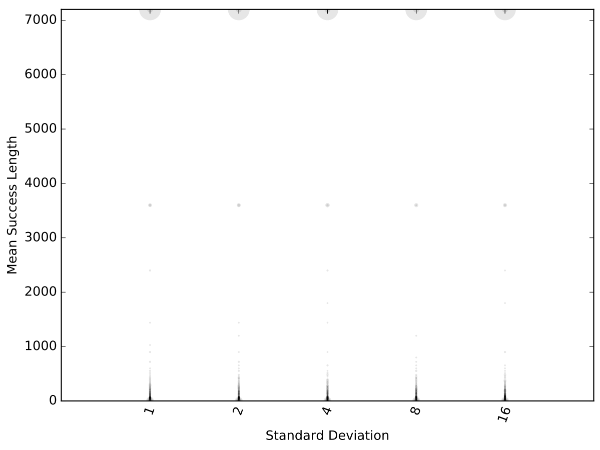


(d) 512-byte configurations

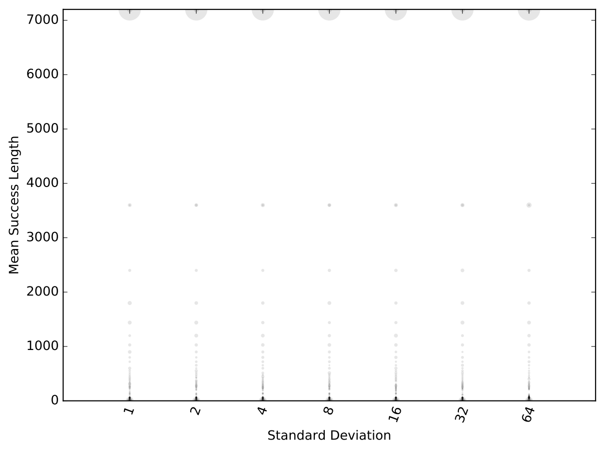


(e) 1024-byte configurations

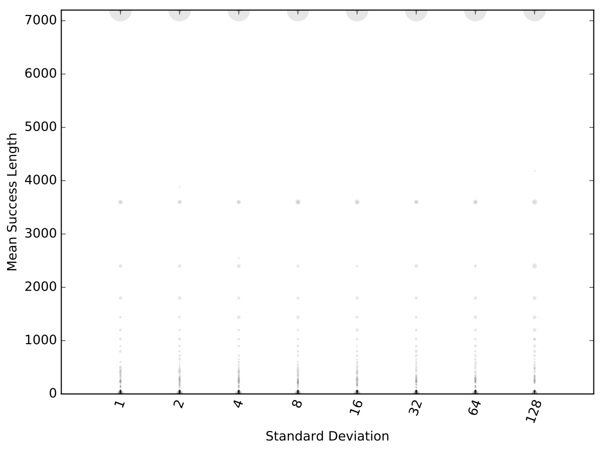
Figure 4.9: Average Number of packets received for Uniform per packet size distribution



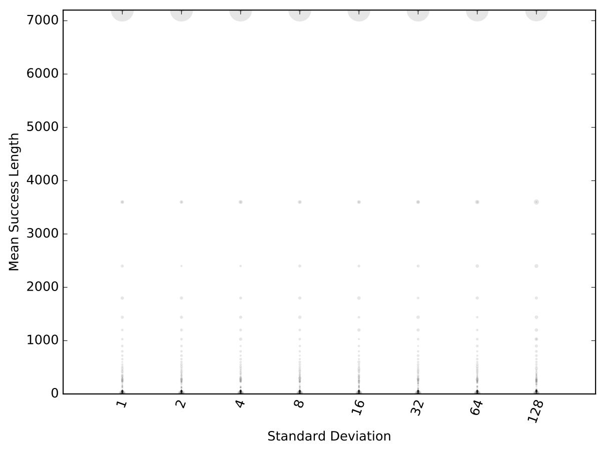
(a) 128-byte configurations



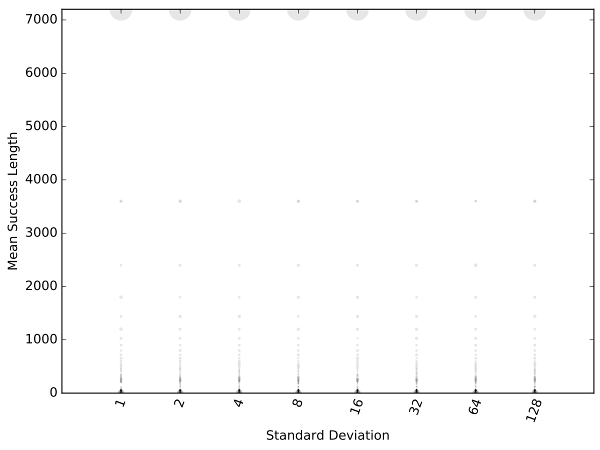
(b) 256-byte configurations



(c) 439-byte configurations



(d) 512-byte configurations



(e) 1024-byte configurations

Figure 4.10: Average Number of packets received for Gaussian per packet size distribution

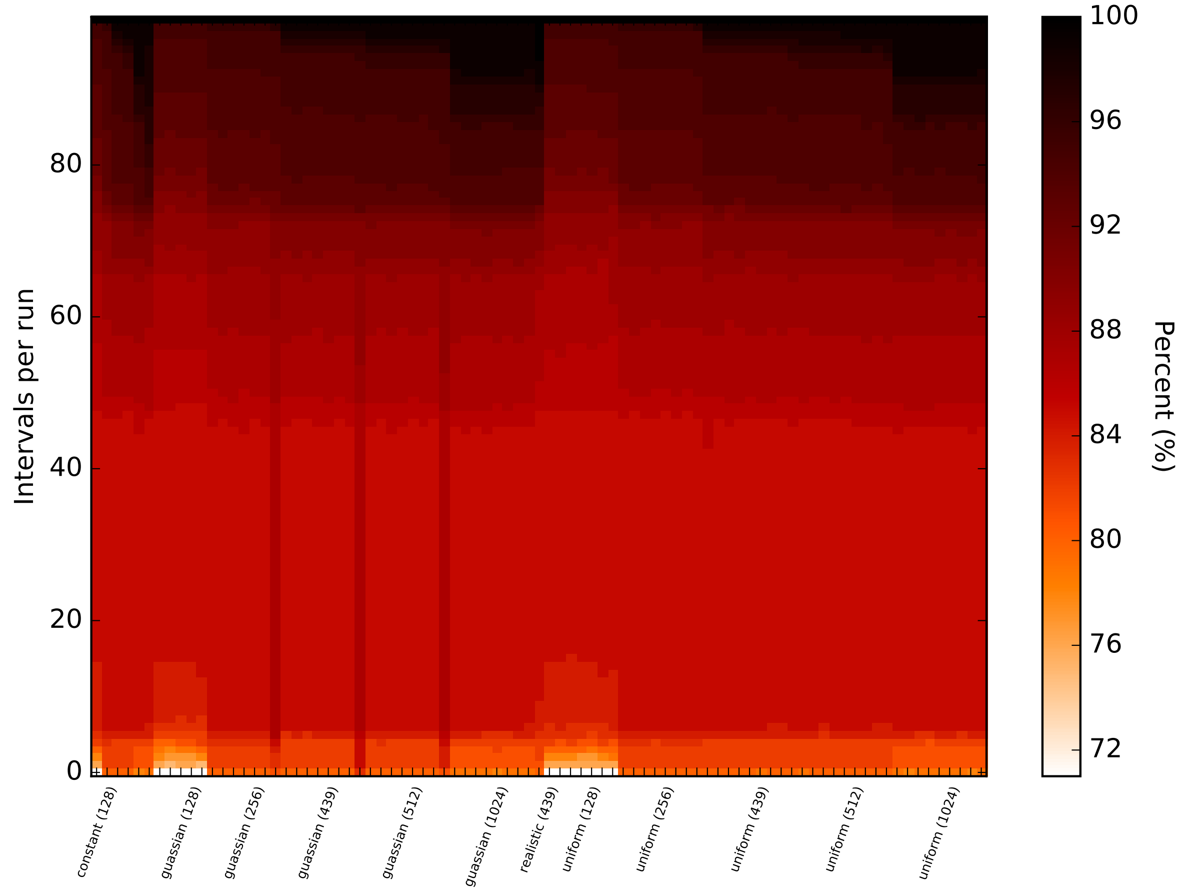


Figure 4.11: CDF of Interval count

Observations from the previous charts indicate that the 128-byte and realistic configurations resulted in different packet losses in the game-traffic, while the other configurations did not appear to vary significantly. To test if there are any significant differences in the game-traffic Maximum Packet Loss due to a change in the cross-traffic packet size distribution, the 2-sample Kolmogorov-Smirnov (ks) Test was used to test the null hypothesis that all game-traffic Maximum Packet Loss distributions are drawn from the same distribution.

The results of the 2-sample ks-test are shown in Figure 4.12. Every configuration group is compared to each other configuration group and the p-value assigned a color gradient according to the scale on right. The null hypothesis states that the distributions of Maximum Packet Loss are the same for each configuration and therefore a p-value of less than 0.05 (5 on the scale) would be sufficient to reject the null hypothesis for that configuration group. The horizontal axis is labeled with the identification of the interesting configurations and the y axis has the same configurations as the x axis, with constant-128 at the bottom.

Inspecting Figure 4.12, there appear to be white or very faintly colored bands vertically and horizontally for the configurations marked as constant-128, and realistic, as well as the configuration groups labeled gaussian-128, and uniform-128. These white bands indicate areas where most of the 2-sample Kolmogorov-

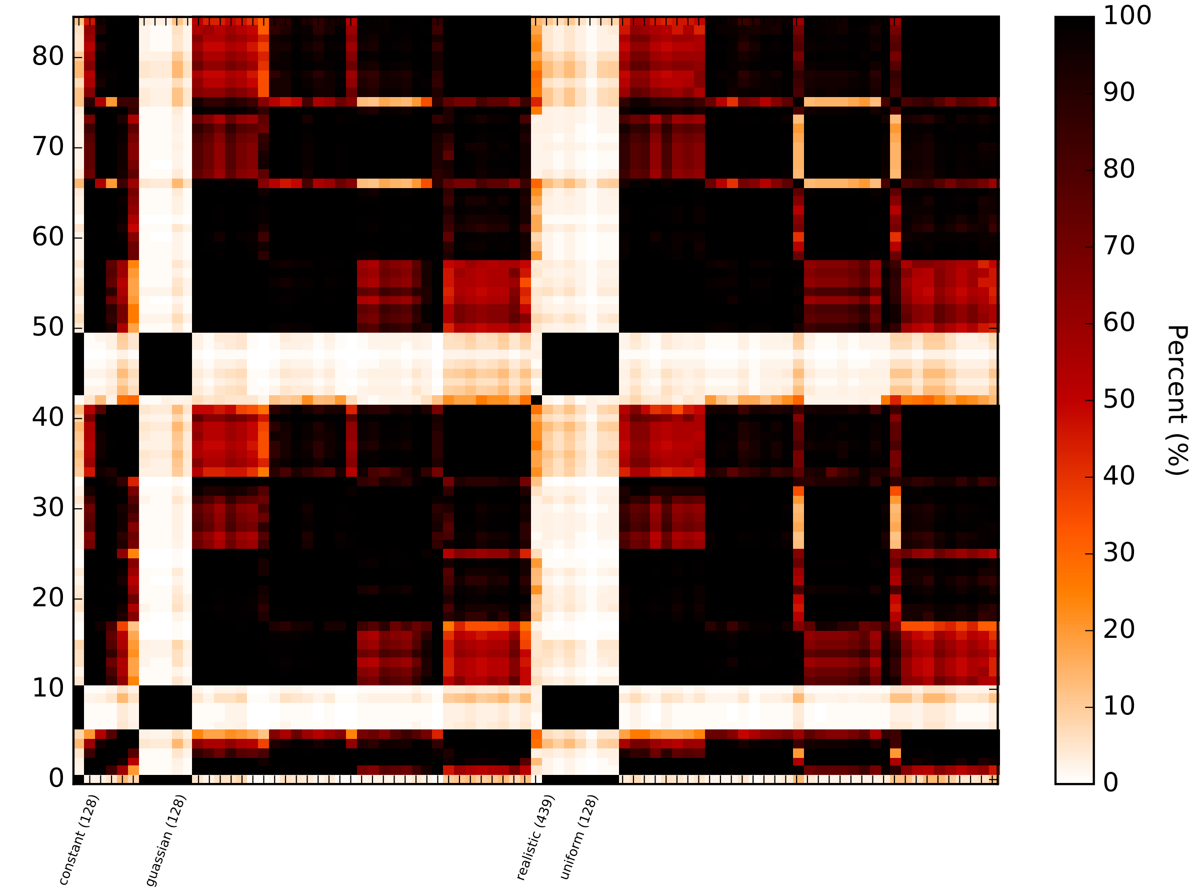


Figure 4.12: Heatmap of p-value from 2-sample Kolmogorov-Smirnov test

Smirnov Tests resulted in a p-value less than the 0.05 threshold, allowing us to reject the null-hypothesis (that states the Maximum Packet Loss were drawn from the same distribution) and indicating that the compared configurations are statistically different. Although it should be noted that some configurations, such as gaussian-128-8 (the second last column in the gaussian-128 section), the comparison between uniform-128 and any 1024-byte configuration (rows 34-41, 75-84), and the comparison between uniform-128 and the uniform-439-256 (row 66) resulted in many values slightly higher than the 0.05 threshold. Additionally, there are a few values scattered within the white areas that exceed the 0.05 threshold. The black squares, present at the intersection of these white bands, indicate a very high similarity between the configuration and itself, and other similar configurations. One further observation is that, other than the previously mentioned configuration groups, no other configuration combination resulted in p-values less than 0.05.

Even though some 128-byte configuration comparisons did not result in significant differences when compared to the other configurations, it is apparent that there is something about the 128-byte configurations that cause many of them to have a significantly different result on the game-traffic than the other configurations. Considering the apparent increase in game-traffic packet loss for the 128-byte configurations, a possible reason for the difference seen for the 128-byte configurations is that the increased number of cross-traffic packets increased the competition for each slot in the router queue and, therefore, reduced the number of game-traffic packets that were able to be transmitted. An investigation into the exact cause of the differentiation of the 128-byte packet configurations has been left for future work.

4.1.3 Influence of Inter-Packet Time Distribution

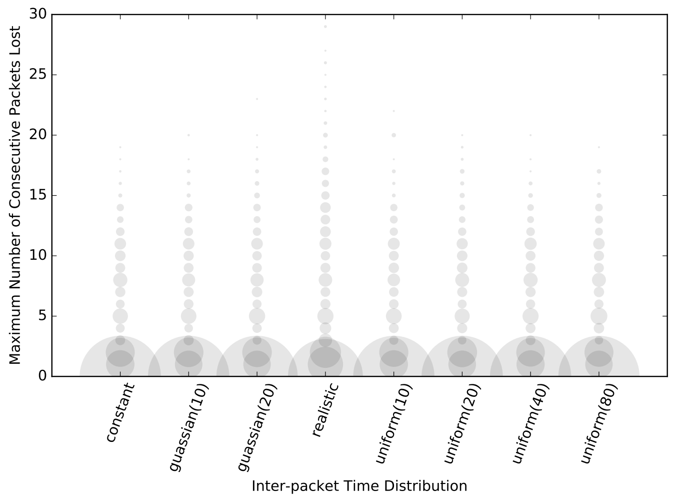
The next analysis focuses on the influence of the timing of the cross-traffic packets on the game-traffic. The Combination of Inter-packet Time Distribution and Inter-packet Time Interval resulted in nine different experimental configurations. After grouping the simulation data by the cross-traffic inter-packet time distribution and interval, it appears that the inter-packet timing has minimal impact on the game-traffic packet loss. However, the realistic inter-packet time distribution did appear to result in more packet loss, confirmed by statistical analysis. The investigation of these results has been left for future work. The full analysis and charts are in the following section.

Results

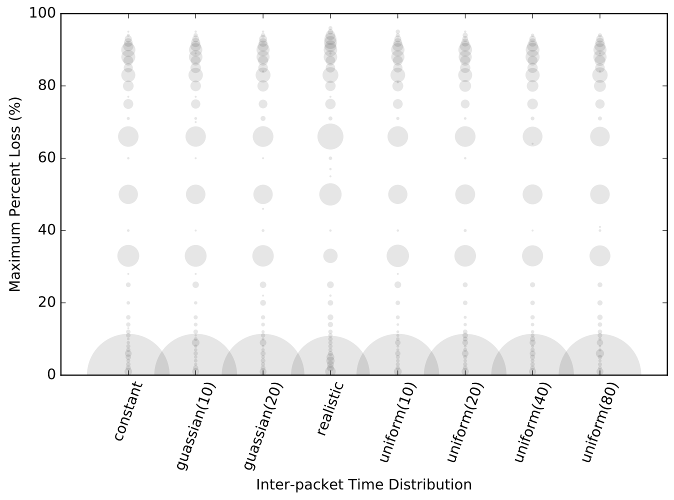
Figure 4.13 contains three charts, representing the distributions of the Maximum Packets Lost, Maximum Percent Loss, and Mean Success Length, similar to the analysis of Packet Size Distribution in Section 4.1.2.

Figure 4.13a represents the distribution of the Max Packet Loss per run, for each of the Inter-packet Time Distribution configuration groups. At first glance, it would appear that the height of the distributions vary slightly, however this could simply be the result of random variation. Closer examination shows that the largest concentration of configurations are once again at 0 packets lost. The next largest concentrations are at one and two packets lost. These three bubbles remain largely unchanged for most of the configurations, with changes only noticeable for the realistic configuration group (where the 0-packets lost bubble is reduced and the 1-packet lost is slightly larger). Each of the configurations then have lower concentrations at 5-packets lost, 8-packets lost, and packets lost for each value up to 15-packets lost. Above 15-packets lost, the bubbles are reduced to points indicating that packet losses beyond that point are probably outliers. The Realistic configuration has more packet losses beyond 15-packets (up to nearly 30-packets). With less than 0.5% of runs with consecutive packet loss greater than 15 packets, packet losses of this degree are low. From this chart, one could reasonably state that the realistic configurations resulted in greater packet loss. While one may be tempted to conclude there are differences between the other configurations, this chart does not provide enough detail to support that conclusion.

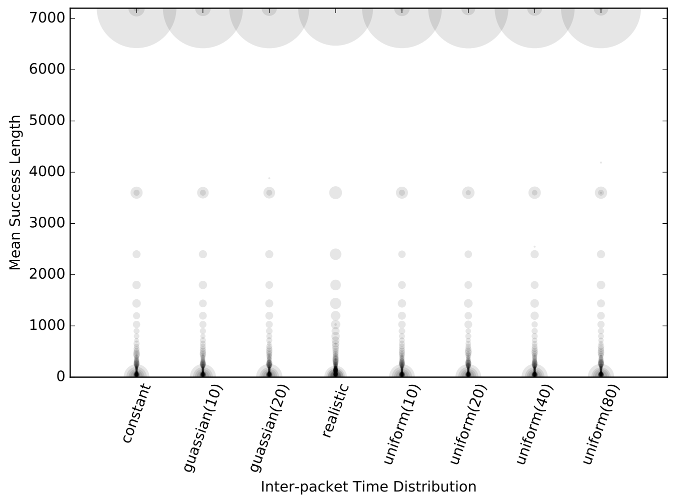
Figure 4.13b the shows bubble chart of the Maximum Percentage Loss for the Inter-packet Time Distribution configurations. For most configurations, the distribution remains the same. However, as before, the realistic configuration group stands out. The realistic group has fewer configurations with up to 33% packet loss per interval and an increase in configurations with up to 50% and 66% packet loss per interval and minor differences in most visible values above 66% maximum packet loss per interval. Together these



(a) Maximum consecutive packets lost



(b) Maximum number of consecutive packets lost



(c) Mean consecutive packets received

Figure 4.13: Inter-packet time distribution

results indicate that the realistic configurations experienced a higher percentage of packet loss than the other configurations. It should be noted that the high concentration of runs at 0% appears to remain constant for all configurations, yet there were nearly 1000 fewer runs at 0% for the realistic configuration. For the Realistic configurations, the decrease in perfect runs caused an increase not only as noted above, but also to values between 1% and 25% percent loss. However, the values at these percentages still remained low and are virtually invisible on this chart.

Figure 4.13c indicates the average time a game could run without experiencing a packet loss; a bubble chart showing the distribution of Mean Success Length (which is the average size per run, of a string of received packets). For most runs, the average size was approximately 7200 packets, indicating a run with minimal or no packet loss. This chart shows only minor changes between configuration groups with the realistic configuration once again having the most significant differences: a decrease in runs with no packet loss and an increase in runs with an average successful run length of 3600 or less.

The Inter-packet Time Distribution had little influence on the maximum consecutive packets lost. The most interesting configuration is the realistic inter-packet time distribution, with a slightly higher number of consecutive packets lost, and thus, higher percentage loss and reduced average success run time then the other configurations. To determine if the realistic configuration is statistically different from the other configurations, the Two-Sample Kolmogorov-Smirnov Test was calulated, testing the null hypothesis that the Maximum Packets Lost distributions were drawn from the same distribution. Looking at the p-values from the Two-Sample Kolmogorov-Smirnov Test in Table 4.1, when comparing the realistic configuration to the other configurations, a p-value lower than the 0.05 threshold is obtained, indicating the null hypothesis can be rejected, concluding that the game-traffic packet loss as a result of realistic Inter-packet Time Distribution is statistically different than the other configurations. The remaining configurations, when compared to each other, resulted in p-values much higher than 0.05 and therefore, are not statistically different from each other.

Table 4.1: Inter-packet time distributions p values from the Kolmogorov-Smirnov test

	Constant	Gaussian(10)	Gaussian(20)	Realistic	Uniform(10)	Uniform(20)	Uniform(40)	Uniform(80)
Constant		1.00	1.00	0.00	1.00	1.00	0.994	0.512
Gaussian(10)			1.00	0.00	1.00	1.00	0.989	0.484
Gaussian(20)				0.00	1.00	1.00	1.00	0.907
Realistic					0.00	0.00	0.00	0.00
Uniform(10)						1.00	0.999	0.764
Uniform(20)							0.981	0.744
Uniform(40)								0.998
Uniform(80)								

However, the realistic cross-traffic Inter-packet Time Distribution does have an impact on the game-traffic. It appears that there is some property of the realistic inter-packet spacing that contributes to the packet loss. One explanation could be the bursty nature of the realistic cross-traffic. The bursts *would* create periods where very few game packets would make it through the router, and *would* create periods where game-traffic would have little to no congestion. This *would* increase the number of packets lost at lower bitrates, due to the transient periods of high load, and reduce the packets lost at high bitrates, due to the transient periods of low load. This result, unfortunately, does not help us with classifying traffic or attempting to predict or react to changing network conditions. However, one could postulate that a more disciplined or predictable packet forwarding algorithm could potentially reduce packet loss in low bandwidth applications that are competing for time on the network. Further research would need to be done to come to any conclusion.

4.1.4 Influence of the traffic classifier

The focus so far has been on determining if the cross-traffic data characteristics have any impact on the game-traffic packet loss. Now the focus moves to the router itself, to determine if the routing algorithms can influence the game-traffic packet loss. In this section, the algorithm for classifying the traffic is modified to determine if separating game-traffic from cross-traffic, or the number of queues, have any impact on the game-traffic packet loss. In this section, the Classifier Type and Number of Queues parameters are varied for a total of five different configuration groupings. As expected, classifying the game-traffic and cross-traffic separately into different queues resulted in zero game-traffic packet loss. Increasing the number of queues increases the number of runs with zero packet loss, but also increases the number of runs with more than five consecutive packets lost. The full analysis is in the following section.

Results

Figure 4.14a is a bubble chart analyzing the number of configurations that experienced different levels of Maximum Packet Loss, where X-axis is the classifier type:

- 3-dedicated: Indicates there are three queues and the game-traffic has been dedicated to a single queue and the cross-traffic has been assigned to the remaining queues.
- 2-dedicated: Indicates there are two queues and the game-traffic has been dedicated to a single queue and the cross-traffic has been assigned to the remaining queue.
- 3-random: Indicates there are three queues and the game-traffic has been dedicated to a single queue and the cross-traffic is randomly distributed between all three queues.
- 2-random: Indicates there are two queues and the game-traffic has been dedicated to a single queue and the cross-traffic is randomly distributed between both queues.
- 1-shared: All traffic goes through one single queue.

A round robin scheduler was used in all cases to determine what queue to forward packets from. As one would expect, having the game-traffic in its own queue, separated from the cross-traffic, appears to eliminate all game-traffic packet loss. In the dedicated instance, the number of queues did not appear to have any impact on the game-traffic.

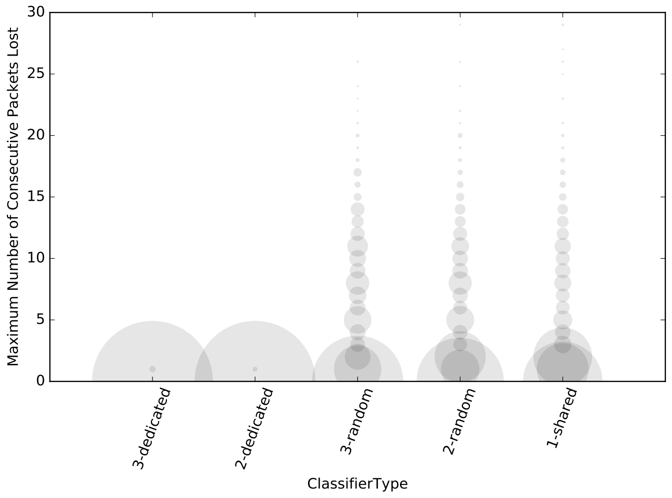
However, it is expected that the game-traffic could experience greater latency in the dedicated configurations due to other queues needing to be serviced, however this increase in latency depending on the number of queues, priorities, and amount of cross-traffic. In this instance, however, the game-traffic may actually experience lower latency in the dedicated queues than other queues because the game-traffic would only be queued for, at most, the amount of time it would take to transmit two cross-traffic packets (instead of a time proportional to the cross-traffic rate as would be experienced in the random and shared configurations). Latency due to cross-traffic, however, is outside the scope of this thesis and could be considered for future work.

Looking at the random and shared configurations (where the cross-traffic is randomly distributed between one, two, and three queues), there appears to be an inverse relationship between the number of queues and game-traffic packets lost. However, this trend does not extend to each of the Maximum Packets Lost values. For example, the number of configurations with Maximum Packet Loss of seven packets increases with the number of queues while the number of configurations with a Maximum Packet Loss of two decreases with the number of queues. For the number of configurations with four or fewer packets lost, an inverse trend appears, an increase in queues decreases the number of configurations.

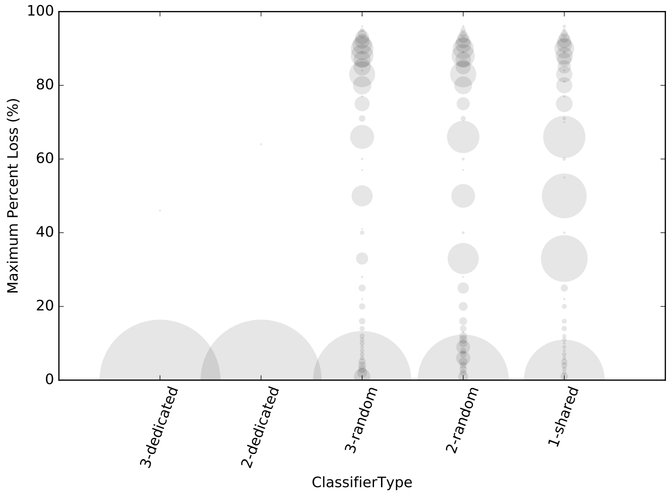
It should be remembered that the router configuration is an oversimplification of a real router and there will rarely be an instance where a router would classify a stream into a queue of its own (unless the router offered QoS guarantees or traffic classes). Additionally, the packet based-queues used may differ significantly from queues that use a fixed byte sized buffer to limit the number of packets. Schedulers other than round robin would also have an impact on the traffic, another investigation for future work.

Figure 4.14b represents the number of configurations that experienced specific Maximum Percent Loss. Once again, the dedicated configurations experience very little or no packet loss. Similarily to Figure 4.14a, there is an inverse relationship between the number of queues and the number of configurations with 0% loss for the random/shared configurations.

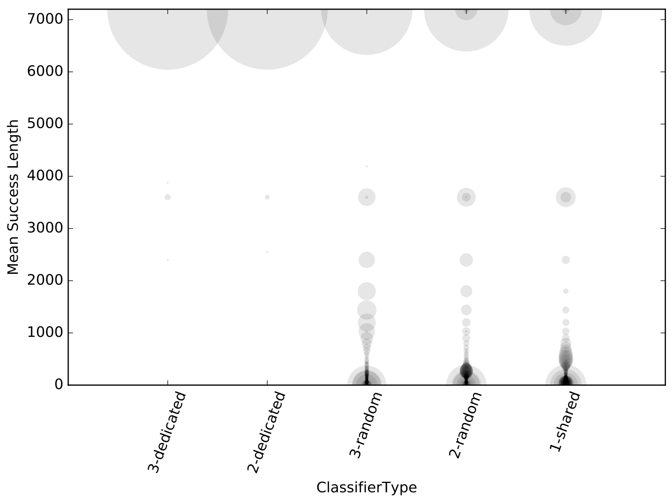
The number of configurations with 0% packet loss is up significantly, with about 15% for the three-queue random configuration, 10% for the two-queue random configuration and 3% for the shared configuration. This increase may seem odd, but remember that percentage loss is calculated as $100 * loss/(loss + success)$ which means that if $success > (199 * loss)$ the result will be less than 0.5% and, therefore, effectively 0%. This does suggest that many of the configurations that did not have a zero Maximum Packets Loss had a significantly larger number of received packets than lost packets in each interval. The random and shared configuration groups also experience an inverse relationship for the 33% and 50% Maximum Percent Loss where, the larger the number of queues, the smaller the number of configurations in that configuration group. Each of the random and shared configurations have a similar number of configurations that experience more than 50% Maximum Percentage Loss. However, the actual distribution appears to be very different between



(a) ClassifierType MaxLost



(b) ClassifierType MaxPercent



(c) ClassifierType MeanSuccess

Figure 4.14: Classifier Type Distribution

the shared and random configurations.

Figure 4.14c shows the Mean Success Length distribution for each of the classifier type configuration groups. Again, the dedicated configurations experienced no packet loss, and therefore had a Mean Success Length of 7200. Of the remaining configurations, there is a clear trend where the increase in queue count increases the Mean Success Length. Figure 4.14c confirms previous observations: an increase in the number of queues decreases the number of configurations with a MTBF below 1000 packets and increases the number of configurations with near perfect runs. The number of runs with MTBF between 1000 packets and 4000 packets also increases, indicating that more packets make it through the router as the number of queues increase. This is expected; since each queue was configured with a size of 200 packets, as the number of queues increase, the number of packets required to fill all the queues also increase. Additionally, the queue containing the game-traffic fills slower, allowing more game packets to be queued and transmitted.

The simplistic classifier type was intended to determine if the number of queues, and how the packets were distributed, had any impact on the game-traffic. Figure 4.14 indicates that having the game-traffic filtered separately from a majority of the cross-traffic significantly decreased the probability of dropping packets from the game-traffic. When game-traffic and cross-traffic packets share a queue, dividing the cross-traffic into multiple queues has an impact on the game-traffic packet loss. Interestingly, as the number of cross-traffic queues increase, the number of configurations with no packet loss, as well as the number of configurations with five or more packets lost, increases. Figure 4.14 additionally indicates that increasing the number of queues increases both the number of high percentage losses (more packets lost than received) and the low percentage losses (more packets received than lost). This result indicates that more queues increases the volatility of the network. While this example does not necessarily reflect real world scenarios, the results here suggest that packet and packet stream classifiers can have a large impact on the performance of the game-traffic. If game-traffic is classified along with bursty traffic, such as data streaming or web traffic, the game data could easily be overwhelmed by the cross-traffic. However, having the game-traffic separated allows the game-traffic to pass through with minimal impact on the cross-traffic.

The scheduler, in this instance, was a simple round robin scheduler but fair or weighted fair queues would likely present a similar result. This is because, unless the game-traffic is queued separately from the cross-traffic or placed in a queue with a high priority, the relatively small amount of game-traffic will always have to compete with the larger volume of cross-traffic. As the number of queues increase, the queue containing the game-traffic will also be competing with other queues. Additionally, with the increase in the number of queues, the packet latency will increase, causing additional problems for online games.

4.2 Extended Attributes

Upon review of the prior results, additional parameters were investigated for the traffic streams and queue algorithms. Due to the exponential increase in the number of runs required to complete the full combination of parameters, many of the parameters from the bitrate, packet-size, inter-packet timing, and traffic classification that did not appear to influence the game-traffic packet loss were dropped. This resulted in a dataset that only explored the new, extended parameter set without much consideration for the interactions between parameters. The following sections are stand-alone results that cannot be directly compared to previous charts and results. The new configuration parameters were presented in Table 3.3 and Table 3.4.

4.2.1 Influence of Traffic Stream Type

With the extended configuration for Traffic Streams Type, there are now seven configuration groups to explore. In this experiment, the cross-traffic was divided evenly between several different stream combinations for a total of seven groups. UDP cross-traffic streams have a detrimental impact on game-traffic packet loss. A single TCP cross-traffic stream has the least packet loss, and an increase in TCP streams increases the packets lost. Combining TCP streams with UDP streams decreases packet loss; however, the results may be due to experimental design. The complete analysis and results follow in the next section.

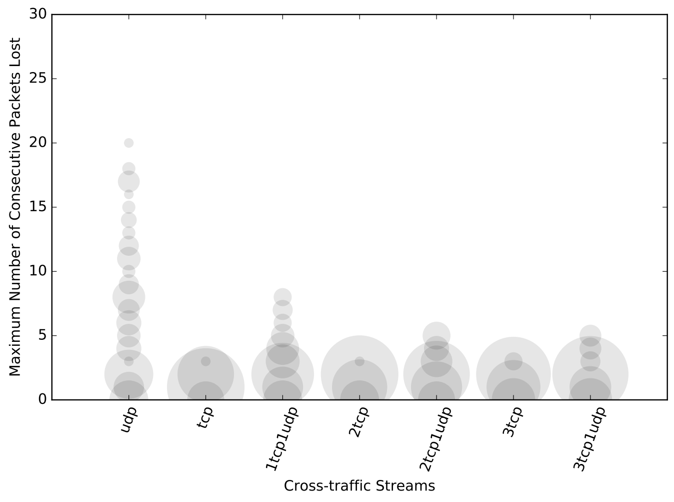
Results

Figure 4.15a is a bubble chart representing the distribution of the Maximum Packets Lost for each configuration group. Since TCP congestion control reduces the bitrate of a TCP stream to help eliminate packet loss, the cross-traffic reaches an equilibrium with the game-traffic after the initial period where the network may be congested. This can be clearly seen in this chart – any configuration containing at least one TCP traffic stream had fewer consecutive game packets lost compared with UDP only traffic.

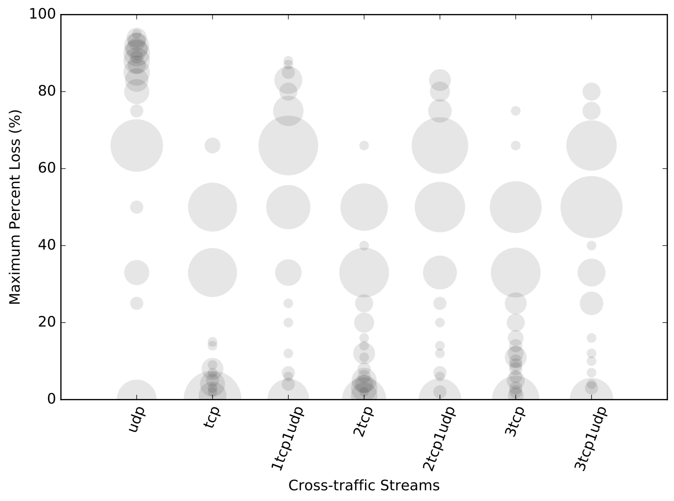
The UDP-only cross-traffic stream has the most consecutive packets lost, with a near uniform distribution of packet loss that trails off after 15 packets (with a larger concentration at two and eight packets). Across all configurations, the 0-packet loss bubble appears to be relatively constant, probably due to the remaining 75 Mbps bitrate configuration that would have near zero packet loss.

It should also be noted that the use of only TCP cross-traffic does not eliminate packet loss, but just reduces the maximum consecutive packets lost to one or two. While an increase in the number of TCP streams seems to increase the amount of packet loss, if there is a UDP stream involved, an increase in the number of TCP streams decreases the packet loss in the game-traffic. This decrease in packets lost is more likely due to the decrease in the bitrate allocated to the UDP stream than the inclusion of another TCP stream. The increase in packets lost due to the increase in the number of TCP streams alone is likely due to the TCP streams competing for bandwidth before settling into equilibrium.

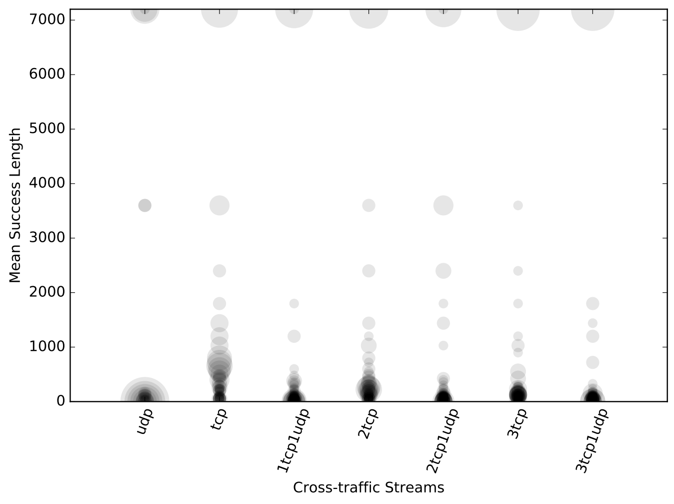
Examining Maximum Percent Loss in Figure 4.15b, two patterns appear to emerge. First, when there is a UDP stream, an increase in TCP streams reduces the percentage of packets lost. Secondly, in TCP-only configurations, the increase in the number of TCP streams increases the Maximum Percent Loss between 0% and 33% (meaning that increasing the number of TCP streams increases the run-length of packet losses), but combined with the results of Figure 4.15a (where only a few extra losses occur), increasing the number of TCP



(a) Maximum number of consecutive packets lost



(b) Maximum percentage of packets lost



(c) Mean number of consecutive packets received

Figure 4.15: Traffic Stream Type

streams does not seem to significantly decrease the run-length of received packets. The UDP configurations have a high concentration of Maximum Percent Loss at 66%, indicating that, for many intervals, the lost packets are double the number of packets received. There is also a large concentration at 50% and 33% for the configurations containing a TCP packet stream. These high concentrations for TCP may be due to the rate limiting algorithm; however, closer inspection of the temporal distribution of the packet loss would be required to support that conclusion.

Figure 4.15c is a bubble chart of the distribution of the Mean Success Length, indicating how long a game could possibly run between packet losses. This chart is somewhat difficult to read due to the high concentrations of configurations with results below 1000 packets. However, this does indicate that a large number of configurations would be virtually unplayable. The configurations with three TCP streams have the most configurations at or near the 7200 successful packets-received mark. One conclusion that can easily be made is that, in the conditions of the extended configurations, the UDP cross-traffic configurations resulted in very few runs where the game ran for a minute (≈ 3600 packets) or two minutes (≈ 7200 packets).

Overall, it appears that UDP traffic has a more detrimental impact on the game-traffic as compared to TCP. If the network administrators controlled the use of UDP, or application designers reduced the usage of UDP in high bandwidth applications in favor of TCP (or another protocol that adjusted to network conditions) the impact of cross-traffic on low bandwidth UDP streams such as those used by video games could be reduced. However, an increase in TCP cross-traffic also appears to cause packet losses in game-traffic, possibly due to the initial burst of traffic that is then throttled. Realistically, we can assume that TCP streams would not be initiated at the same time and would not be going through the process of congestion control at the same time, so the distributions may vary in real data. In this experiment the increase in number of TCP streams reduced the bitrate of the UDP traffic, which is unrealistic as, in the Internet, the attempted bitrate of independent data streams are unrelated in actual network deployments. In the simulated deployments of UDP and TCP, there are some differences that could occur in the real Internet, but without taking a real network trace and applying the analysis we cannot come to any conclusions. Further investigation of the impact of TCP/UDP cross-traffic streams on game-traffic have been left for future work.

4.2.2 Influence of Queue Management Type

The next parameter examined is the Queue Management Type, which determines if and when a packet is dropped upon packet arrival at the router. The simplest implementation, DropTail, drops packets once the queue is full. AdaptiveRED and RED follow a similar algorithm that randomly drops packets, with increasing probability, as the queue fills up, AdaptiveRED is an extension to the basic RED algorithm that attempts to dynamically adjust the probability of packets being dropped, based on the average queue length. CoDel is an algorithm that attempts to minimize the length of time packets are in the queue, and does this by dropping packets that have been in the queue for too long.

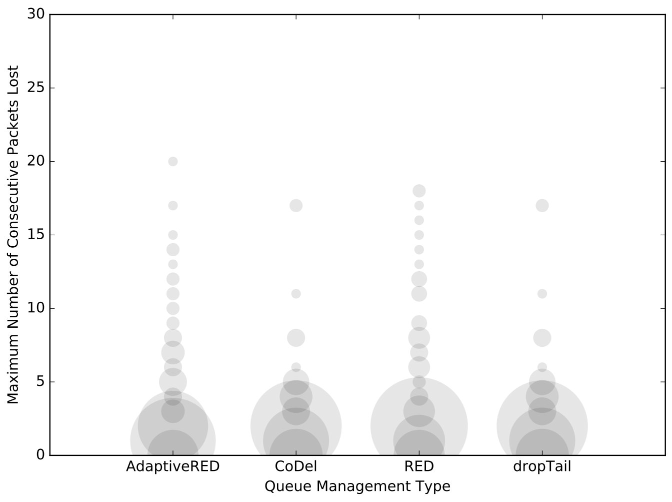
The RED and AdaptiveRED algorithms behave similarly, with only minor differences in game-traffic packet loss. The CoDel and DropTail algorithm appear to have identical results indicating packets may not be in the queue long enough to trigger the CoDel algorithm. Overall, it appears that the more passive DropTail algorithm performs best, resulting in the fewest game-traffic packets lost. The full analysis and results are in the following section.

Results

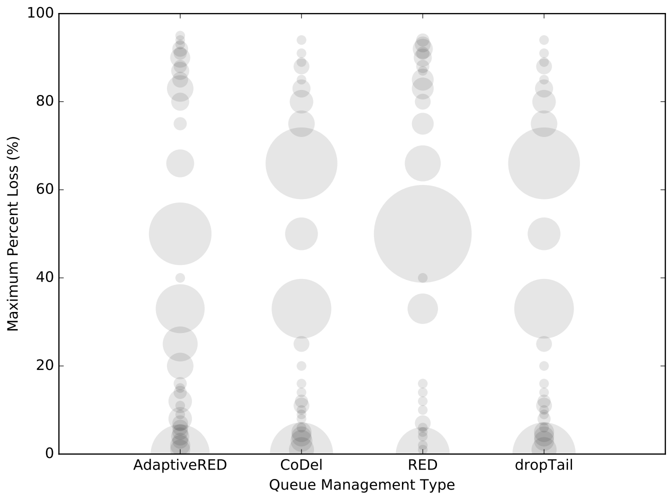
Figure 4.16a depicts the Maximum Packet Lost parameter and already a clear pattern emerges. CoDel and DropTail appear to be almost identical. AdaptiveRED and RED are very similar with only slight differences in the distribution at the low end. From this observation, one reasonable hypothesis is that the queue size (200 packets) is too small for CoDel to be effective, effectively becoming DropTail, and that the RED configuration appears to be similar to the result of the dynamic configuration of AdaptiveRED. The largest difference between AdaptiveRED and RED is the number of configurations with one and two consecutive packets lost; RED has 40% and 27% of the configurations with a maximum consecutive packet loss of one and two respectively while AdaptiveRED has 14% and 52% of the configurations with one and two consecutive packets lost respectively. Overall, DropTail and CoDel drop fewer game-traffic packets than the RED variants. This may be due to the aggressive nature of RED, where it starts dropping packets even when the queue is not yet full.

Figure 4.16b visualizes the distribution of Maximum Percent Loss. In this chart, once again the CoDel/Droptail and RED/AdaptiveRED pairs appear somewhat related. AdaptiveRED drops a lower percentage of packets than RED (with RED having an increase at 50% and AdaptiveRED having an increase in configurations below 50%), though they both have similar shapes at the top end of the distribution. The RED configurations perform better than CoDel and DropTail, where CoDel and DropTail have an increase in configurations with runs having 66% packet loss. However, the increase in the number of configurations with runs experiencing over 80% packet loss for RED/AdaptiveRED, compared to the relatively few configurations for CoDel/DropTail, suggest that (under some circumstances) the DropTail/CoDel configurations may outperform the RED Variants. Although the reduction in packet losses between zero and 50% may make it look like RED is performing better than AdaptiveRED, RED has the fewest configurations with 0% packet loss and a significant increase in packet losses at or above 50%.

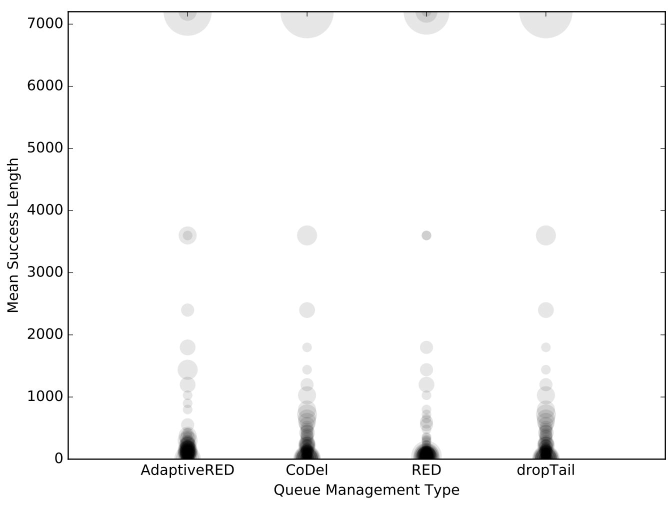
Figure 4.16c represents the distribution of Mean Success Length grouped by Router Queue Type. Not unsurprisingly, given the results of the Traffic Stream Type experiments, most runs performed poorly and resulted in a fairly low Mean Success Length. There are no clearly distinct configuration group(s). It appears that CoDel and DropTail have the most configurations at, or near, 7200 packets and again the most configurations at, or near, 3600 packets. This indicates that the CoDel and DropTail configurations allowed the game to run longer than the RED variants. Of the RED variants, it appears that AdaptiveRED allowed more configurations to run longer than RED, which is consistent with the observations in both Figure 4.16a and Figure 4.16b



(a) Maximum number of consecutive packets lost



(b) Maximum percentage of packets lost



(c) Mean number of consecutive packets received

Figure 4.16: Router Queue Type

Of the queue management types used in the configurations, CoDel and Droptail caused the fewest game packets to be dropped. CoDel and Droptail perform very similarly, likely due to the fact that most packets would not stay in the queue for very long and would, therefore, not be dropped by the CoDel algorithm (instead reverting to Droptail behavior). RED and Adaptive RED also perform similarly, but RED drops a higher proportion of the game-traffic than AdaptiveRED, likely due to adaptiveRED adjusting the drop probabilities to be less aggressive than the RED configuration.

Changing the Queue Type does not really have an impact on the game-traffic. If network operators wanted to ensure low bandwidth UDP stream did not drop packets, they would have to increase total network capacity to ensure that queues would not become full.

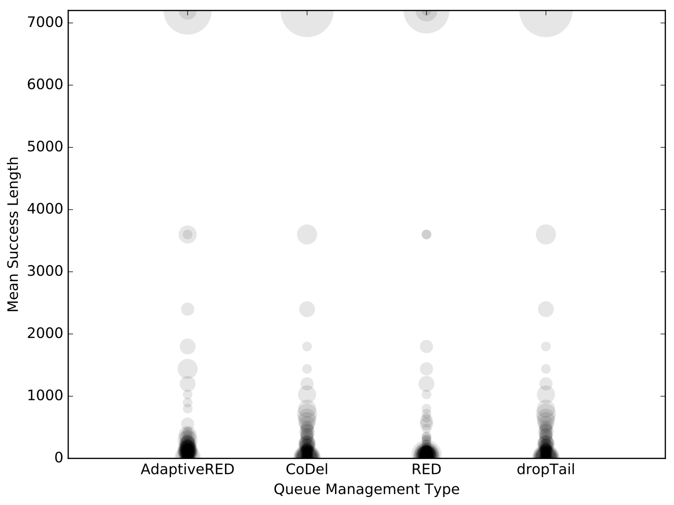
4.3 Results of Proposed Mitigation Technique on Simulated Data

As discussed in Section 3.5 an analytical approach was taken to determine if bundling previous packets with each game update would help mitigate the problem of packet loss. In the following sections, the mitigation technique is applied analytically to the simulation results and analyzed. With only four packets retransmitted, approximately 90% of runs experienced little or no data loss for a full two minutes.

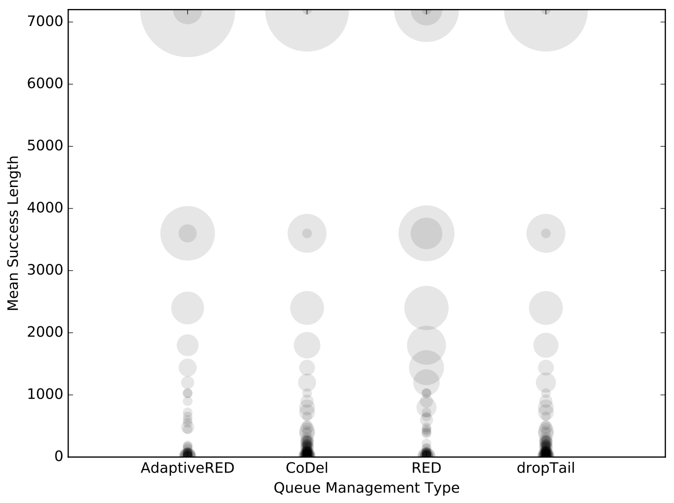
Results

Figure 4.17 compares the results of the analytical application of the mitigation technique on the results of the Queue Type analysis from Section 4.2.2, using the Mean Success Length metric to demonstrate how long the game would run without data loss. Comparing Figure 4.17b (where the previous packet data is bundled to each outgoing packet) with Figure 4.17a (from Section 4.2.2) shows that even a single packet retransmitted has a noticeable impact on the playability of the game. In all cases, each of the configuration groups resulted in less packet loss - increasing the length of time the game could play without data loss. The largest increase appears to be at the top of the chart where the bubbles(s) represent runs that had no data lost (7200 consecutive packets received) or runs where one or two packets were lost at the beginning of the run or the end (eg: 7199). The inclusion of the previous packet with each outgoing packet also reduces the number of runs at, or around, zero for the Mean Success Length and increases the number of runs that are playable for half of the run (3600) and a third of the run (2400).

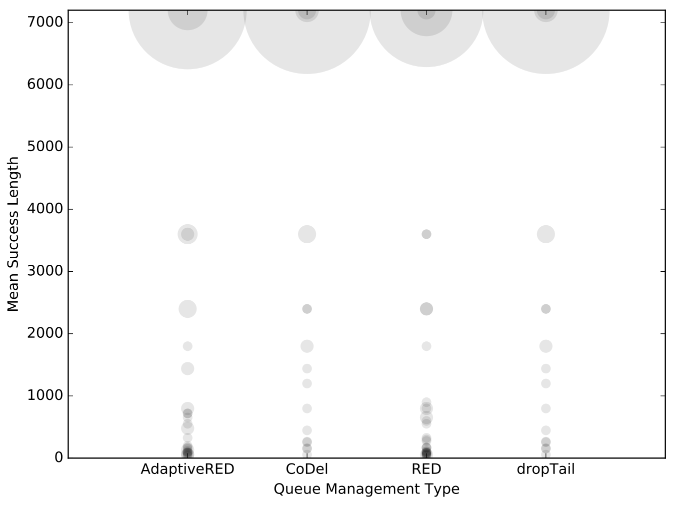
Without the mitigation technique, the number of configurations with no data loss was around 14%. However, with the inclusion of the previous packet, the number of runs with no (or little) data loss increased to 28% for RED, 38% for DropTail/CoDel and 53% for AdaptiveRED. Contrary to the earlier results, it appears that AdaptiveRED works best with this type of mitigation strategy. This also confirms that the configurations using AdaptiveRED resulted in intervals where only a single packet is dropped, followed by one or more successfully recieved packets. This is supported by Figure 4.16b where most of the runs have a maximum percentage of packets lost at or below 50% and Figure 4.16a where most runs had one or fewer



(a) No retransmission



(b) Retransmitting one packet



(c) Retransmitting four packets

Figure 4.17: Results of mitigation: QueueType

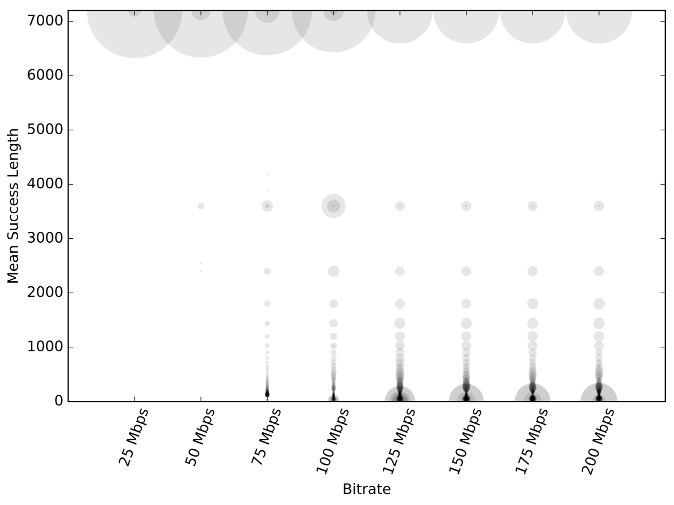
packets lost for any given interval. Even under bad network conditions, the mitigation strategy would increase the probability of a good gameplay experience.

Increasing the number of packets retransmitted to four (the four previous packet payloads are retransmitted with each new packet) further increases the length of time the game is playable. However, depending on the game, the increased latency due to having four lost packets may be noticeable, even if the data is eventually received. In Figure 4.17c, the number of runs with no data loss, or minimal data loss, has increased again, this time to 93% for DropTail/CoDel, 86% for RED and 84% for AdaptiveRED. CoDel and DropTail are performing better, with only a few configurations experiencing any packet loss. This may be unexpected, but returning to Figure 4.16a, CoDel and DropTail had the most configurations with four or fewer packets lost. AdaptiveRED and RED both have configurations with more than 15 packets lost. It is interesting to note that, even though there are four previous packets payloads with each outgoing packet, all configuration groups have at least one configuration experiencing enough packet loss to drop the playable game time to mere seconds. However, this mitigation technique shows promise and has significantly increased the number of runs with minimal data loss.

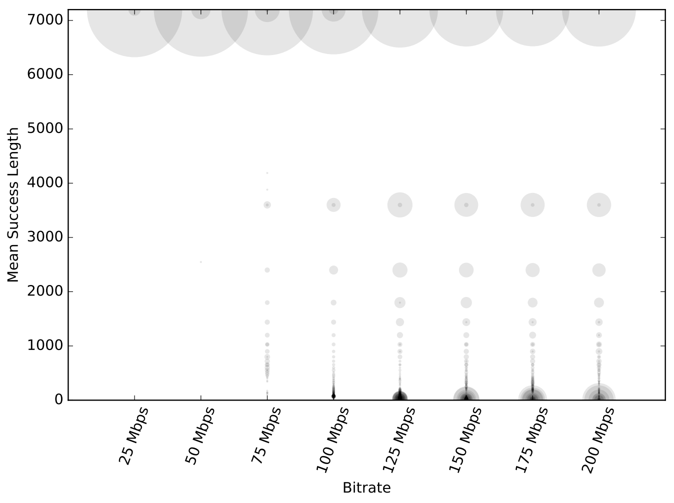
Examining the impact of the mitigation technique with respect to Bitrate (Figure 4.18), the results are less clear. There is an increase in the number of runs with no data loss, but at high bitrates there are still a large number of runs experiencing data loss. Figure 4.18b shows the distribution of Mean Success Length for the Bitrate configuration groups utilizing a one packet retransmit mitigation strategy. For most configurations, there is an increase in the runs experiencing no data loss. Looking at the raw data, the highest increase is at 125 Mbps (with a 34% increase). The remaining configurations (over 125 Mbps) experienced a 25% increase in this metric. Around 60% of configurations experienced no data loss for the configuration groups over 100 Mbps while 90% of the configurations at or below 100 Mbps had no packet loss. There is an increase in the number of runs with a Mean Success Length of 3600 and 2400 for the configurations over 100 Mbps and a general reduction in the other values. The 25 Mbps, 75 Mbps, and 100 Mbps see a reduction in all values less than 7199. As expected, an under-saturated network appears to function well, allowing most of the game packets to travel freely through the router. With the inclusion of an extra packet being retransmitted, the amount of data loss is further reduced. Even at higher bitrates, the mitigation strategy reduces data loss; a majority of configurations are playable for the entire game period.

Figure 4.18c is the distribution of the Mean Success Length when retransmitting four packets, grouped by Bitrate. Once again, there is a noticeable decrease in the number of configurations with a Mean Success Length less than 7199. Despite the increase in the number of configurations around 7200, there are still a number of configurations clustered below 1000 - indicating these configurations, on average, would allow the game to run smoothly for approximately 17 seconds. Additionally, there still exist configurations at 75 Mbps and 100 Mbps that exhibit data loss.

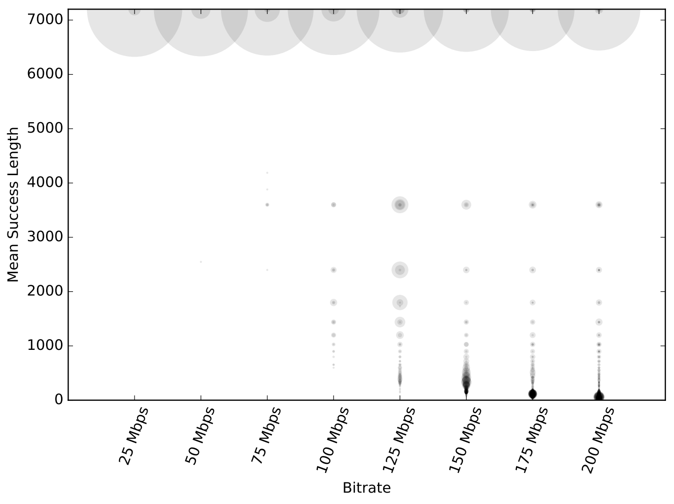
There is an increase in the number of runs slightly less than 7200, indicating that there were more than four packets lost at the end of the run, or that more than four packets were lost at the very beginning of the



(a) No retransmission



(b) Retransmitting one packet



(c) Retransmitting four packets

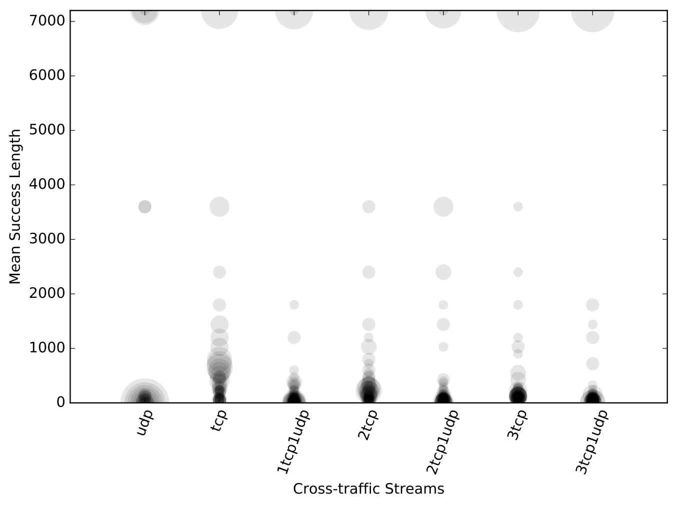
Figure 4.18: Results of mitigation: cross-traffic Bitrate

run. Looking again at the raw data for bitrates over 100 Mbps, the number of configurations experiencing little or no data loss increases to between 75% and 85%, with the data loss increasing with the bitrate. For configurations at or below 100 Mbps, the number of configurations experiencing little or no data loss was between 98% and 100% (again, the data loss increased with the bitrate). The 50 Mbps configuration group does experience data loss with runs at 3600 and around 2400, but on this chart those small numbers are hard to see. Once again, the mitigation strategy appears to be useful; however, higher bitrates or high congestion networks still present a problem for applications sensitive to data loss.

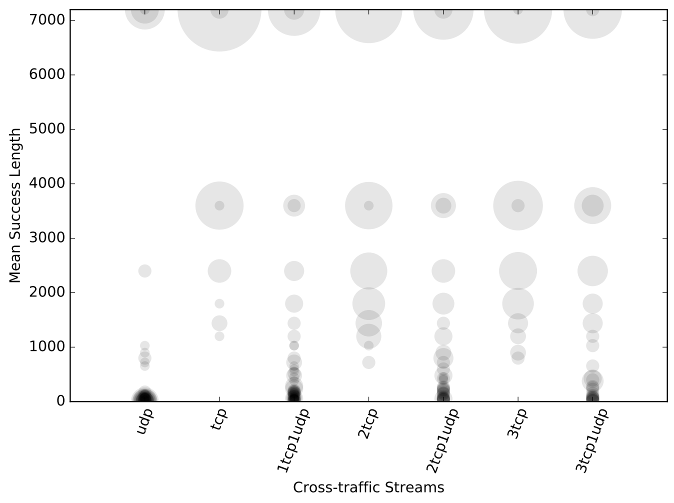
Figure 4.19 compares the distributions of the Mean Success Length for the cross-traffic Streams with and without the mitigation strategy. Figure 4.19b demonstrates the effects of retransmitting a single packet. When Figure 4.19b is compared to Figure 4.19a, there is an increase in the Mean Success Length for every configuration group. Of all the configurations groups, the TCP-only configuration groups have the lowest data loss. The configurations with two and three TCP streams also have a reduction in data loss but not to the same extent as the single TCP configuration group. All TCP-only configurations still have runs with the Mean Success Length less than 50% of the total run time. However, the single TCP-only configuration has the least number of runs with a Mean Success Length less than 50% of the total run time. Any configuration group with a UDP stream did not respond as much to the mitigation strategy of retransmitting a single packet, with only a slight increase in the number of runs with little or no data loss. The UDP stream configuration groups still have many configurations that were only able to run for small numbers of seconds at a time before experiencing data loss.

Figure 4.19c shows the results of retransmitting four packets. In this scenario, the TCP-only configurations now have almost all perfect runs, losing data only at the beginning or ends of the runs. The configurations with a UDP stream still experience data loss; as the amount of bandwidth allocated to TCP streams increases, less data is lost. The configurations with a UDP stream have more packets lost at the beginning or end of the run resulting in several overlapping bubbles near 7200. The UDP-only configuration group still has 25% of the configurations only able to run for at most 3.6 seconds (215 packets), and nearly 50% of the runs have a Mean Success Length less than or equal to 3600 packets (60 second without losing data). However, in Figure 4.15a, most configurations have fewer than 10 packets lost at any given time, and most packets lost were in groups of four or less, suggesting that an increase in the mitigation strategy could produce better results. Since a large portion of Internet traffic is TCP, these results suggest that a mitigation of as little as four packets retransmitted may be sufficient to handle most cases of packet loss, assuming that the game can tolerate a delay in game state of as much as four frames. As UDP traffic increases, the mitigation strategy would also need to be made more aggressive by increasing the number of packets retransmitted.

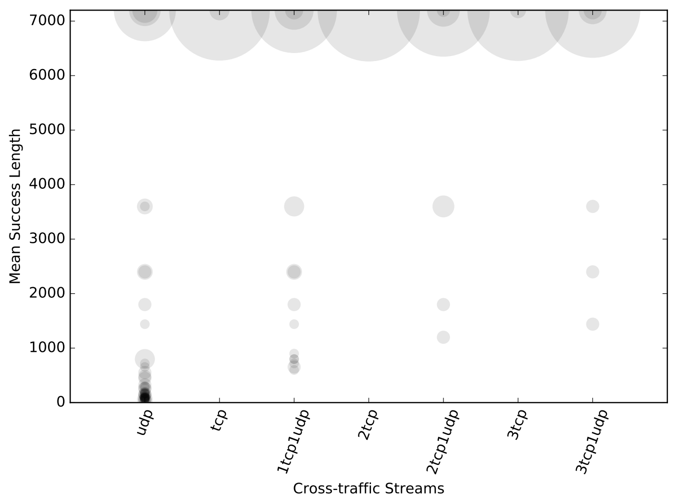
Figure 4.20 visualizes the results of the mitigation strategy on the Classifier Type analysis. Figure 4.20a shows the Mean Success Length for the Classifier Type without any mitigation strategy applied. Figure 4.20b shows that a simple mitigation strategy of retransmitting a single packet of data has a noticeable impact on the Mean Success Length, increasing the number of runs that have little or no data loss as compared



(a) No retransmission



(b) Retransmitting one packet



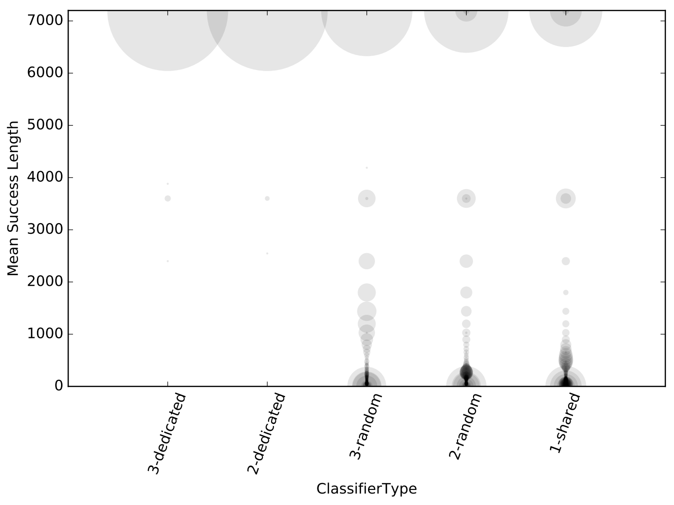
(c) Retransmitting four packets

Figure 4.19: Results of mitigation: cross-traffic Type

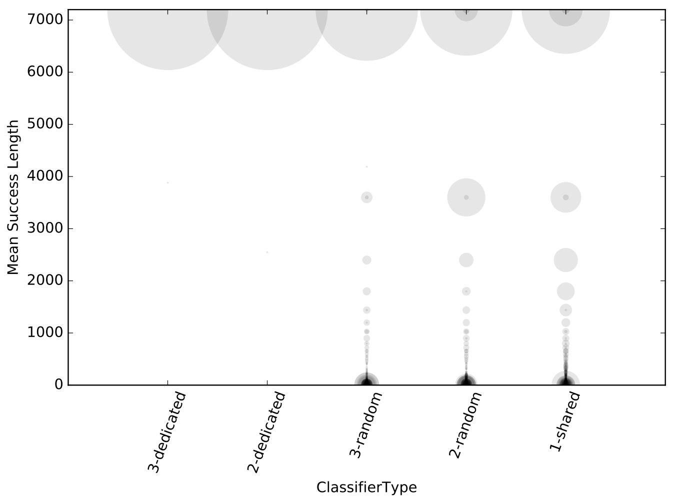
to Figure 4.20a where no mitigation strategy was applied. Much like Figure 4.20a, where an increase in the number of queues appears to have a decrease in the data lost, Figure 4.20b follows a similar pattern. However, across the non-dedicated configurations, 27% of all runs are unable to sustain a stable game for more than 30 seconds (1800 Mean Success Length). Additionally, contrary to the data loss trend, the two-queue configuration has an increase in runs that were able to run for one minute (3600 Mean Success Length) without any data loss. Another observation is that the increase in the number of queues reduced the number of runs with a Mean Success Length just below 7200, indicating that with fewer queues, the game loses more packets either at the beginning of the run, or at the end of the run. However, the mitigation strategy does not appear to have a noticeable impact on values around 7200.

If the mitigation retransmission rate is increased to four, there is an increase in the number of runs that have little or no data loss. At this point, each of the non-dedicated configurations appear to have a similar distribution for the Mean Success Length. There is also a slight increase in the number of configurations just below 7200, once again indicating that there are groups of lost packets at the beginning of runs and at the end of runs. A closer look at the raw data does indicate that the previous trend still exists, where an increase in queues correlates to a decrease in the number of almost perfect runs (with little or no data loss). Additionally, an increase in queues sees an increase in configurations at nearly every Mean Success Length value lower than 3600 (one minute of data-loss free runtime). This supports the results in Section 4.1.4, where an increase in the number of queues seems to increase the data loss. Despite these results, even at four packets retransmitted, no non-dedicated packet classifier configuration group ever achieved 90% near perfect runs and the difference between the non-dedicated results was less than 10%. This result indicates that the classifier type has little useful influence on the packet loss and that the mitigation strategy works similarly across any realistic configuration group.

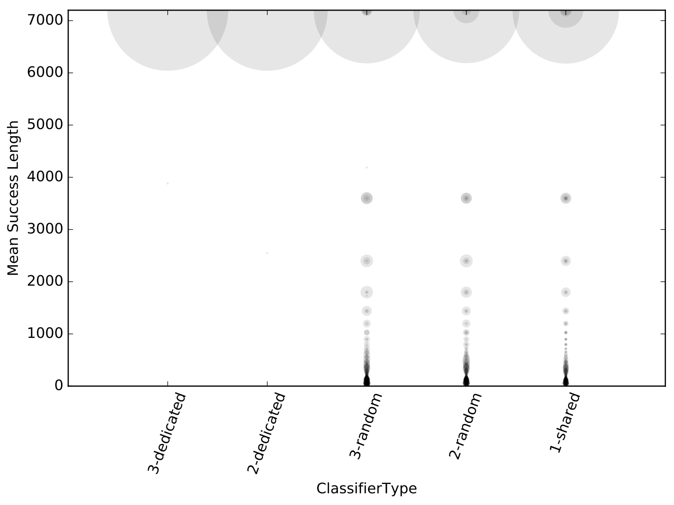
The Packet Size Distribution (Figure 4.21) results followed a similar pattern to what was seen in Section 4.1.2. Figure 4.21 depicts the distribution of the Mean Success Length for the Packet Size Distribution using different mitigation strategies. The x-axis shows each configuration group with labels centered on their respective configuration group. With only one packet being retransmitted, the 128-byte configurations continue to stand out. The 128-byte configurations became more pronounced, resulting in the 128-byte configurations having fewer runs near 7200 and increasing the number of runs with a Mean Success Length at or below 700 ($\approx$11 seconds). Contrary to Figure 4.4, where 128-byte configurations groups had an increased number of configurations with a Mean Success Length less than 70 (1.2 seconds), the single packet mitigation strategy decreased the number of configurations with a Mean Success Length less than 70 (for the 128-byte configurations) at a higher rate than the other configuration groups resulting in the 128-byte packet configurations having fewer configurations with a Mean Success Length under 70. With the single packet mitigation strategy, the 128-byte configurations still stand out with an increase in runs with a Mean Success Length below 3600 and a slight decrease in runs with a Mean Success Length at 3600. Once the mitigation strategy was increased to four packets, the distinction between the configuration groups becomes less pronounced as



(a) No retransmission



(b) Retransmitting one packet



(c) Retransmitting four packets

Figure 4.20: Results of mitigation on the Classifier Type Analysis

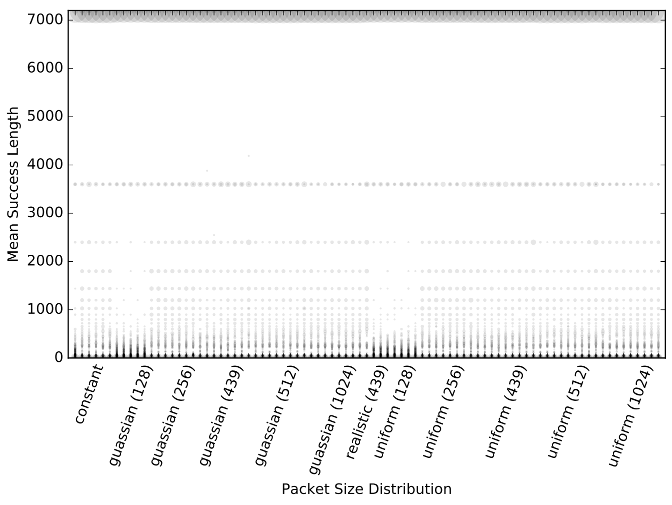
the results converge to similar distributions with no obvious trends in their differences.

The remaining configuration groups differed very little from each other, reinforcing the results from Section 4.1.2. The results of the mitigation strategy seem to indicate that many of the 128-byte packet configurations had runs with many single-packet losses that caused the initial analysis to calculate a lower Mean Success Length but did not alter the Maximum Percent Loss and Maximum Packets Lost by a noticeable amount. The mitigation strategy then eliminated those single packet losses, redistributing the Mean Success Length to higher values. However, by increasing the number of packets retransmitted, the configurations appear to converge - indicating that all configuration groups had a similar Maximum Packets Lost distribution, as confirmed by Figure 4.2, Figure 4.3, and Figure 4.4.

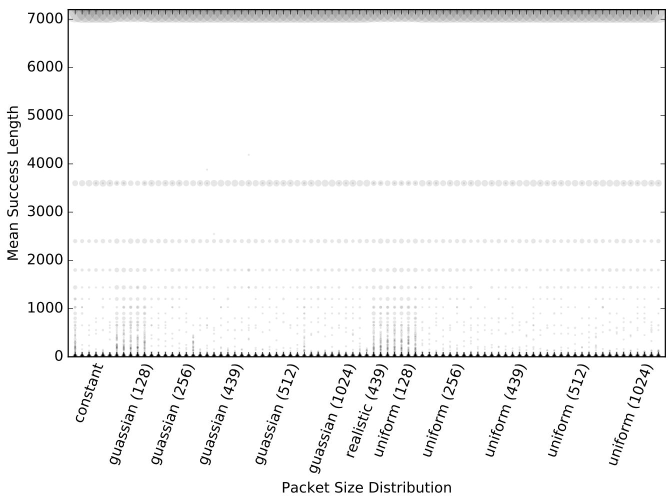
The remaining configuration, Inter-packet Time Distribution (Figure 4.22), provided very little insight into the mitigation strategy. Much like the results of Section 4.1.3, each of the Inter-packet Time Distribution configuration groups responded very similarly to the mitigation strategy. With each increase in the number of retransmitted packets, the number of near-perfect runs increased by a similar margin, (from 71% near perfect runs with no retransmission, to 79% near-perfect with the single packet retransmission, to 89% near perfect runs with four packets retransmitted). The only exception was the Realistic configuration, which had a slightly lower number of near-perfect runs (60% near-perfect runs with no retransmission, 73% near-perfect runs with a single retransmission and 86% near-perfect runs with four packets retransmitted). These results suggest that the Inter-packet Time Distribution has very little influence on the game packet loss and the mitigation strategy. However, this may also indicate that the synthetic distributions are extremely unrealistic. Since the synthetic distributions perform better, having fewer packets lost than the realistic distribution, this may suggest possible improvements to packet transmission algorithms such as transmission of packets at fixed intervals. In aggregate, with many traffic streams on the network, the synthetic nature may be lost, losing any advantage from more synthetic packet generation. Further analysis of Inter-packet Time Distribution has been left to future work.

4.4 Real World Observations

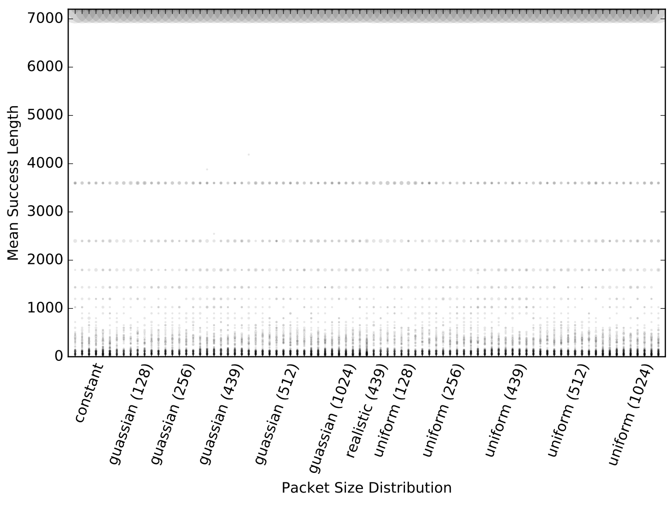
Part of the data collection included capturing game-like packets sent over a real network from a variety of different sources (as described in detail in Section 3.3). This experiment involved sending packets from multiple devices on several different networks to a central server that then recorded the packets received and lost in a similar format as to what was used in previous experiments in this chapter. From the analysis, the results were unlike the results from the simulations. LTE and ADSL proved to have the most unstable networks and experienced high packet loss. WiFi appeared to have a significant impact on the packet loss, but further investigation has been left for future work. The full results and analysis are below.



(a) No retransmission

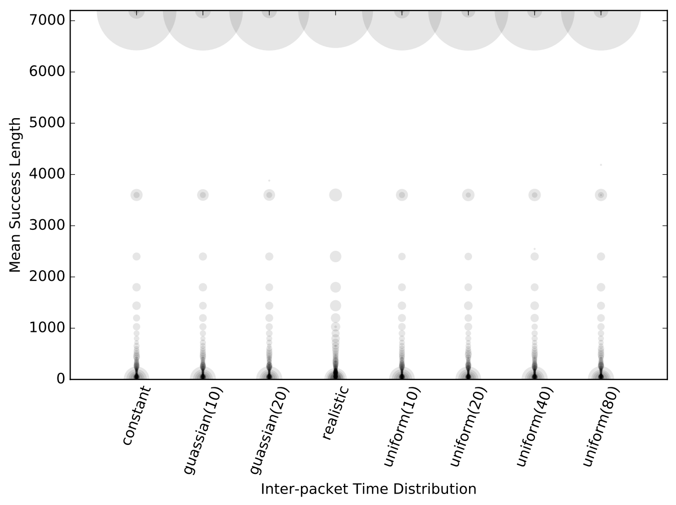


(b) Retransmitting one packet

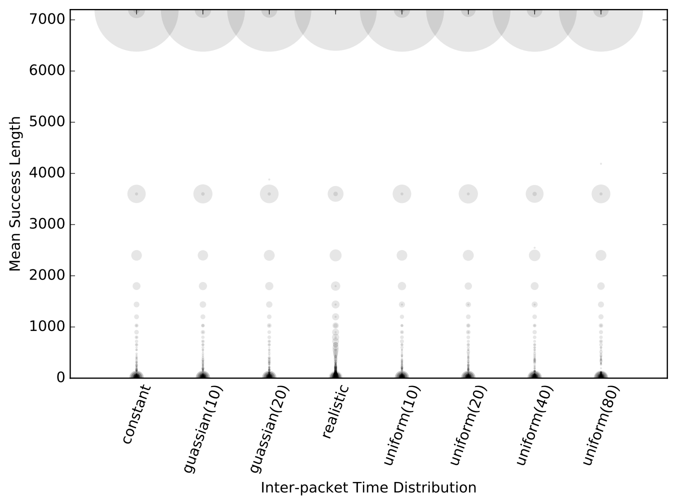


(c) Retransmitting four packets

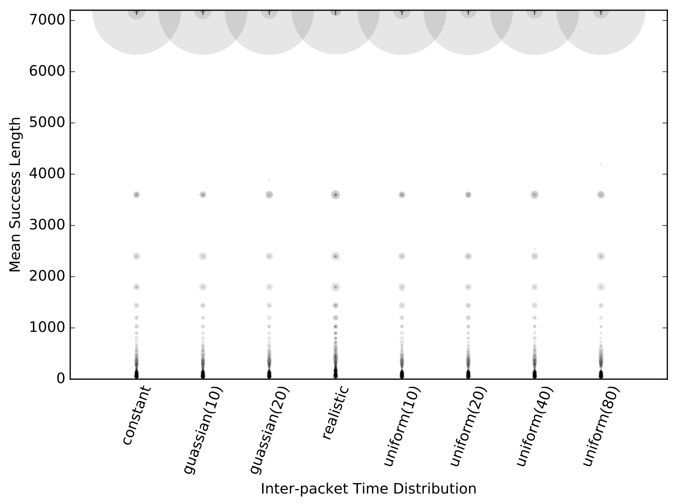
Figure 4.21: Results of mitigation on the Packet Size Distribution Analysis



(a) No retransmission



(b) Retransmitting one packet



(c) Retransmitting four packets

Figure 4.22: Results of mitigation on the Inter-packet Time Distribution Analysis

Results

As the network conditions were unknown during the experiment runs, configurations are grouped by source host and ISP as in Table 4.2. Using the same analysis as in previous sections, the results for Maximum Packets Lost, Maximum Percent Loss, and Mean Success Length were compiled and visualized in Figure 4.23 and Figure 4.24.

Table 4.2: Real World Device Configurations

Device	A	B	C	D	E	F
OS	WIN 10	WIN 10	CentOS 6	WIN 10	WIN 10	Android
Network Type	WiFi	Wired	Wired	Wired	WiFi	LTE
ISP	Business Park	University	Business Park	ADSL/Cable	ADSL/Cable	Telco

In Figure 4.23, the ADSL experiments exhibited packet loss of up to 2690 consecutive packets lost in a run. Outliers exist at 6442 consecutive packets lost for one of the University runs, 4832 consecutive packets lost for one of the Wired Cable runs, 6551 for one of of the Wired Cable runs, and 3842 for one of the LTE runs. These outlying results were left in the quantitative analysis. Even though these outliers are impossible to deal with at their magnitude, they do happen, albeit rarely. This does indicate that, in some rare instances, game-play will be disrupted by extreme packet loss that no mitigation strategy will ever be able to hide. In keeping with previous analysis in this chapter, a truncated chart (Figure 4.24a) with the Maximum Packets Lost capped at 30, will be examined further for detail at the low end of packet loss. The apparent differences between Figure 4.23 and Figure 4.24a are due to the differences in the y-axis scale between the two charts. Figure 4.23 has a larger y-axis scale, and therefore the vertical distance between the bubbles are reduced making the two charts appear to represent different datasets. This is most noticeable with the cable scenarios (D, E), where, in Figure 4.23, the bubbles for no packets lost and one consecutive packets lost are more separated and distinct than the corresponding bubbles in Figure 4.24a (where it appears the two bubbles are closer in size).

In Figure 4.24a, most runs had run length intervals with fewer than five consecutive losses. The devices on ADSL (D, E) and the device on LTE (F) experienced the most packet loss while Cable devices (D, E) experienced the least packet loss. For these scenarios, any run with more than four consecutive packets lost is a relatively rare event, occurring less than 2% of the time. The ADSL devices (D, E) had 8% of the runs with more than four consecutive packets lost and the LTE device (F) had 7% of the runs with more than four consecutive packets lost. The distribution of the Maximum Packet Loss for the LTE network appears significantly different than the other distributions. At first glance the LTE distribution appears to follow the half-normal distribution, but with additional samples the distribution may become more well-defined. Despite device A (Windows 10 Wifi) and device C (CentOS 6 Wired) operating over a business level network, they seem to experience a large number of consecutive packet losses in the same run. This may be explained by the traffic-shaping rules applied to the office router or business gateway. Further investigation has been left for future work.

In the office scenario (A, C), WiFi seems to have a minimal effect on the outcome, as either the router or ISP appears to have the most influence on the rate of packet loss. The cable ISP (D, E) frequently drops small groups of packets, while ADSL (D, E) occasionally drops large groups of packets. The effect of WiFi is minimal on ADSL and Cable, although the distributions vary slightly, this may just be due to the highly variable nature of the network traffic. The LTE results indicate that packets are frequently lost, typically in groups between one and seven, but occasionally up to 16 packets are lost at a time. The LTE loss may correspond to load on the network, but that analysis has also been left for future work.

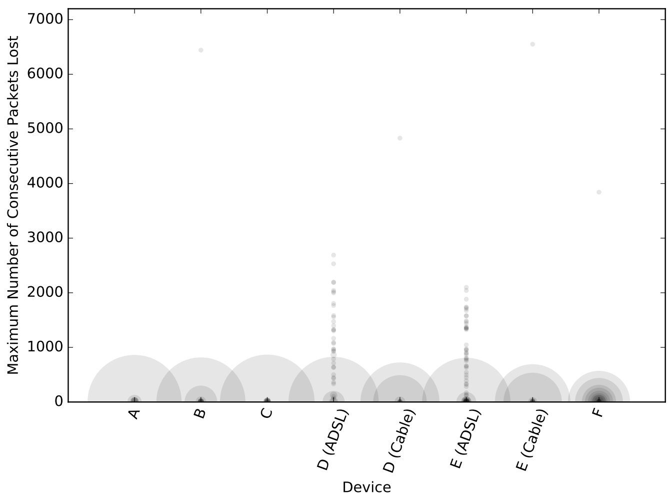
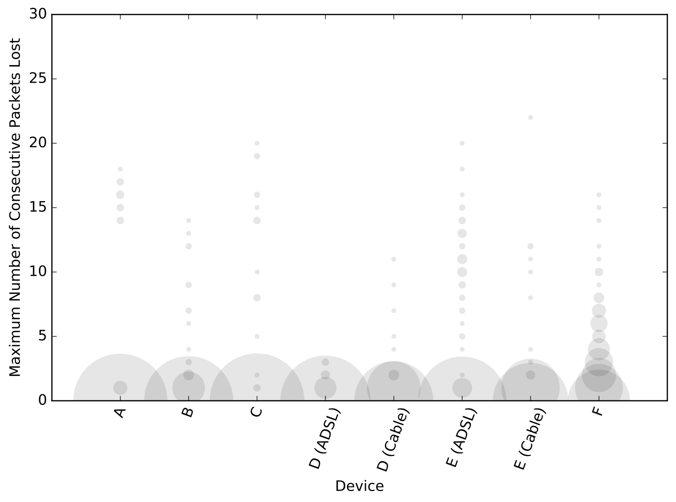
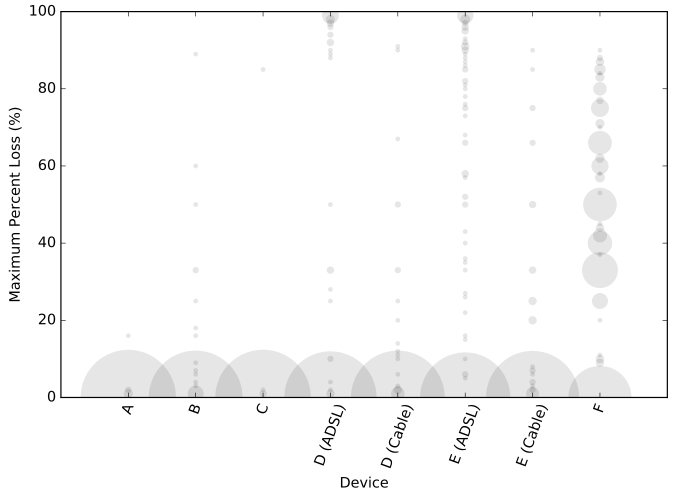


Figure 4.23: Maximum number of consecutive packets lost (Full Chart)

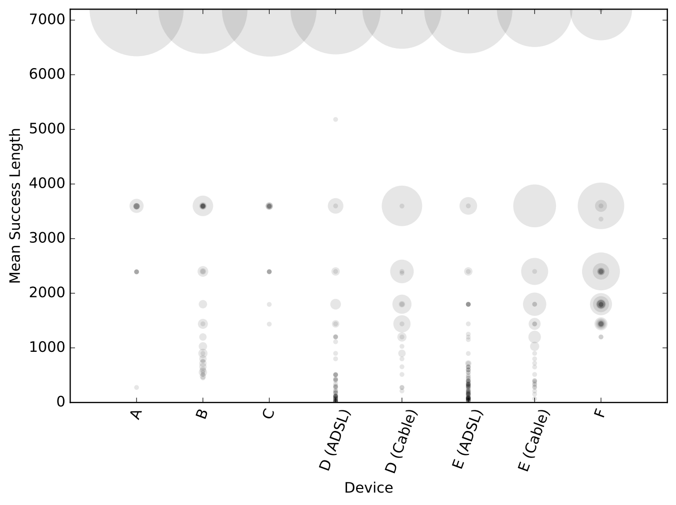
The Maximum Percent Loss paints a slightly clearer picture of the packet loss. In Figure 4.24b, the ADSL devices (D, E) lose a higher percentage of packets than any of the other configurations. The 100% packet loss for the ADSL indicating packets were lost at the beginning of the sample run or one of the instances where ADSL dropped more than 200 packets for every one packet received. The remaining high percentages for ADSL likely indicate sequences where the packet received to lost ratio was between 1:9 (90% packet loss) and 1:25 (96% packet loss) or some multiple of these ratios. The Office network (A, C) experienced the least percentage of packet loss with most run length intervals having below 2% packet loss with outliers at 16% and 85% indicating that even though there were more instances of high packet loss (Figure 4.24a), these losses were accompanied by more received packets thereby reducing the Maximum Percent Loss. The Office network outliers at 85% are one of the rare instances where more packets were lost in an interval, than received. The Cable (D, E) and University (B) networks exhibited similar results with most of the runs having a Maximum Percent Loss less than 2% with a few runs with up to 91% packet loss in an interval. The WiFi Cable (E) experienced twice the number of instances with over 2% than the Wired Cable, but followed a similar distribution. Scenario E (LTE) clearly indicates that the LTE network was the most volatile network



(a) Maximum number of consecutive packets lost



(b) Maximum percentage of packets lost



(c) Mean number of consecutive packets received

Figure 4.24: Real World Results

– with many runs experiencing intervals with over 50% packet loss and large concentrations between 30% and 70%. Nevertheless, the LTE network still had over 80% of its runs with a Maximum Percent Loss less than 50%.

The next metric to look at is the Mean Success Length, corresponding to an approximate Mean Time Between Failure. If the system can operate for a sufficient duration between unrecoverable packet loss events, then the network performance is acceptable. However, if data is lost while the game is running and the game is unable to compensate for the lost data, the user experience will be negatively impacted.

In Figure 4.24c, the vast majority of the datasets collected did not lose a single packet. Smaller bubbles occur around the 3600 packet run-length (half of the experiment run-time), likely indicating a single packet loss event during the run, and smaller bubbles leading all the way down to almost zero indicate progressively worse network conditions. Table 4.3 groups the results of Figure 4.24c into three categories: Near Perfect (Percentage of runs from that host that had little or no packet loss), Around Half (Percentage of runs from that host that were around the 3600 mark), and Less than a Quarter (Percentage of runs for that host that had on average 2400 or fewer consecutive packets received). The corporate network (A, C) is, in fact, the most stable, resulting in over 96% of the runs with little to no packet loss and long periods of successful runs. The runner up, unexpectedly, was the Wired ADSL (D) with over 88% of the runs having little to no packet loss followed by the University network (B) with over 86%, WiFi ADSL (E) with 85%, the Wired Cable (D) with 68%, WiFi Cable (E) with 62%, LTE (F) with only 41% of the runs having little or no packet loss. Device B (University Network), and Devices D and E on ADSL experienced a small number of captures that resulted in short mean time between failures (indicating a very unstable network). Even though the ADSL has over 84% near perfect runs, they also had some of the highest percentage of runs that were unable to run for a quarter of the time, only having better results than the Cable (D, E) and LTE (F) networks. Conversely, the ADSL had some of the lowest percentages of runs that had Mean Success Length around 3600, only the Office (A, C) network having fewer results in that category. Overall, the Cable (D, E) and LTE (F) proved to be the most unstable with the highest number of runs resulting in either a smooth runtime of 50%, or runs that were unable to run smoothly for much more than 30s.

The LTE network has no collected run with a Mean Success Length less than 1200; most of the average successful run-lengths were for the entire duration or around 1/4, 1/3, or 1/2 of the total run length (1800, 2400, and 3600 respectively). This was not expected. However, a close look at the run length data suggests that many LTE connections experience a rough start, which would account for reduced MTBF values. If the application waited for the network state to stabilize before capturing packets, the MTBF would likely increase further. This means that mobile games connected over LTE could implement a warm up period during loading screens or cut-scenes where packets are exchanged, so that the network has a chance to get used to the new traffic, allowing the game play to run more consistently.

WiFi appears to have had an impact on packet loss. In Table 4.4 (a table containing the number of runs that had a given Maximum Packet Loss), device A (WiFi on Business Park) has 50% more runs with packet

Table 4.3: Real World Data Mean Success Length Breakdown

	Near Perfect	Around Half	Less than a Quarter
A (Office WiFi)	96.1%	3.14%	0.714%
B (University)	86.1%	6.00%	7.86%
C (Office)	97.4%	1.71%	0.857%
D (ADSL)	88.4%	2.57%	9.00%
D (Cable)	67.6%	17.3%	15.1%
E (ADSL WiFi)	84.6%	3.14%	12.3%
E (Cable WiFi)	61.6%	19.3%	19.1%
F (LTE)	41.6%	24.4%	34.0%

Table 4.4: Real World Data Maximum Packet Loss

Packets Lost / Host	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	>30
A (Office WiFi)	673	13													3	3	4	3	1													
B (University)	603	77	7	2	1		1	2		2			2	1	1																	1
C (Office)	682	3	1			1			3		1				1	1	2			2	1											2
D (ADSL)	619	34	5	3																												39
D (Cable)	473	214	7		1	1		1		1		1																				1
E (ADSL WiFi)	592	27	1		1	2	1	2	2	3	6	6	2	5	3	2	1		1		1											42
E (Cable WiFi)	431	255	5	1	1				1		1	1	2										1									1
F (LTE)	291	170	85	57	34	12	20	13	7	1	4	1	1		1	1	1															1

loss than device C (Wired on Business Park). Packet loss in both cases is below 4%, and both of the scenarios only had approximately 2% of the runs with greater than four consecutive packets lost. The ADSL WiFi (E) device had 33% more runs with packet loss than the Wired ADSL device (D). In this instance, the ADSL WiFi device (E) had twice the number of runs with more than four consecutive packets lost than the wired device on ADSL (D). The WiFi on Cable (E) experienced an 18% increase in runs with packet loss over the Wired device on Cable (D). However, the Cable network (D, E) has less than 2% of the runs with four or more consecutive packets lost.

The Cable devices (D, E) experienced a greater number of packet losses and had the greatest increase from wired to WiFi. Since the Cable devices used the same internal network as the ADSL devices (D, E), the differences between the wired and Wifi may be due to interactions between the WiFi network and the relatively unstable Cable network. Investigating these interactions has been left for future work.

The difference in WiFi vs. Wired is more noticeable looking at the Maximum Percent Loss (Figure 4.24b) where the WiFi networks experience a more varied distribution of packet loss in the case of ADSL and simply an increase in runs with packet loss in the case of Cable.

The Mean Success Length also supports the conclusion that WiFi has increased packet loss with an increase in non-perfect runs and higher concentrations of runs with low Mean Success Length. The similarities between the WiFi and Wired results on the Business Network indicates there are other factors influencing the results. Perhaps the Business Park had reduced radio interference or a better access point (as compared to the ADSL and Cable networks), reducing the number of packets lost due to the wireless medium, or the difference in packet loss may just be a result of the unpredictable nature of Internet traffic. However, the consistently higher packet loss on WiFi networks may indicate that the WiFi networks are slightly more unstable. More experiments are necessary to draw such a conclusion with any confidence and this has been left for future work.

The results of the real world data appear to be very dissimilar to the results of the simulated traffic from previous sections. Not only does each ISP exhibit different packet loss characteristics, but none of the exhibited packet loss distributions appear to match any of the charts from previous sections. Additionally, the overall packet loss appears to be lower in the real-world data as compared to the simulated data, indicating that either the network is unsaturated, or there are traffic shaping algorithms in place to prevent the loss of game-traffic packets. However, the erratic nature of LTE and the unexpected bursts of packet loss in the ADSL do indicate there is a need to improve performance in real world networks and that there exists a role for in-app mitigation strategies.

4.4.1 Mitigation Analysis on Real World Data

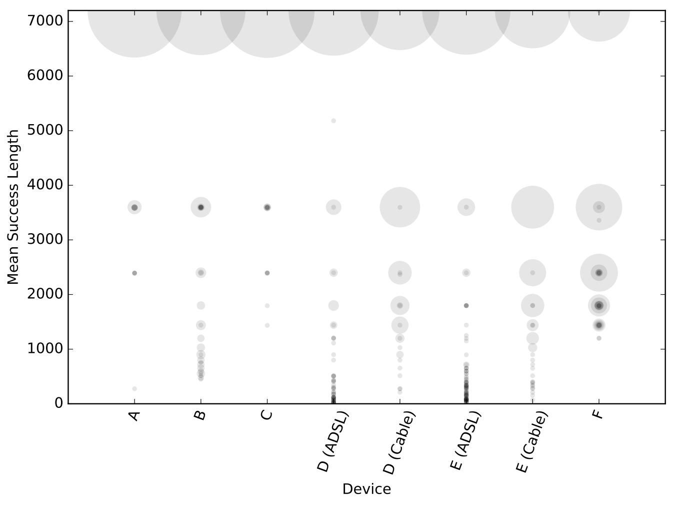
Similar to the analysis done for the simulation data, an analytical approach to the mitigation strategy was applied to the collected real world data. In this analysis, the run lengths were recalculated to simulate repackaging the payloads of previous packets with each new packet sent. The results indicate that approximately 90% of the runs would experience little or no data loss with four packets retransmitted. However, the ADSL and LTE networks still experienced significant packet loss that the mitigation strategy could not overcome. The full analysis and results are in the following section.

Results

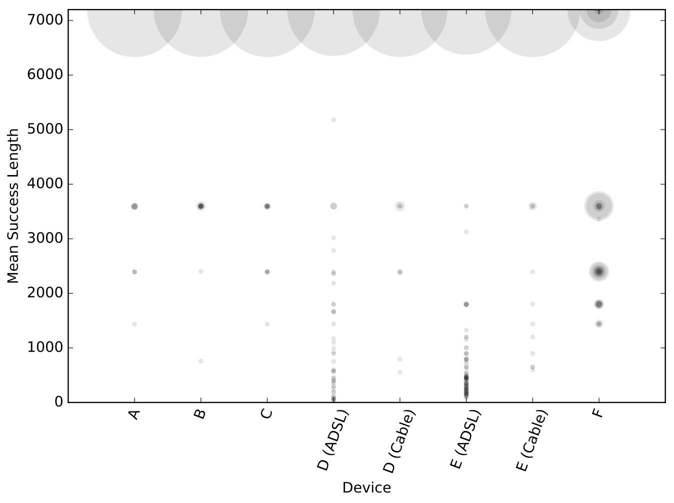
To match the presentation from Section 4.3, the results of one payload concatenation and four payload concatenation on the Mean Success Length are compared to the non-mitigated data in Figure 4.25b. From Figure 4.25b, the data loss on the University, Office, and Cable networks is all but eliminated. These three networks maintained a Mean Success Length for at least half of the run time of two minutes with only a few runs at 1/3 the total runtime, 1/5 the total runtime, and 1/8 the total runtime.

The ADSL appears mostly unchanged with only a minor reduction in the number of runs at each value at or below 3600 (1 minute) and a minor increase in the number of runs with no data loss. This is not surprising given the number of runs with high packet loss observed in Figure 4.23.

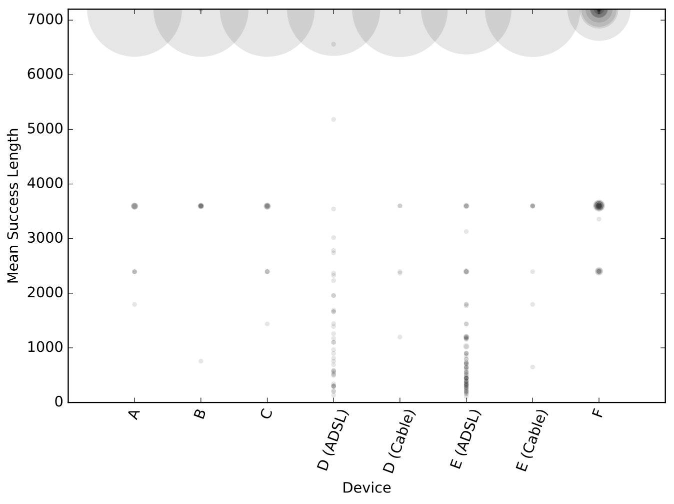
The LTE results further support the classification of the network as unstable by redistributing data points to nearby values. For example, if the Mean Success Length without mitigation was 3600, with one packet retransmitted the Mean Success Length became 3601, or a value near 2400 became near 3600, or something



(a) No retransmission



(b) Retransmitting one payload



(c) Retransmitting four payloads

Figure 4.25: Results of mitigation: Real World Measurement

similar. These near values caused overlapping circles to appear near 3600, 2400, and 1800. The redistribution in this manner suggests that the run length intervals in these runs were of varying sizes and with differing distributions of received/lost packets. While the number of runs with no packet loss did not seem to increase by much for LTE, the number of runs with almost no data lost increased – causing overlapping circles near 7200 to appear in the chart.

Increasing the number of payloads retransmitted did very little to data loss for most scenarios. The most significant changes in data loss visible in Figure 4.25c were from the LTE network that again increased the number of runs with almost no data loss indicating that many runs had packet loss at the beginning, supporting previous observations. The remaining Devices/ISPs had very little change (with a slight reduction of data loss in each), supporting the initial observations from Figure 4.23 and Figure 4.24a that all of the devices/ISPs experienced runs with more than four consecutive packets lost.

Despite how different the real world traffic is to the simulated traffic, the mitigation strategies, even at the lowest level, appear to have a significant impact on the loss of data in a game or similar application. Even in unstable networks like ADSL and LTE, the mitigation strategy works well enough that a game would be playable for a full two minutes over 90% of the time.

4.5 Discussion

From Section 4.1 and Section 4.4, packet loss is random and hard to characterize. There are however, some takeaways from the experiments that could help inform future network or application design. From Section 4.1.2, we learn that cross-traffic consisting of small packets appear to have a significant detrimental impact on game-traffic, potentially leading to network or application policies that would reduce small packet transmissions. Section 4.1.3 demonstrated that synthetic distributions for the inter-packet time performed better than the more realistic bursty transmission distribution. This insight into the inter-packet time could lead to more organized transmission times, reducing the impact of bursty traffic. Section 4.2.1 indicated that UDP cross-traffic also has a negative impact on game-traffic. Further research into the impact of UDP cross-traffic could lead to mechanisms to control UDP more effectively or may lead to developers choosing to use TCP instead. From the real world results in Section 4.4, we can see that some networks perform better than others indicating that network design, traffic shaping policies, network medium, and total available bandwidth can potentially have a large impact on game-traffic. However, without further investigation it is difficult to come to any concrete conclusions.

Looking at the maximum number of packets lost in these experiments, it becomes clear that it may not be possible to come up with a simple mitigation scheme that would work 100% of the time. It may be possible, however, to devise a scheme that is "good enough" most of the time. However, mitigation techniques greatly depend on the type of game that employs them. For example, a game that periodically sends out the complete world state to all clients can probably afford to lose the occasional packet, because the world state would be retrieved at a later time and the user would only experience a slight disruption in the user experience. However, games that send incremental non idempotent updates or events would be particularly impacted by a lost packet because game clients would become out of sync and the game would become unplayable.

Section 4.3 explores the mitigation technique of packaging multiple packet payloads together per update and compares the mean time between failures to the results without the mitigation technique and determines that retransmitting previous packets may be sufficient to mitigate most packet loss. Section 4.4.1 explores the same mitigation strategy with real world data and comes to the same conclusion, that the mitigation strategy will work for most network situations.

Provided that the game can survive a delay in packets equal to a small multiple of the inter-packet time, the mitigation technique explored would reduce the experienced effect of packet loss. In reliable transmission protocols that employ ACKs or NAKs, delays due to packet loss are already at least on the scale of round trip time. For small numbers of packets lost, retransmission would result in a more consistent game experience and reduced latency and jitter for games that employ TCP or similar reliable protocols.

4.6 Summary

In this chapter, the data collected is evaluated in an attempt to identify patterns that may emerge. It is shown that the cross-traffic protocols, packet size, and the overall bitrate have the largest impact on the game-traffic. Real world data was also collected as a means to verify the simulation and validate the mitigation techniques. The simulated traffic appears to have more packet loss than the real world data and the distributions of the two datasets appear very dissimilar. The mitigation technique of retransmitting past packet payloads with new updates appears to work much better with real world data than with the simulated data.

5 Conclusions And Future Work

5.1 Summary

This thesis explored the impact of cross-traffic on low-bandwidth game packets passing through a common network. A simulation environment was constructed to allow manipulation of the cross-traffic parameters and record detailed logs of the game-traffic. For comparison, a simple real world experiment was also conducted. The results were then encoded using run length encoding and statistics of the game-traffic characteristics were visualized using bubble charts. Additionally, a mitigation strategy was explored and applied to both the simulation results and the real world results.

In Section 4.1.1, the interaction between cross-traffic bitrate and game-traffic packet loss was explored via simulation. The results from the initial batch of simulations were grouped into equal-sized categories for each of the experimental cross-traffic bitrates and then analyzed. Each of the simulations was configured to generate cross-traffic packets with a specific mean packet size from a configured random distribution. The simulations were grouped based on the configured cross-traffic packet size and distribution to examine the effect of these parameters on the game-traffic. Further classification analyzed the impact of the mean and distribution of the cross-traffic inter-packet time (the time between packet transmissions).

The next experimental simulations focused on the impact of the number of queues in the router, and how the traffic streams are classified. Some of the configurations classified the packets such that the game-traffic was isolated from the cross-traffic, while other configurations directed cross-traffic packets into the same queue as the game-traffic. In either instance, the simulations were run with between one and three queues.

The second batch of simulations grouped the experiments by the protocol(s) of the cross-traffic. The protocol groupings included between zero and three TCP streams, with or without a UDP stream, in an effort to determine if the protocol of the streams or the number of streams in the cross-traffic impacted the game-traffic. The queue management protocol formed the final grouping of experiments: RED, AdaptiveRED, CoDel, and DropTail.

A retransmission mitigation technique (with up to four packets retransmitted) was analytically applied to the simulation data and then analyzed. Under most circumstances, the mitigation strategy eliminated data loss with only a minor increase in the latency. However, the analysis of the simulation data revealed that packet loss appeared to be bursty. In some instances, this bursty packet loss exceeded what a simple retransmission of packets would be able to mitigate. The analysis of the mitigation strategy also revealed that most packet losses occurred in runs shorter than five packets. The simulation parameters appeared to have minimal impact resulting in similar Mean Success Length distributions with a retransmission of four packets.

Real world data was captured by simulating game-traffic over a number of real networks. These networks included ADSL, Cable, Business Park, University, and LTE networks with both WiFi and wired infrastructure where available.

The mitigation technique (with up to four packets retransmitted) was analytically applied to the real world data and then analyzed. The real world data exhibited a similar pattern to the simulated data where an increase in the number of packets retransmitted resulted in similar distributions. However, the ADSL results had a noticeably different distribution with a larger number of runs that were unusable. Other than the ADSL runs, the mitigation strategy worked well, providing a stable network for over 90% of the captured runs.

The simulation series showed that cross-traffic characteristics could have an impact on the game-traffic. However, applying the lessons learned to real world applications may prove difficult as most of the lessons learned would require large systemic changes to network infrastructure. Both the real world and the simulated results reacted well to the mitigation strategy in all but the most extreme packet loss scenarios, leading to possible solutions that could be applied in real world games or applications.

5.2 Contributions

The simulations showed that, as the cross-traffic bitrate increases, more game-traffic packets are lost. This suggests a link between cross-traffic bitrate and game-traffic packet loss. However, even with cross-traffic bitrates below the destination network saturation point (100 Mbps), game-traffic packets are lost. Other factors in the network contribute to packet loss and it is not just congestion that results in game-traffic packet loss.

One of the factors that had the most impact on the game-traffic packet loss was the packet size of the cross-traffic data. Each of the simulations were configured to generate cross-traffic packets from a configured random distribution with a specific mean packet size. Small cross-traffic packets (128 bytes in the simulations) resulted in increased game-traffic packet loss but a decrease in packet loss interval size. Overall, the small cross-traffic packets resulted in reduced playability (the average length of time a game could run without experiencing a packet loss) when compared to the other configurations. Outside of the anomaly with the smaller cross-traffic packets a more expected pattern appears: an increase in the cross-traffic packet size resulting in fewer cross-traffic packets queued reduced the maximum percentage of game-traffic packets lost per run. Other than the small packet size configurations, the packet size (and distribution) did not have a significant impact on the playability of a game over the network. A Two Sample Kolmogorov-Smirnov statistical analysis of the results confirmed that the game-traffic packet loss of the 128-byte configurations differ from over 80% of the remaining configurations; most configurations did not result in statistically different packet loss. This would suggest that reducing the number of small packets in a network, possibly by bundling smaller packets, may significantly reduce the number of game packets lost.

The configurations with synthetic distributions (Constant, Guassian, Uniform) for cross-traffic had lower packet loss than the configurations with the realistic inter-packet distribution (Poisson Pareto Burst Process [1]), resulting in reduced playability. The configurations with synthetic distributions did not seem to differ greatly from each other. A statistical analysis of the packet loss using the Two Sample Kolmogorov-Smirnov test confirmed that the game-traffic packet loss from configurations with realistic inter-packet timing was significantly different from the packet loss experienced in the other configurations but that the synthetic cross-traffic inter-packet time distributions were not significantly different from each other. One possible explanation for the statistical difference is the bursty nature of the Poisson Pareto Burst Process of the realistic inter-packet timing distribution. A more aggressive traffic shaping algorithm is a potential solution that could smooth bursty traffic, improving overall network performance.

Some of the simulation configurations classified the packets such that the game-traffic was isolated from the cross-traffic, greatly reducing the game-traffic packet losses resulting in a stable, but unrealistic, network for gameplay. In more realistic configurations, with game-traffic and cross-traffic sharing queues, there appears to be a direct relationship between the number of queues and the playability of a game on the network. The realistic configurations also displayed an inverse relationship between the number of queues and achieving 0% packet loss, indicating that distributing the cross-traffic over more queues reduced the packet loss of the game-traffic and *generally* made the network more stable for playing games. An increase in the number of queues will likely increase the latency of the game-traffic which may be more detrimental for some games. The traffic shaping algorithms used can have a significant impact on mobile games; it may be possible to identify these low bandwidth game-traffic streams and classify them differently to reduce packet loss.

The protocol type of the cross-traffic substantially affects game-traffic packet loss. The configurations with TCP cross-traffic generally reduce the number of game-traffic packets lost. However, if there wasn't a UDP stream included in the cross-traffic, increasing the number of TCP streams appeared to increase the number of game-traffic packets lost. For the configurations with TCP, and after equilibrium had been reached, the general playability of games on the network does not appear to differ much between the different TCP configurations. The bitrates of the UDP streams in the simulations were reduced with the increase in TCP streams (to maintain a constant data-rate) which would not be a general pattern found in practice. Overall, it appears that UDP traffic, present in the cross-traffic, has a detrimental impact on the game-traffic. This would suggest that developers and network operators should strive to limit the amount of raw UDP in favor of TCP or technologies that implement congestion control.

In simulations that analyze the game queue, the CoDel and DropTail configurations perform identically, likely because packets are never in the queue long enough to trigger the CoDel algorithm, effectively turning the CoDel queues into DropTail queues. The RED variants performed similar to each other, with the AdaptiveRED simulations performing slightly better than RED, likely due to the dynamic configuration.

Overall, the CoDel and DropTail configurations resulted in fewer packets lost and longer run times than either of the RED queues, suggesting that a more passive queue management mechanism works in favor of the game-traffic. However, a passive queue management system may have adverse affects on other traffic so more testing is required. These results do suggest that the queue management has an impact on game-traffic packet loss and may need to be tuned for the desired result.

The mitigation strategy, when applied to the simulated data in Section 4.3, would eliminate data loss with only minor additional delays in data transmission time. However, some simulations, such as those configured with high bitrate or UDP cross-traffic, still experienced significant data loss. Under normal conditions the mitigation strategy with only four packets retransmitted should be sufficient to prevent data loss with only an additional delay of up to 67 ms (per the simulation configuration). With an increase in packet loss, the number of prior packets bundled with each packet could be increased further to reduce data loss. The impact of increased game packet size on the packet loss has been left for future work.

As part of Section 4.4, real world data was captured by simulating game-traffic over a number of real networks. The real world data indicated that the different networks exhibit very different results, results that varied greatly from the simulated data. As expected from the real world data, the Business Park and University topped the list with the most stable networks and longest uninterrupted game play times. The ADSL connections also performed well, with almost as many games playing uninterrupted as the Business Park and University; however, most of the remaining ADSL runs were virtually unplayable. Even the University network experienced some unplayable games but to a lesser extent than the ADSL runs. The Cable and LTE networks were playable about half the time, but experienced a lot of packet loss with very many small runs of received packets. In each case, the resulting charts from the analysis were unique and showed very little similarity to simulated results. These observations indicate that there are many different factors, some of which were not explored, that can impact game-traffic. In some instances where a wired and WiFi option were available, a direct comparison between the two technologies can be made. WiFi, in general, has a detrimental impact on game playability, decreasing the number of perfect runs and increasing the number of packets lost. An unanswered question for future work is whether the additional packet loss from WiFi is simply due to the WiFi protocol, the wireless medium, or an additional network device in the path between source and destination.

The mitigation strategy, when applied to the real world data in Section 4.4.1, produced results indicating that a game could run for two minutes, without any data loss, over 90% of the time, with only four previous packets being bundled. However, every network still experienced data losses; under most circumstances the resulting data loss still resulted in a playable game. The ADSL and LTE networks experienced more packet loss than the other networks, resulting in more games that were completely unplayable or only playable for a shorter period. These results suggest that the proposed mitigation strategy may work well, most of the time, in real world games.

5.3 Future Work

While investigating the impact of cross-traffic packet size on the game-traffic packet loss in Section 4.1.2, the 128-byte packet configurations exhibited behavior that differentiated them from the remaining configurations. Speculation as to why these configurations differed includes load on the router and interactions with the CDMA network. In real world traffic, many applications including mobile games have small packets (on the order of 128-bytes) indicating that this problem may exist in practice. If the cause of the detrimental impact of 128-bytes could be determined, and is confirmed to exist in the real world, the results could add guidance to network or application design.

Outside of the investigation of the 128-byte configuration, there were some minor visual differences that appeared to exist in the charts such as the difference in packet loss between packet sizes. These patterns may have been due to the randomness of the simulations or may indicate unexplored relationships between the configurations. Further investigation through repetition and multi variable analysis would help determine if packet size is more important than originally determined.

During the analysis of the classification of traffic in Section 4.1.4, the question of latency, due to the number of queues and amount of cross-traffic, was touched upon. Since some applications are sensitive to changes in latency (jitter), an investigation into the impact of cross-traffic on the game-traffic latency could lead to important improvements in both application and network design.

In Section 4.1.4, only round robin schedulers and simplistic classifiers were used. In real world applications, routers typically employ much more sophisticated routing and traffic shaping algorithms. As part of future investigations into the impact of routers on game-traffic, more realistic router designs would yield more useful results.

As part of Section 4.2.1, where the protocol of the cross-traffic streams were analyzed, there were questions about the impact of mixed UDP/TCP in real world scenarios. In the simulation experiments, the overall bitrate of generated packets was a constant; however, for TCP traffic streams the number of transmitted packets would be reduced due to congestion control. Additionally, the UDP streams were configured to transmit at a constant rate despite the TCP streams reducing the actual transmitted bitrate; therefore, the simulated bitrates were inaccurate whenever there was a TCP stream in the simulation. In real world scenarios, the actual bitrates are dependent on the application and network congestion. More realistic and controlled experiments with TCP, UDP, and Reliable UDP as cross-traffic, such as using real world traffic captures, may provide insights into game-traffic packet loss that may yield improvements to the protocols, network design, or application designs in the future.

In Section 4.3, realistic inter-packet time configurations performed significantly worse than the synthetic configurations. This indicates that the inter-packet time of cross-traffic has an impact on the game-traffic packet loss. However, it is currently unknown whether these observations could be applied to real world data transmission. Further experimentation with synthetic distributions may lead to new traffic shaping algorithms that can be applied to real world applications.

The results from Section 4.4 indicated that the office network performed the best, yet still experienced bursts of packet loss. The LTE network experienced the most packet loss and variability of all the networks tested. The WiFi networks experienced more packet loss than Wired; however, the difference between WiFi and Wired differed greatly between networks. There is some speculation as to these results being due to the medium, congestion, or traffic shaping rules. Further investigation with a more controlled network, where the cross-traffic can be inspected and compared to game-traffic, may yield results allowing for a better understanding of packet loss in real world scenarios.

References

[1] Doreid Ammar, Thomas Begin, and Thomas Guerin-Lassous. A new tool for generating realistic Internet traffic in NS-3. In *ICST Conference on Simulation Tools and Techniques*, pages 81–83, Pisa, Italy, March 2011.

[2] David P. Anderson. BOINC: A system for public-resource computing and storage. In *IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, PA, November 2004.

[3] Grenville Armitage. *At the Intersection of Networks and Highly Interactive Online Games*, pages 403–434. Springer, 2010.

[4] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *Computer Communication Review*, 2014.

[5] Anastasiia Beznosyk, Peter Quax, Karin Coninx, and Wim Lamotte. Influence of Network Delay and Jitter on Cooperation in Multiplayer Games. pages 351–354, Hong Kong, China, December 2011.

[6] Tom Bova and Ted Krivoruchka. Reliable UDP protocol, 1999. URL `https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00`.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Giuliana Carullo, Marco Tambasco, Mario Di Mauro, and Maurizio Longo. A performance evaluation of WebRTC over LTE. In *Annual Conference on Wireless On-demand Network Systems and Services*, pages 1–6, Cortina d'Ampezzo, Italy, January 2016.

[9] Kuan-Ta Chen, Polly Huang, and Chin-Laung Lei. How sensitive are online gamers to network quality? *Communications of the ACM*, 49(11):34–38, 2006.

[10] Min Chen. AMVSC: A framework of adaptive mobile video streaming in the cloud. In *IEEE Global Communications Conference*, pages 2042–2047, Anaheim, CA, December 2012.

[11] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. DeepRS: Deep-Learning Based Network-Adaptive FEC for Real-Time Video Communications. In *IEEE International Symposium on Circuits and Systems*, pages 1–5, Seville, Spain, 2020.

[12] Richard A. Comroe and Daniel J. Costello. ARQ schemes for data transmission in mobile radio systems. *IEEE Transactions on Vehicular Technology*, 33(3):88–97, 1984.

[13] Wu-chang Feng, Francis Chang, Wu-chi Feng, and Jonathan Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, 13(3):488–500, 2005.

[14] Wu-chang Chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin. Techniques for eliminating packet loss in congested TCP/IP networks. Technical Report UM CSE-TR-349-97, University of Michigan, 1997.

[15] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[16] Tobias Fritsch, Hartmut Ritter, and Jochen Schiller. The effect of latency and network limitations on MMORPGs (A field study of Everquest2). In *Workshop on Network and System Support for Games*, pages 1–9, Hawthorne, NY, October 2005.

[17] Chen Gao, Haifeng Shen, and M. Ali Babar. Concealing jitter in multi-player online games through predictive behaviour modeling. In *IEEE International Conference on Computer Supported Cooperative Work in Design*, pages 62–67, Nanchang, China, May 2016.

[18] Anna Giannakou, Dipankar Dwivedi, and Sean Peisert. A machine learning approach for packet loss prediction in science flows. *Future Generation Computer Systems*, 102:190–197, 2020.

[19] Max Goldman, Greg Little, and Robert C Miller. Real-time collaborative coding in a web IDE. In *ACM Symposium on User Interface Software and Technology*, pages 155–164, Santa Barbara, CA, October 2011.

[20] Yunhong Gu and Robert L. Grossman. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks*, 51(7):1777–1799, 2007.

[21] Richard W Hamming. Error detecting and error correcting codes. *Bell Labs Technical Journal*, 29(2): 147–160, 1950.

[22] Chad Hansen, Nigel Jurgens, Dwight Makaroff, David Callele, and Philip Dueck. Network performance measurement framework for real-time multiplayer mobile games. In *Workshop on Network and Systems Support for Games*, pages 1–2, Denver, CO, December 2013.

[23] Zhanwei Hou, Changyang She, Yonghui Li, and Branka Vucetic. Ultra-reliable and low-latency communications: Prediction and communication co-design. *IEEE International Conference on Communications*, pages 1–7, May 2019.

[24] Selim Ickin, Karel De Vogeleer, Markus Fiedler, and David Erman. The effects of packet delay variation on the perceptual quality of video. In *IEEE Conference on Local Computer Networks*, pages 663–668, Denver, CO, October 2010.

[25] IEEE. IEEE Standards for Local Area Networks: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *ANSI/IEEE Std 802.3-1985*, 1985.

[26] Roger Immich, Pedro Borges, Eduardo Cerqueira, and Marilia Curado. QoE-driven video delivery improvement using packet loss prediction. *International Journal of Parallel, Emergent and Distributed Systems*, 30(6):478–493, 2015.

[27] V. Jacobson. Congestion avoidance and control. In *International Conference on Communications Architectures and Protocols*, pages 314–329, Stanford, CA, August 1988.

[28] Rudolf E. Kalman and Richard Buey. A new approach to linear filtering and prediction theory. *Journal of Basic Engineering*, 83(Series D):95–108, 1961.

[29] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. A measurement study on achieving imperceptible latency in mobile cloud gaming. In *International Conference on Multimedia Systems*, pages 88–99, Taipei, Taiwan, June 2017.

[30] Partha Kanuparthy and Constantine Dovrolis. DiffProbe: Detecting ISP service discrimination. In *IEEE International Conference on Computer Communication*, pages 1–9, San Diego, CA, March 2010.

[31] Partha Kanuparthy and Constantine Dovrolis. ShaperProbe: End-to-end detection of ISP traffic shaping using active methods. In *ACM Conference on Internet Measurement*, pages 473–482, Berlin, Germany, November 2011.

[32] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[33] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[34] Dan Lake, Mic Bowman, and Huaiyu Liu. Distributed scene graph to enable thousands of interacting users in a virtual environment. In *Workshop on Network and Systems Support for Games*, pages 1–6, Taipei, Taiwan, November 2010.

[35] Emil Larsson. Movement prediction algorithms for high latency games. Bachelor's thesis, Blekinge Institute of Technology, Karlskrona, Sweden, 2016.

[36] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *ACM International Conference on Mobile Systems, Applications, and Services*, pages 151–165, Florence, Italy, May 2015.

[37] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[38] Bomin Mao, Zubair Md Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. A Tensor Based Deep Learning Technique for Intelligent Packet Routing. In *IEEE Conference on Global Communications*, pages 1–6, Singapore, December 2017.

[39] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *ACM International Conference on Embedded Networked Sensor Systems*, pages 39–49, Baltimore, MD, November 2004.

[40] Niwas Maskey, Seppo Horsmanheimo, and Lotta Tuomimaki. Latency analysis of LTE network for M2M applications. In *International Conference on Telecommunications*, Graz, Austria, July 2015.

[41] Laurent Massoulié and James W Roberts. Bandwidth sharing and admission control for elastic traffic. *Telecommunication Systems*, 15(1-2):185–201, 2000.

[42] Paul E McKenney. Stochastic fairness queueing. In *IEEE International Conference on Computer Communications*, pages 733–734, San Francisco, CA, June 1990.

[43] Arash Molavi Kakhki, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. Identifying traffic differentiation in mobile networks. In *ACM Conference on Internet Measurement*, pages 239–251, Tokyo, Japan, October 2015.

[44] John Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4): 435–438, 1987.

[45] Hung Xuan Nguyen and Patrick Thiran. The boolean solution to the congested IP link location problem: Theory and practice. In *IEEE International Conference on Computer Communications*, pages 2117–2125, Anchorage, AK, May 2007.

[46] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7): 42–50, July 2012.

[47] Carlos Oliveira, Jaime Bae Kim, and Tatsuya Suda. An adaptive bandwidth reservation scheme for high-speed multimedia wireless networks. *IEEE Journal on Selected Areas in Communications*, 16(6): 858–873, 1998.

[48] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3): 344–357, 1993.

[49] Caleb Phillips and Suresh Singh. CRAWDAD dataset pdx/vwave (v. 2009-07-04). Downloaded from `https://crawdad.org/pdx/vwave/20090704`, July 2009.

[50] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Workshop on Network and System Support for Games*, pages 152–156, Portland, OR, August 2004.

[51] Markus Rank, Zhuanghua Shi, Hermann J. Müller, and Sandra Hirche. Predictive communication quality control in haptic teleoperation with time delay and packet loss. *IEEE Transactions on Human-Machine Systems*, 46(4):581–592, August 2016.

[52] Michal Ries, Philipp Svoboda, and Markus Rupp. Empirical study of subjective quality for massive multiplayer games. In *International Conference on Systems, Signals and Image Processing*, pages 181–184, Bratislava, Slovakia, June 2008.

[53] George F. Riley and Thomas R. Henderson. The NS-3 network simulator. *Modeling and tools for network simulation*, pages 15–34, 2010.

[54] Kalpana Saha Roy and Tune Ghosh. Study of Packet Loss Prediction using Machine Learning. *International Journal of Mobile Communication & Networking*, 11(1):1–11, 2020.

[55] Jose Saldana, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete Navarro, and Luis Casadesus. The effect of router buffer size on subjective gaming quality estimators based on delay and jitter. In *IEEE Conference on Consumer Communications and Networking*, pages 482–486, Las Vegas, NV, January 2012.

[56] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The Effect of Latency on User Performance in Warcraft III. In *Workshop on Network and System Support for Games*, pages 3–14, Redwood City, CA, May 2003.

[57] Joel Sommers and Paul Barford. Cell vs. WiFi: on the performance of metro area mobile connections. In *Internet Measurement Conference*, pages 301–314, Boston, MA, November 2012.

[58] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. A geometric approach to improving active packet loss measurement. *IEEE/ACM Transactions on Networking*, 16(2):307–320, 2008.

[59] Lingfen Sun and E. C. Ifeachor. Prediction of perceived conversational speech quality and effects of playout buffer algorithms. In *IEEE International Conference on Communications*, volume 1, pages 1–6, Anchorage, AK, May 2003.

[60] P. Laith Suresh and Ruben Merz. NS-3-Click: Click Modular Router integration for NS-3. In *ICST Conference on Simulation Tools and Techniques*, pages 423–430, Pisa, Italy, March 2011.

[61] Andrew S. Tanenbaum and David Wetherall. *Computer Networks*. Pearson Prentice Hall, 2011.

[62] Doan Thanh Tran and Eunmi Choi. A reliable UDP for ubiquitous communication environments. In *International Conference on Computer Engineering and Applications*, pages 1–6, Gold Coast, Australia, January 2007.

[63] Arthur Valadares, Eugenia Gabrielova, and Cristina Videira Lopes. On designing and testing distributed virtual environments. *Concurrency and Computation: Practice and Experience*, pages 3291–3312, 2016.

[64] Prateek Verma, Alessandro Ilic Mezzay, Chris Chafe, and Cristina Rottondi. A Deep Learning Approach for Low-Latency Packet Loss Concealment of Audio Signals in Networked Music Performance Applications. In *Conference of Open Innovation Association*, pages 268–275, Trento, Italy, September 2020.

[65] Udi Weinsberg, Augustin Soule, and Laurent Massoulie. Inferring traffic shaping and policy parameters using end host measurements. In *IEEE International Conference on Computer Communications*, pages 151–155, Shanghai, China, April 2011.

[66] Takahiro Yasui, Yutaka Ishibashi, and Tomohito Ikedo. Influences of network latency and packet loss on consistency in networked racing games. In *Workshop on Network and System Support for Games*, pages 1–8, Hawthorne, NY, October 2005.

[67] Jihwang Yeo, David Kotz, and Tristan Henderson. CRAWDAD: A community resource for archiving wireless data at Dartmouth. *ACM SIGCOMM Computer Communication Review*, 36(2):21–22, April 2006.

[68] Zhiyong Zhang, Ovidiu Mara, and Katerina Argyraki. Network Neutrality Inference. In *ACM Conference on SIGCOMM*, pages 63–74, Chicago, IL, August 2014.