MCCA

A Communication Architecture for Online Multiplayer Games

by

Armin Bahramshahry

B.A.Sc., The University of British Columbia, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2009

Abstract

Over the last decade the ability of the Internet infrastructure to carry traffic has not improved at the same rate as the desktop technology. This imbalance has increased the perceived difference in the quality of service (QoS) offered by online multiplayer games compared to single player games.

This thesis introduces MCCA, a communication architecture for online multiplayer games to improve the observed QoS and to lower the development complexity. MCCA takes advantage of online game's relaxed state consistency and predictable workload. MCCA enables a game to label its traffic as belonging to different classes, each with different priorities and requirements. Such labelling, in turn, enables differentiated traffic management, efficient use of available network resources, and ultimately, improved perceived QoS. In addition, MCCA enables a game to adapt to network conditions, through distributed quality aggregation, for each of the game's generated network traffic. Consequently, MCCA supports a set of generic group communication and quality estimation techniques, and yet it enables a game to define customized methods.

This thesis presents the MCCA architecture and its simulation-based evaluation using Quake III, Voice-Over-IP (VoIP), and file transfers. Experiments demonstrate that workload classification, prioritization, and class targeted QoS improve user experience and lower the generated network traffic, while quality aggregation and reporting enable game adaptation to network conditions.

Table of Contents

Appendices

List of Tables

List of Figures

List of Programs

Acknowledgments

I had the privilege of receiving guidance and support from my supervisor, Dr. Matei Ripeanu. His insight, support, and enthusiasm are very much appreciated.

I would like to also thank Dr. Sathish Gopalakrishnan for his effort in reviewing this work and providing helpful comments and technical advice. I am also thankful to the Networked Systems research group students for their friendship, support, and insightful comments.

I am thankful to my friends and family, specially my parents Rashid Bahramshahry and Rohangiz Khosraviani, for their support, encouragement, and confidence in my abilities throughout my education. I am also thankful for having the support of my sister, Anahita, and brother, Abtin, at all times. I would like to foremost thank Arshia Mandegarian for her assistance, encouragement, and unconditional love.

Dedicated to my parents:

Rashid Bahramshahry

Rohangiz Khosraviani

Chapter 1

Introduction

The revenues of the computer game industry are two times larger than those of the movie industry [64] and have been growing faster over the past decade [25] (expecting to reach $48.9 billion worldwide in 2011 [61]). In recent years, online multiplayer games (i.e., games where several players interact simultaneously over networks like the Internet) have become increasingly popular as multiplayer games such as World of Warcraft (WoW), Halo, Quake, and Counter-Strike have attracted millions of players [8]. These games provide an online virtual playground that offers players the tools to interact with one another. This social online environment has attracted more players to online games as competing with human counterparts is typically more interesting and challenging than playing against a computer.

The interesting game-plays, growing demand, and popularity of online games have attracted a large number of game development companies. Most importantly, online multiplayer games provide new sources of revenue such as virtual item sales, subscription tiers, advertising, auctions, player trades, expansion packs, and event or tournament fees for developers and publishers. These new opportunities introduce a heterogeneous workload that, in addition to critical game–generated events, include advertisement banners, Voice-Over-IP (VoIP) and text messages, player statistics, and other types of data that are transferred among players and servers.

Online multiplayer games traditionally have strict network communication requirements in order to run a smooth game. In addition, game development companies tend to support wide range of players around the world, with different network infrastructure, to maximize their profit. Challenges for supporting the required game features, revenue streams, and their generated network traffic that uses such a diverse network environment, motivates the study of a communication infrastructure that efficiently utilizes the network resources, and enables multiplayer games to improve the user perceived QoS, and furthermore enables games to adapt to real-time network conditions.

1.1 The Problem

Most online multiplayer games support a wide range of players, irrespective of their network capacity, as a result online multiplayer games are designed to saturate the narrowest last–mile links [21]. In fact, online multiplayer games developed for PS3 and XBOX 360 are mandated to support the range of players specified by Microsoft and Sony. As a result, player's upload capacity has been a key scaling limitation for online multiplayer games [4], as each player needs to frequently send critical game updates to every other player for the game to run smoothly. As a game's scale increases, the upload capacity of each player remains constant, yet the requirement to distribute game events to all participants makes the upload traffic grow linearly with the number of players.

The other major problem of network-based multiplayer games is caused by the network transmission delay. This means that it takes a while until information (e.g., about the movement or action of the opponents' objects) reaches the receivers. This delay causes several difficulties and may lead to paradoxical situations.

Due to scaling limitations caused by low network capacities, game designers have

been exploring game– or genre–specific solutions to decrease the amount of generated traffic using group communications and interest management techniques [3, 23, 54], or to reduce the volume of generated traffic using interest management techniques [4, 5, 24, 28, 30, 36, 55], or by decreasing the game update rate by using various prediction and multi-resolution simulation techniques [27, 29, 41, 42]. Such improvements are only possible due to a game's state and behavior predictability, game's ability to manage its event generation, and games' pre-hand knowledge about its generated workload and requirements.

The strict online multiplayer game's characteristics and network requirements have motivated this research. This thesis focuses on challenges resulting from a combination of bandwidth limitations in game's deployment platform, and soft real-time constraints for the game traffic. To address these challenges, we explore optimization alternatives that exploit traffic differentiation to: 1) reduce the volume of generated traffic, and 2) independently fine-tune the QoS delivered for each service class.

1.2 Approach

Online multiplayer games' workload consists of multiple types of data (e.g., game events, advertisements, VoIP traffic, etc), each with different requirements and inherently different importance for a good game-play experience. The key to deal with such workload is to separate the traffic into multiple classes and treat each class independently. This thesis presents Multi-Class Communication Architecture (MCCA), a communication architecture for online multiplayer games that takes advantage of games' heterogeneous workload characteristics to provide communication layer optimizations that improve the players' online game experience. The following techniques allow MCCA to better

utilize network resources, manage network traffic, and serve user demand to improve the observed QoS:

1. Traffic prioritization
2. Differentiated message loss, ordering, and transfer policies for different classes of traffic
3. Targeted message forwarding policies, leading to message distribution overlays, optimized for the requirements of each flow of network traffic
4. Flow–specific distributed quality estimation, aggregation, and reporting to enable game adaptation to network conditions

1.2.1 Traffic Prioritization

The demand for communicating a large and diverse workload can easily downgrade the observed game quality (e.g., VoIP or file transfer traffic can interfere with critical game events/updates). To least degrade a player's experience when network resources are constrained, and to favor the more important network traffic, MCCA enables a game to prioritize its network traffics. In order to do so, the game is required to label its messages as belonging to different classes with different priorities.

1.2.2 Differentiated Transfer Policies

Online multiplayer games support a number of non-core functionalities such as text chats, voice chats, live advertisements, custom player looks and more. Each of these functionalities has its own behavior and data transfer requirements. For example, game

events require the transfer of a large number of small packets at fixed time intervals [18, 20–22, 38, 39], while VoIP traffic has different packet size and frequency [46, 52]. Additionally, players tolerate the delay and loss of game events differently than those for VoIP packets. To provide targeted QoS, MCCA allows the game to define the characteristics of each class of data, and assign each of its network traffic flows (such as game updates, VoIP, or text messages, and referred to as a "flow" in this thesis), to a class of network traffic. Satisfying each class's requirements in isolation from other classes, as oppose to treating all data transfers similarly, to satisfy every class's requirements, not only reduces the overall generated network traffic but can also improve the observed QoS.

1.2.3 Targeted Message Forwarding Policies

Network traffics with tight latency requirements prevent the use of efficient message broadcast techniques, such as multicast trees, for other network traffics. In turn, network traffic classification enables a game to employ a multicast tree for broadcasting messages of classes of traffic with relaxed latency constraints. This message dissemination approach lowers the total volume of the generated network traffic, and migrates some of the network traffic from players with low network capacity to others. Traffic classification and the insight to each class's latency requirements enable this optimization.

1.2.4 Distributed Quality Aggregation and Reporting

MCCA enables a game to adapt to network conditions by providing a set of mechanisms to measure and aggregate each flow's observed quality. Furthermore, MCCA reports the observed quality and other flow specific network transfer information to the game,

for the game to make adjustments (e.g., disable VoIP) that could preserve/improve the observed QoS, when facing changes in the network conditions.

1.3 Evaluation Methodology

To evaluate our assumptions, and to analyze MCCA we have developed a prototype implementation of the MCCA. In addition, we have developed a network simulator and traffic generators for game, VoIP, and file transfers. Through experiments with various configuration of network traffics, network deployment conditions, and network traffic class definitions, we demonstrate the flexibility and potential performance improvements of the proposed communication architecture for online multiplayer games.

We measure the correctness of the implemented network simulator by comparing it with a network emulator, running various workload and network deployment configurations. In addition, we compare MCCA with other online multiplayer game optimizations, as well as the optimal case.

To demonstrate the flexibility of MCCA, and the benefits of the proposed guidelines and mechanisms in MCCA, we define and implement game-level optimizations built on top of MCCA, and use MCCA's quality and network reports to adapt to various real-time conditions such as the player's network capacity, number of players, and the generated network traffic.

1.4 Contributions

The main contributions of this work is the introduction of new mechanisms that enable the use of game-level information on the characteristics of the generated traffic and requirements that drive players' quality of experience, to improve communication layer

efficiency. In addition, this research takes the first step toward providing a general purpose game communication layer taking advantage of game-level information, workload predictability, and flexibility.

More concretely, the work presented makes five main contributions with respect to problems facing online multiplayer games (as described in Section 1.1). The following is an overview of these contributions:

- Design and evaluation of MCCA, a communication architecture supporting game workload classification for providing targeted QoS (providing a combination of reliable/unreliable, in-order/out-of-order message delivery, as well as mechanisms for supporting low latency communication)

- Design and evaluation of an efficient group communication, evolving over time, for the game generated network traffic

- Design and evaluation of distributed quality estimation, aggregation, and reporting for online multiplayer games based on a flow of network traffic's communication statistics and more

- Design, implementation, and evaluation of a few game-level optimizations taking advantage of MCCA's design; demonstrating potential for new types of game-level optimizations using MCCA

- Validation of the following ideas based on experimental result using emulation of Quake III, VoIP, and file transfer traffic: 1) game workload classification, prioritization, and targeted QoS improve observed game and voice quality, 2) classification is even more beneficial at higher number and amount of network traffics, 3) application-level multicast reduces network capacity requirements and improves

overall quality, 4) distributed quality estimation, as a feedback mechanism, can enable online multiplayer games to adapt to network conditions

1.5 Overview

The remaining chapters of this thesis are laid out as follows: Chapter 2 presents the background on online multiplayer games as well as previous research on online games and efficient network communication. Chapter 3 presents our proposed communication architecture, MCCA, and compares it to other solutions. Chapter 4 discusses the methodology for evaluating MCCA, and presents some of the implementation details. Chapter 5 explains the set of experiments performed and discusses the results. Chapter 6 is the conclusion that summarizes the problem, the approach, and the contributions of this work, and it also presents a future direction for the continuation of this research.

Chapter 2

Background and Related Work

MCCA builds on past work on network QoS and application layer multicast. Our work differentiates from the above mainly due to the focus on online multiplayer games with tight latency requirements. We exploit the different characteristics and requirements for the categories of generated traffic, to efficiently use network resources and improve the perceived quality of gaming experience. Furthermore, we propose distributed quality aggregation and reporting (described in Section 3.3) at the communication layer, to enable game adaptation to network conditions. The combination of generic and yet powerful services at the communication layer can benefit many types of online multiplayer games.

This chapter presents the generic structure of online multiplayer games and their communication related requirements (Section 2.1), previously proposed game-level optimizations (Section 2.2), an analysis of transport protocols' efficiency for online multiplayer games (Section 2.3), players tolerance to network conditions for game and VoIP (Section 2.4), and finally, previous research on network QoS (Section 2.5) and application level multicast (Section 2.6).

2.1 Online Games Structure and Generated Network Traffic

Games build virtual environments, known as "game worlds", where players interact with each other and with small set of objects. The game state is typically structured as a collection of game objects representing a part of the game world. Game objects, such as the game world's terrain, player's avatars, computer controlled players (i.e., bots), and items (e.g., guns) are defined by their state and behavior. Games run a discrete event loop and update game objects: First Person Shooter (FPS) games execute 10 to 20 such iterations each second, while Real Time Strategy (RTS) games can afford a lower iteration frequency.

When running on separate machines, online multiplayer games attempt to update the game objects for all players similarly to have the same game state on every player's machine. There are two methods for achieving this goal:

1. **Deterministic model:** In this model, every player calculates the position and actions of all players in the game (the model mainly used for multiplayer sport games). Each player broadcasts its controller inputs to all other players for everyone to update the game objects exactly the same. Consistent and exact updates are crucial in this model as game state on each device can quickly diverge.

2. **Non-deterministic model:** In this model, players send updates of their game objects (e.g., position in the virtual world, velocity, etc) to others as opposed to their controller inputs. In this model the game state is weakly consistent and frequent state updates, exchanged between players, are used to make it converge. *This thesis focuses on non-deterministic multiplayer games as they have a more*

diverse communication workload.

Network games generate a significant share of today's Internet traffic. McCreary et al. [43] report that 3-4% of all packets in the Internet backbone can be associated with only six popular games. In addition, it is expected that game traffic will account for 25% of all LAN traffic by the year 2010 [43]. Network game traffic tends to employ small, highly periodic UDP packets [18, 20–22, 38, 39]. Periodicity is due to game's dynamics, which require frequent state updates from each player; additionally the low latency requirement of online multiplayer games makes message aggregation impractical leading to a large number of messages.

2.2 Game-Level Optimizations

Most online multiplayer games are designed to saturate the narrowest last-mile links in order to support wide range of players irrespective of their network capacity [21]. As a result, game designers have been exploring game-specific solutions to decrease the amount of generated traffic, by using group communications and interest management techniques such as the ones presented in AMaze [3], Mercury [54], and Fiedler et al. [23].

Game updates, that is, messages that include information about changes in game objects' state, account for a large part of generated network traffic. For FPS games such as Quake III, game updates are broadcast every 50ms and include information such as the game objects' position, health, and state. Since most objects' attributes are rarely changed, delta-encoding [4, 5] is used to efficiently compress the generated game updates.

Another effective method to reduce the volume of generated traffic is to eliminate the transmission of irrelevant and unnecessary information. Most games leverage players'

limited Area of Interest (AOI): players are only interested in a subset of the entire game world, typically in their local area or field of view. Consequently, to run a smooth game, players only need to receive updates only from other players in the same AOI. As a result, AOI filtering reduces bandwidth demand to the extent that players have limited proximity to each other. A widely adopted solution is interest filtering by partitioning the game world into predefined [24, 30, 36, 55] or arbitrarily specified (e.g., Colyseus [5], VON [28]) disjoint AOIs.

Filtering based on AOI works well for games that have an upper limit for the number of players in any given area; however, naturally the object density in games follows a power law [51]. This has introduced a new challenge as game designers have to discourage players from clustering, a limitation that eliminates some classes of interesting game play such as epic battles [9, 59]. To circumvent this limitation Bharambe et a. [4] build on the intuition that human attention span is limited and introduce the notion of "size bounded player's interest set", defined as the set of players to whom a player is paying attention.

Finally, another method to reduce the volume of generated traffic is decreasing the game update rate. Two proposed techniques are Predictive Contract Mechanisms (PCM) [29] and multi-resolution simulation [27]. PCM requires the sender and receiver to use a shared model for predicting the object's state. The sender only sends state updates if the predictor's error exceeds a threshold. The most common PCM uses dead reckoning, extrapolating an object's position based on its past position, velocity, and sometimes other movements [49]. Other predictors include a hybrid dead-reckoning/shortest-path predictor [41] and neural networks [42].

In contrast to the past work which focuses on game-level and mostly game-specific optimizations we propose a generic communication layer that enables broad class of

games to efficiently use network resources, and integrate such game-level optimizations.

2.3 Transport Protocols for Online Games

The first fundamental decision in designing online games is selecting the transport protocol. Chen et al. [14] evaluate TCP performance in the context of online games by analyzing a 1,356 million packet trace from ShenZhou, a commercial, mid-sized Massively Multiplayer Online Role-Playing Game (MMORPG). They conclude that TCP is "unwieldy and inappropriate" for online multiplayer games due to online games' traffic characteristics: 1) tiny packets, 2) low packet rate, 3) application-limited traffic generation, and 4) bidirectional traffic. In addition, TCP's window-based congestion control and the fast retransmit algorithm for loss recovery are ineffective and an overkill since not all game packets need to be transmitted reliably and in-order.

Chen et al. [14] recommend that a transport protocol specialized for online multiplayer games should:

i) Support both reliable and unreliable delivery

ii) Support both in-order and out-of-order delivery

iii) Support accumulative delivery

iv) Allow multiple streams

v) Implement a coordinated congestion control mechanism.

2.4 Players Tolerance to Degraded Network Conditions

The growth in the popularity of online interactive tools and applications such as interactive network games and VoIP has motivated researchers to better understand the effects of degraded network conditions (e.g., packet loss and increased latency) on users' perceived QoS.

This section presents past research on user perceived quality for online multiplayer games and VoIP, both highly correlated to the end to end delay. Within this thesis, and for quality estimation purposes, the communication delay for all network traffics is considered to be the time from event generation to the moment the consequent resulting updates are presented to users. To avoid extra memory and computation overheads, average event recognition and presentation delays are used. For example, a workload generating updates every 20ms has an average delay of 10ms for recognizing an event, and a 10ms delay for presenting it to the player once events are received.

2.4.1 Online Multiplayer Games

Player tolerance to message loss and delay varies from game to game. Generally, games where a player guides an avatar, rather than directly controls the game action, better tolerate latency. For example, strategy-oriented or role-playing games tolerate message latencies up to several seconds [2, 57] since the player controls the game by "telling" his avatar what to do such as "pick up object" or "attack monster", rather than how to do it (e.g. aiming and firing a gun in FPS games).

Sheldon et al. [57] showed that network latency up to several seconds has a minimal effect on the overall players' performance in Warcraft III (RTS game). Nichols et al. [45] achieved similar results with Online Madden NFL Football for delays below 500ms, while

latencies higher than this threshold degrade performance by almost 30%. Pantel and Wolf [48] analyzed another sports game, a car racing simulation, and found that delay above 50ms affects game results, while delays above 100ms prevents a realistic driving behavior.

Players in FPS games move rapidly and must react quickly to their surroundings. Consequently, limiting lag, the time difference between a player's visible state and his actual state, is crucial for a satisfactory game experience. Beigbeder et al. [2] examined the effects of loss and delay on users' performance in Unreal Tournament 2003. They explain that players were able to notice delays as low as 75ms and found game play less enjoyable at latencies over 100ms. They find that precision shooting is very sensitive to latency, with a steady decrease in hit accuracy for latencies of 100ms or over. The authors suggest to keep the delay below 150ms and the packet loss below 3%. Finally, Quax et al. [53] notice two groups of players, the "complainers" and the "optimists" which perceive the game quality differently.

From all of the existing research, we conclude that degraded network conditions in general have a negative effect on how players perceive a game, while the exact threshold varies between games of different categories. Furthermore, findings for all games are similar in a sense that for every game latencies lower than a game specific threshold don't affect players, and latencies higher than a certain game specific threshold can be completely intolerable, while perceived quality decreases in between.

2.4.2 Voice Over IP (VoIP)

Past literature on end-to-end VoIP measurements has often focused on the study of network loss patterns and delay characteristics [10, 11, 35, 50, 63]. For example, Kostas [37] studied the feasibility of real-time voice over the Internet and discussed measured delay

and loss characteristics. In order to evaluate the quality of VoIP, Marsh [40] provided network performance data (in terms of delay and losses) collected from a wide range of geographically distributed sites. All these studies were based on round-trip delay measurements. While information about delay and losses can give valuable insights about the quality of VoIP, they do not characterize the actual subjective quality experienced by VoIP users. Cole et al. [17] propose a method for monitoring the quality of VoIP applications based upon a reduction of the E-model [34] to measurable transport level quantities (such as delay and loss).

The classic way to evaluate speech quality is Mean Opinion Score (MOS) [31]. However, it is time consuming, costly, and not repeatable, as human experts are involved in the evaluation. Speech Quality Measure (PSQM) [32] and Perceptual Evaluation of Speech Quality (PESQ) [33] are the most common objective measurement methods for voice quality. Both still require a reference signal to compare a degraded speech signal against and predict a MOS value. They are called psychoacoustic models.

The Telecommunication Standardization Sector (ITU-T) E-model does not depend on a reference signal, but uses a computational model to predict voice quality directly from network measurements. The output of the E-model is a single value, called an "R-factor", derived from delays and equipment impairment factors. To present the correlation between the "R–factor" and VoIP quality, the ITU-T G.107 [34] defines the relationship between the R-Factor and MOS as below (Equation 2.1) resulting in Table 2.1:

$$MOS = \begin{cases} 1, & R \leq 0 \\ 1 + 0.035R + R\,(R-60) \times (100-R) \times 7 \times 10^{-6}, & 0 < R < 100 \\ 4.5, & R \geq 100 \end{cases} \tag{2.1}$$

R-Value Range	MOS	VoIP Quality
100 – 90	4.50 – 4.34	Best
90 – 80	4.34 – 4.03	High
80 – 70	4.03 – 3.60	Medium
70 – 60	3.60 – 3.10	Low
60 – 0	3.10 – 1.00	Very Poor

Table 2.1: VoIP Transmission Quality Classes and Corresponding Rating Value Ranges

The R-Factor calculated by the E-model ranges from 0 (poor) to 100 (excellent) and can be obtained by the following expression:

$$R = R_o - I_s - I_d - I_{e-eff} + A \tag{2.2}$$

where

R_o : Basic signal-to-noise ratio

I_s : All impairments that occur simultaneously with the voice signal

I_d : Delay impairment factor

I_{e-eff} : Effective equipment impairment factor caused by packet loss

A : Advantage factor

Cole et al. have reduced Equation 2.2 to Equation 2.3 after taking default values for those parameters other than delay and loss [17].

$$R = 94.2 - I_d - I_{e-eff} \tag{2.3}$$

In this thesis, we use Equation 2.3 simplified according to [17] for translating one-way delay (d), in milliseconds, and loss rate (e) to I_d and I_{e-eff} in order to measure the quality of the transmitted VoIP.

$$I_d = 0.024d + 0.11(d - 177.3)H(d - 177.3) \tag{2.4}$$

$$I_{e-eff} = 0 + 30ln(1 + 15e) \tag{2.5}$$

where

$$H(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \tag{2.6}$$

2.5 Network Quality of Service

Today's Internet infrastructure provides one simple service: best effort datagram delivery. This minimalist service allows the Internet to be stateless, that is, routers do not need to maintain any fine grained information about traffic. As a result of this stateless architecture, the Internet is both highly scalable and robust. However, as the Internet evolves into a global commercial infrastructure that supports a many new applications such as online games, IP telephony, interactive TV, the existing best effort service is extended to provide more powerful services such as guaranteed services, differentiated services, and flow protection.

IntServ [7] and DiffServ [58] provide differentiated services that classify and manage network traffic to provide network QoS guarantees; howerver, these services require network infrastructure support as opposed to application-level solutions such as OverQoS [60].

Among the solutions that do not require network support, OverQoS uses Controlled Loss Virtual Links (CLVL) to smooth packet losses, prioritize packets within an aggregate, and provide statistical bandwidth and loss guarantees. One of CLVL's main

features is supporting "Bundle Loss Control", which prevents losing many packets in a row.

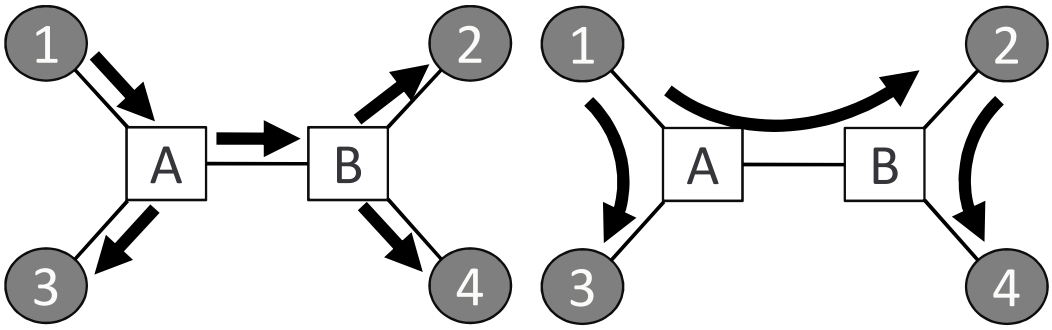
The network QoS solutions generally filter, duplicate, and send packets based on certain heuristics to provide different classes of service. MCCA uses similar ideas to provide targeted network QoS and differentiated services for a game to properly manage its heterogeneous network traffic and improve gaming experience.

2.6 Application Level Multicast

Multicasting is a mechanism for packet delivery in one-to-many data transfer applications. Multicasting can be provided at network infrastructure (IP–Multicast) or at the application level (Application Level Multicast). IP–Multicast is the optimal multicast protocol that eliminates redundant packet replication in the network infrastructure; however, deployment of IP multicast has not been widely adopted by most commercial ISPs.

Application Level Multicast protocols do not require support from the network infrastructure, instead they implement multicast forwarding functionality exclusively at end-nodes. Such application level multicast protocols are increasingly being used to implement efficient commercial content distribution networks. Consequently, application level multicast and overlay construction has been extensively studied [12, 13, 26, 51, 56].

The basic idea of application level multicast is shown in Figure 2.1. Unlike IP multicast, where data packets are replicated at routers inside the network, in application level multicast data packets are replicated at end nodes. Since application level multicast protocols must send the identical packets over the same link, they are less efficient than IP multicast. Two intuitive measures of "goodness" for application level multicast



(a) IP Multicast (b) Application Level Multicast

Figure 2.1: *IP Multicast vs. Application Level Multicast.* Square nodes are routers, and circular nodes are end-nodes.

overlays, namely stress and stretch, are defined in [15].

1. **Stress:** Defined per-link and counts the number of identical packets sent by a protocol over each underlying link in the network.

2. **Stretch:** Defined per-member and is the ratio of transfer delay from the source to the member along the overlay to the delay of the direct unicast transfer.

Overlay construction algorithms are generally categorized as mesh-based or tree-based. Mesh-based overlays provide multiple paths, while tree-based overlays provide a single path between any pair of nodes. In the mesh-based approach, group members first organize themselves into the overlay mesh topology. Multiple paths exist on the mesh between a pair of members. Each member participates in a routing protocol on this control topology to compute unique overlay paths to every other member. A source specific tree rooted at any member can then be created using the well-known Reverse Path Forwarding (RPF) based construction used by many IP multicast protocols (e.g., Distance Vector Multicast Routing Protocol (DVMRP) [62]). In contrast, tree-based protocols that distributedly construct a shared data delivery tree at first. Subsequently,

each member discovers a few other members of the multicast group that are not its neighbors on the overlay tree and establishes and maintains additional control links to these members. This enhanced overlay (the data delivery tree with the additional control links) is the control topology in the tree-first approach.

Online multiplayer game event delivery requires four properties from a multicast scheme: 1) a strict bound on end-to-end latency, 2) support for frequent group membership changes, 3) support for heterogeneous client capacity, and 4) ability to scale with the number of groups and total number of participants.

Structured (DHT-based) multicast designs [13, 51] allow nodes to join quickly and minimizes control overhead. Thus, such schemes handle the points above 2 and 4 well. However, it is difficult to optimize latency or handle node heterogeneity effectively [6]. At the other end, unstructured approaches [1, 15, 47] allow clients to join anywhere and try to optimize the tree structure. As a result, they are better at supporting (1) and (3) but worse at (2) and (4) due to control overhead. Unstructured, source controlled tree structures [4] minimize control overheads and can still scale to large number of players. CoopNet [47], used for multicasting video frames, uses source-controlled trees and requires Multiple Description Coding to encode video frames over multiple trees to handle churn and packet loss. Game updates as opposed to video frames are too small relative to packet headers to be efficiently encoded in this way.

MCCA is designed for online multiplayer games with tens to hundreds of players per session, and uses application level multicast (tree-based overlays) to reduce the volume of generated traffic for classes of data with relaxed latency requirements.

Chapter 3

Architecture

Most soft real-time distributed applications require a combination of reliable/unreliable, in-order/out-of-order message delivery. Online multiplayer games are one of the most complex real-time distributed applications that benefit from concurrently using multiple communication modes, as some messages have tight latency requirements and require only limited transmission reliability (e.g., game updates), while others have more relaxed latency requirements (e.g., VoIP) and require reliable and in-order delivery (e.g., file transfer). Consequently, online multiplayer games require a combination of reliable/unreliable, in-order/out-of-order delivery with latency requirements varying depending on the type of data transferred.

Online multiplayer games, as described in Section 2.1, essentially present a controlled environment, simulating virtual worlds, which allows players to interact with the world's objects and one another. Simulating and controlling the virtual world enables game–level decision making, which can be exploited to improve game quality and/or to support higher number of players (Section 2.2). MCCA, the architecture presented in this chapter, is designed to exploit game–level opportunities, and use game insight at the communication layer to efficiently utilize network resources.

To effectively utilize network resources, support a diverse set of requirements and network traffics generated by online multiplayer games, and to enable games to dynam-

ically adapt to the network conditions, MCCA supports:

1. The ability to define different classes of service in order to provide targeted network QoS (differentiated services)

2. The ability to label messages, and consequently assign different game generated network flows (i.e., game updates, VoIP, text messages, etc) to a class of service

3. Flow specific efficient group communication/message dissemination to distribute network traffic according to player's network resources and flow's requirements

4. Flow specific quality estimation, aggregation, and reporting to enable game adaptation to network conditions

The remaining sections of this Chapter present the MCCA's support for differentiated services (Section 3.1), group communication (Section 3.2), and distributed quality aggregation (Section 3.3). Following which, the internal implementation details of MCCA are presented (Section 3.4), and MCCA's usability issues are discussed (Section 3.5). Finally, Section 3.6 compares MCCA to other existing solutions for online multiplayer games.

3.1 Differentiated Services

To support a diverse set of flows (i.e., game updates, VoIP, text messages, etc) and their requirements, MCCA allows games to define their own classes of service and to label the generated network messages with the appropriate class information. The game specifies each class's priority, communication model (e.g., unreliable and out-of-order), latency constraints, and other information (described in Sections 3.1.1). Following the

class definition, each flow of network traffic is assigned to a class of service (described in Section 3.1.2) with a flow specific group communication (described in Section 3.2) and quality estimation (described in Section 3.3). From this point, the communication infrastructure uses a separate virtual channel for each class to manage and transfer messages: it reads the outgoing message queues, processes messages based on their class definition, and attempts to meet each class' requirements in the order of their priorities in a best effort fashion (Figure 3.1).

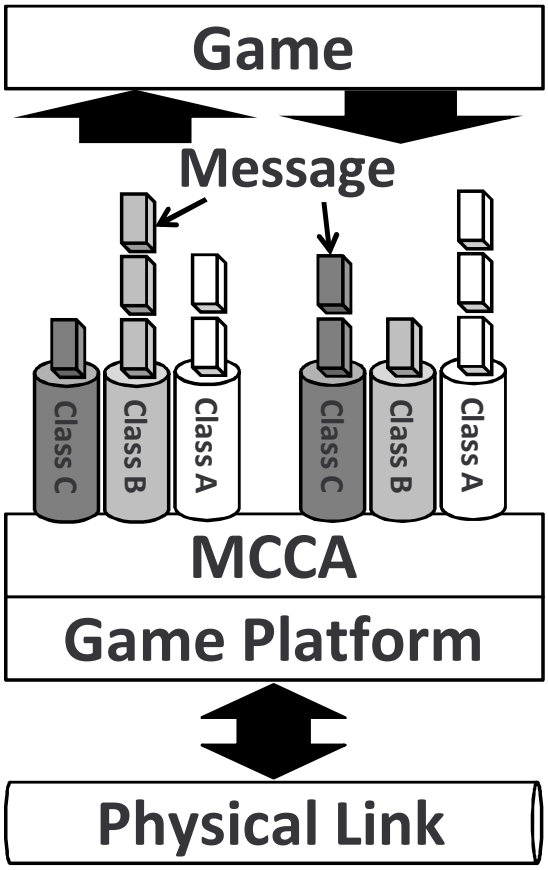


Figure 3.1: MCCA Communication Architecture

3.1.1 Network Traffic Class Definition

MCCA is a communication architecture targeted for game developers, who usually have a great deal of insight into the game's network generated workload and requirements. In order to take advantage of such game-level knowledge at the communication layer, MCCA requires the following set of parameters for defining a class of network traffic:

- **Priority:** To differentiate messages based on their importance to the game, and to decide the order in which messages are serviced; MCCA requires a game to specify each class's priority.

- **Reliable/Unreliable, In-Order/Out-of-Order:** Reliable and in-order communication is a required characteristic for many classes of data such as file transfer and game patches. This requirement often implies higher latency, due to queuing, and higher bandwidth consumption because of message retransmission. Most game messages, however, do not need in-order processing and/or reliable transmission as newer messages overwrite the older ones. Separating the reliable and in-order traffic from the rest helps games in using the network resources efficiently, and reduces the communication latency.

- **Bit-rate:** To limit the amount of pending traffic, and to provide rate control for each class of network traffic, MCCA requires games to specify each class's maximum bit–rate (i.e., maximum bandwidth associated to the class). Bit–rate is represented by the communication window–size (described in Section 3.4) in MCCA, which limits the number of unacknowledged network messages. Such limitation, in conjunction with the packet size limit, acts as a bit–rate limiter for each class of network traffic.

- **Latency Requirements and Loss Rates:** On a lossy communication channel, redundant message transmissions is a common method to increase reliability without increasing transfer latency (as it avoids waiting for acknowledgments to detect message loss). This technique however introduces obvious overheads: increased traffic volume. MCCA introduces the concept of *redundancy level*: the number of times a duplicate message is transmitted. In addition, to avoid bursty losses and lower the redundancy overhead, MCCA introduces the concept of *redundancy interval*: the duplicate messages are spread over time with the specified intervals to lower the chance of losing all duplicate messages due to a network loss burst. The redundancy interval also provides an amount of time to possibly receive acknowledgments from the receiver to avoid unnecessary retransmissions: the game specifies the amount of time that is required to pass prior to resending the message. The redundancy interval affects the average communication latency; however, a game developer can make informed choices on the redundancy interval for each class based on the knowledge about the game's workloads and behavior. Message redundancy increases the potential of successfully transmitting a message to the receiver, and avoid message timeouts and re-transmition. Such technique in combination with the described redundancy interval, provide the means to the game to define the loss and latency requirements of its network traffics.

Program 3.1 demonstrates the MCCA's API for creating and destroying a class of network traffic[1]:

[1]Refer to Program A.1 in Appendix A for the definition of types and enumerations

Program 3.1 Network Traffic Class Creation and Destruction

```
// Precondition:  The class specified by the ‘classId’ should be
//                undefined.
// Postcondition: The class specified by the ‘classId’ will be defined.
//                Upon unrealistic parameters, or predefined class, an
//                error message is returned; otherwise, Error_OK is
//                returned.
Error createNetworkTrafficClass(
  __in  ClassId classId,
  __in  unsigned int priority,
  __in  bool isReliable,
  __in  bool isInOrder,
  __in  unsigned int communicationWindowSize,
  __in  unsigned int redundancyLevel,
  __in  double redundancyInterval );

// Precondition:  The class specified by the ‘classId’ should be defined
// Postcondition: The specified class will be destroyed, and
//                Error_OK will be returned. If the class is undefined
//                an error message will be returned.
Error destroyNetworkTrafficClass(
  __in ClassId classId );
```

3.1.2 Flows and Network Traffic Labeling

MCCA enables network traffic differentiation by enabling a game to assign each of its network traffic flows (referred to as a "flow" in this thesis) to a defined class of service (Section 3.1.1). Due to such separation, multiple flows could belong to the same class (presented in Figure 3.2) and benefit from a flow specific group communication (described in Section 3.2) and quality estimation (described in Section 3.3).

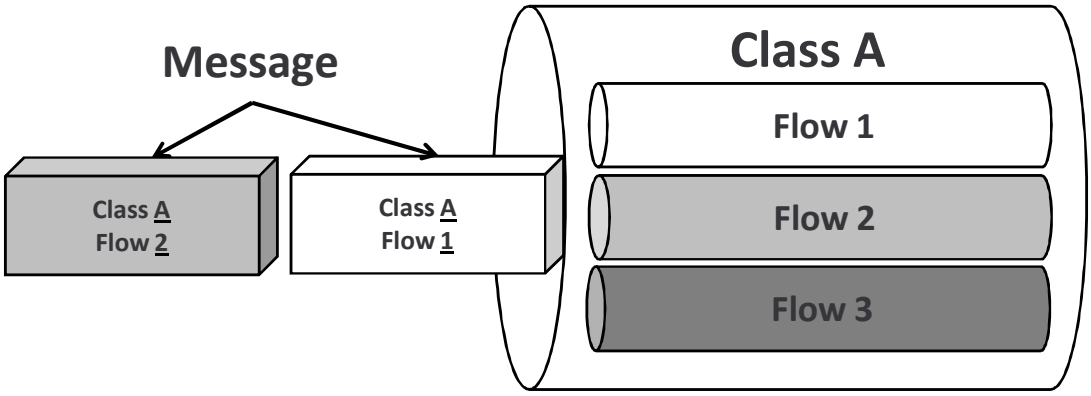


Figure 3.2: Network Traffic Flows and Class Structure in MCCA

Consequently, MCCA requires a flow to be registered with a defined class of service. The game is then required to specify a set of required and optional callbacks for receiving messages, notifications, and more. Program 3.2 presents the MCCA's API for flow registration [2].

Upon class definitions and flow registration, the game can begin pushing messages labeled with their Class–ID and Flow–ID to the communication layer (Program 3.3 presents the API for sending messages). MCCA pushes received and ready–to–process messages to the game through the flow's registered message receiver (see "FlowCall-

[2]Refer to Program A.2 in Appendix A for the detailed interface of the *FlowCallback*

back::OnReceive" in Program A.2).

Program 3.2 Register and Unregister a Flow

```
// Precondition:  The class specified by the 'classId' should be
//                defined, and the flow of 'flowId' should not be
//                registered, and the callback should not be null.
// Postcondition: The flow and its callback of the 'flowId' will
//                be registered with the specified class. An error
//                message is returned if conditions are not met;
//                otherwise, Error_OK is returned.
Error registerFlow(
  __in  ClassId classId,
  __in  FlowId flowId,
  __in  FlowCallback* callback );

// Precondition:  The specified class and the flow should be
//                defined and registered respectively.
// Postcondition: The specified network traffic is unregistered,
//                and Error_OK is returned. An error message is
//                returned if conditions are not met.
Error unregisterFlow(
  __in ClassId  classId,
  __in FlowId   flowId );
```

The presented hierarchy of "classes of service" and "flows" can reduce the number of classes defined by the game, which reduces the amount of memory and execution overhead in such communication architecture, which is highly beneficial to any game. To simplify and achieve a flat hierarchy, a one-to-one mapping of flows to classes could be applied; however, we believe such hierarchy can be beneficial to online multiplayer games as it decouples the class of service from the broadcast and quality estimation techniques. For example, in a given online multiplayer game all game updates can share the same class definition, while the game can benefit from a one-to-one mapping between the categories of game updates and flows. In turn, the game may use a different

Program 3.3 MCCA's Interface for Sending Messages

```
// Precondition:  The class specified by the 'classId' and the flow
//                specified by the 'flowId' has to be defined.
// Postcondition: The message is placed in the appropriate send
//                queue and is essentially sent. Error is returned
//                if the class or network traffic is not defined
//                and/or the communication window is full; otherwise,
//                Error_OK is returned.
Error send(
  __in  ClassId classId,
  __in  FlowId flowId,
  __in  void* message,
  __in  unsigned int messageSize );
```

broadcast technique depending on the game update's urgency (e.g., using a multicast tree for broadcasting player's health, points, etc), or target group (e.g., disseminating group messages in capture the flag games to group members only).

3.2 Group Communication

Online multiplayer games require the ability to send messages to a single target (point-to-point/private) or to a set of targets (group broadcast). Considering that all communications are based on a series of point-to-point communication, every message in MCCA is tagged with a "MessageType" [3] indicating whether it is a private or group broadcast message. Private messages are messages transfered only among two players/nodes, while a forwarding policy, specified by the game, is used to propagate a broadcast message to all group members.

Efficient group communication is an essential step in decreasing the network traffic

[3] Refer to Program A.1 in Appendix A for the definition of types and enumerations

and better utilizing network resources. Using MCCA, the game can specify a customized forwarding policy (e.g., a two level multicast tree optimized for bandwidth) for each of its flows. Supporting a flow specific forwarding policy can be extremely beneficial to games considering different flows have different requirements, behavior, and characteristic. Furthermore, the game can assign one of the general forwarding policies supported by MCCA to a flow, or define and re-use forwarding policies of its own.

3.2.1 MCCA Supported Forwarding Policies

A communication layer for online multiplayer games, such as MCCA, can internally build and support a few relatively general broadcast topologies for the game. MCCA supports the following broadcast topologies:

- **Star:** Simply a one-to-all message broadcast policy. "Star" is a network bandwidth consuming policy. "Star" does not scale as its generated network traffic grows linearly with the number of players; however, it is potentially the best forwarding policy for network traffics with extremely tight latency constrains.

- **Two–level multicast tree:** Two–level multicast tree uses other players/nodes as relay agents to propagate messages. MCCA incorporates two level multicast trees to minimize the introduced delay/latency in propagating messages. Other multicast trees with deeper hierarchies can also be supported and become available to games; however, they are not explored in this thesis considering the online multiplayer games' latency constraints.

  In order to support players joining and leaving the game quickly, the multicast overlay is created randomly at the beginning. The overlay adjusts itself as time

goes on, when more information about players' network conditions (e.g., upload capacity and latency) are gathered.

In building multicast trees different factors such as player's bandwidth, latency or even group membership can be taken into account. MCCA supports the following two types of multicast trees:

- **Optimized for available bandwidth:** The primary factor for building the multicast overlay is players' available bandwidth for forwarding. This policy will essentially migrate the network load from players with low upload bandwidth capacities to players with higher upload bandwidth capacities.
- **Optimized for latency:** The primary factor for building the multicast overlay in this case is the end-to-end latency.

Ping messages, sent frequently, are used to carry information such as the amount of available forwarding bandwidth, and measuring communication latencies. The updated information are used to adjust and re-build multicast trees. Potential changes in a tree translate to requests for a new forwarder. Once the request is accepted by the forwarder, the tree is updated and the previous forwarder is released through notifications. A forwarder accepts a request only if it has enough resources, and if the requester has the most up-to-date information about the forwarder. Upon an out-of-date requests, a forwarder rejects the requests and provides updated information in order for the requester to make a new decision.

3.2.2 Custom Forwarding Policy

Using MCCA a game can define its own forwarding policy. This can be useful as games are able to use network information and/or game-level information (e.g., players' team

affiliations) to build a forwarding policy, which better fits the needs and requirements of their corresponding workloads. Program 3.4, presented on the next page, demonstrates the forwarding policy interfaces, essentially answering two questions:

1. To whom a new broadcast message, using this forwarding policy, should be sent to?

2. To whom a received broadcast message should be forwarded to?

A reasonable forwarding policy for online games has to adapt to new changes, considering the frequent changes in the game state (e.g., players leaving or joining the game) and network conditions (e.g., updated information about the latency between players as well as their available forwarding bandwidth). Consequently, MCCA requires the game to register a forwarding policy creator [4] with it, which is used to create, update, and manage a game defined forwarding policy. Program 3.5 presents the forwarding policy.

3.3 Quality Aggregation and Reporting

One of the characteristic features of online multiplayer games is their generated traffic volume predictability, and their ability to dynamically adapt and adjust to improve the perceived QoS. Some of the mechanisms such as AOI and PCM that exploit these opportunities have been discussed in Section 2.2. Such optimizations could be altered to dynamically adjust the game generated network traffics and their interpretation, depending on the observed network quality, to improve players' experience. In order to do so, a game needs to estimate the observed QoS for its flows. Consequently, MCCA

[4] Refer to Program A.3 in Appendix A for the interface of the forwarding policy creator

Program 3.4 Forwarding Policy Interface

```
class ForwardingPolicy {
  // Postcondition: Returns the list of players that a broadcast message
  //                using this forwarding policy should be send to.
  Error GetSendListFor(
    __out PlayerId** forwardees,
    __out int* numForwardees );

  // Postconditions: Returns the list of nodes that a broadcast message,
  //                 received from ‘‘fromPlayerId’’, should be send to
  //                 via this node.
  Error GetForwardListFor(
    __in  PlayerId fromPlayerId,
    __out PlayerId** forwardees,
    __out int* numForwardees );
};
```

Program 3.5 Setting the Forwarding Policy for a Network Traffic Flow

```
// Precondition:  The class specified by the ‘classId’ and the flow
//                specified by the ‘flowId’ has to be defined. The
//                forwarding policy type has to be one of the MCCA
//                supported types, with ‘‘customForwardingPolicy’’
//                set to NULL; otherwise, the game has to specify
//                its custom forwarding policy.
// Postcondition: The forwarding policy for the specified network
//                traffic is set. Error_OK is returned if the pre-
//                conditions are met, and if the forwarding policy
//                is set properly; otherwise, an error message is
//                returned.
Error setFlowForwardingPolicy(
  __in ClassId classId,
  __in FlowId flowId,
  __in ForwardingPolicyType forwardingType,
  __in CustomForwardingPolicy* customForwardingPolicy = NULL );
```

treats the game as an intelligent, and most importantly, an adaptable entity that can change its behavior to improve the estimated quality. This is done by estimating and aggregating each flow specific quality; to report the estimated perceived quality to the game.

Many have studied the effects of network conditions on the player's perceived QoS [2, 45, 48, 53, 57] (presented in Section 2.4). Based on these studies, MCCA provides a set of standard and targeted quality estimation techniques for the game. In addition, MCCA enables a game to define its own quality estimation technique (Section 3.3.3).

A game can specify the quality estimation type (see "QualityEstimationType" in Program A.1, and Program A.4 in Appendix A for the interface) each of its flows. MCCA tracks latency distribution, average loss, and average jitter of messages for a flow that requires quality estimation (i.e., for flows with quality estimation type set to other than "QualityEstimationType_None") for estimating the player perceived quality. Program A.4 in Appendix A presents the detailed interface for setting the quality estimation type, the quality estimation interface, and the latency distribution interface.

MCCA periodically gathers the observed quality from all players by essentially measuring each player's estimated observed quality for the flow X and player Y combination for the last period. To consider the quality of previous periods, MCCA applies an aging factor described in Section 3.3.1, and finally reports the gathered quality to the game via the game's registered callback (see "FlowCallback::OnObservedQualityUpdate" in Program A.2 for the notification callback interface).

Furthermore, MCCA provides information about the number of consecutive message losses to better facilitate delta-encoding (described in Section 2.2) that is a common mechanism to decrease the amount of generated network traffic. Consequently, the game

can register for a notification on N-consecutive message losses for a flow. Program A.5 in Appendix A presents the interface for registering and unregistering such notifications (see "FlowCallback::OnNConsequetiveLosses" in Program A.2 of Appendix A for the notification callback interface).

This section details the quality aging in MCCA (Section 3.3.1), the method for estimating VoIP, game, and file quality (Section 3.3.2), and the interface for defining a game specific quality estimation method (Section 3.3.3).

3.3.1 Quality Aging

MCCA periodically gathers the estimated quality (i.e., the average of other players' perceived quality of the local player's updates) for different flows, and presents it to the game. To provide a realistic view of the current estimated quality, MCCA calculates the estimated quality as a combination of the current and previous periods' qualities. An aging factor is applied to degrade the effect of a period's quality as time goes on. To decrease the required state for calculating the final estimated quality, the "exponentially weighted moving average" is calculated, where a period's quality factor is cut to half every period. Consequently, final quality could be measured based on old average quality and current period's quality (Equation 3.1).

$$New\_Average\_Quality = \frac{Current\_Period's\_Quality + Old\_Average\_Quality \times 0.5}{1.5} \tag{3.1}$$

3.3.2 MCCA Supported Quality Estimation

MCCA internally supports three generic quality estimation types: 1) for VoIP, 2) for the game (based on the model proposed in this thesis), and 3) for bandwidth constraint

workloads (e.g., file transfer). In addition, MCCA enables games to define their own quality estimation techniques, based on network conditions (reported by MCCA), and other game–level information.

VoIP Quality Estimation

Effects of network conditions on VoIP have been extensively studied in the past [10, 11, 35, 50, 63] (presented in Section 2.4.2). Consequently, Equations 2.3, 2.4, 2.5, and 2.6 are used to estimate the observed quality for VoIP.

Game Quality Estimation

Many have studied the effects of network conditions such as loss, latency and jitter on online multiplayer games [2, 45, 48, 53, 57] (presented in Section 2.4.1); however, there has not been a standard method for estimating the perceived quality as opposed to VoIP, as research have found that players' tolerance to network conditions is different from game to game.

To estimate the value of each game update message, in order to measure the user perceived quality, this thesis makes the following propositions:

1. A lost message has no value.

2. A message latency is considered to be the latency between the time of an event occurrence to the time that the event is processed (presented in Figure 3.3). The game has to specify the average delay between an event occurrence to its corresponding time–of–send. Such information can be estimated based on the games periodicity (e.g., such delay for game updates is $\frac{1}{2}$ of the game update interval).

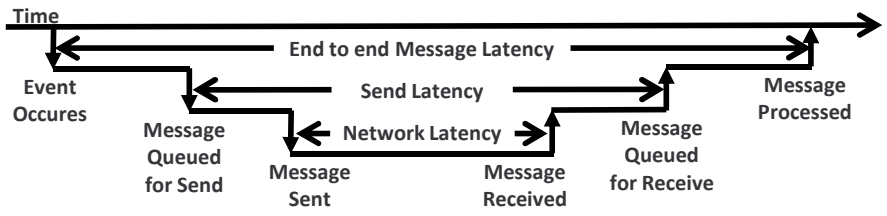


Figure 3.3: End-to-End Message Latency

3. The effect of jitter is found to be neglible [53]; thus, jitter is discarded in quality estimation as queuing penalties shown in Figure 3.3 are added (similar to VoIP).

4. A message has full value at latencies lower than the game's tolerated latency (α) and has no value at latencies higher than the game's un–tolerated latency (β) (based on the previous research [2, 45, 48, 53, 57], discussed in Section 2.4.1). The α and β values are estimated, based on user studies, for many online multiplayer games, demonstrating the ability for game developers to either estimate such values or measure via user studies.

5. A message value decreases linearly between the α and β latencies. There are other possibilities such as exponential or logarithmic value depredation; however, considering that the message latency measurement in distributed systems can not be completely accurate, linear value depredation is arguably the better choice as it avoids any abrupt changes in the message value.

Figure 3.4 demonstrates the effect of latency on the game message quality in MCCA. Consequently, the game quality can be estimated using the Equation 3.4.

$$H(a,b) = \begin{cases} 0, & \text{if } a \leq b \\ 1, & \text{if } a > b \end{cases} \tag{3.2}$$

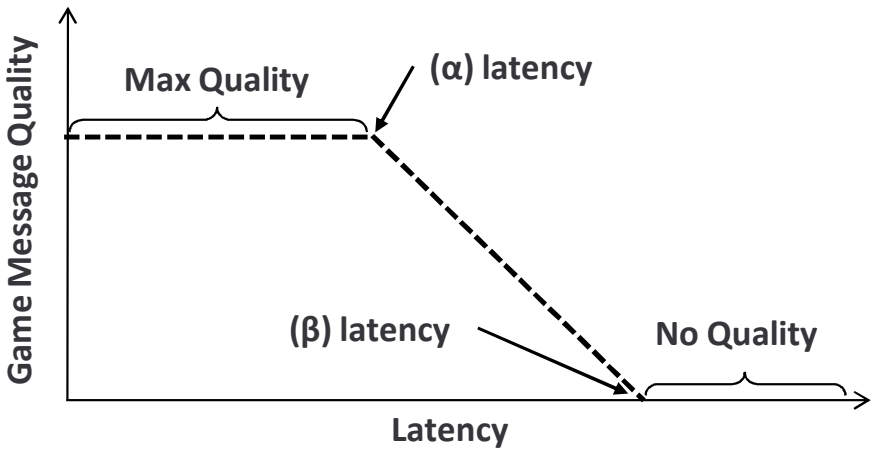


Figure 3.4: Game Message Quality vs. Message Latency

$$H'(a,b) = 1 - H(a,b) \tag{3.3}$$

$$Quality = (1-\sigma) \times \sum_{\lambda=\lambda_0}^{\lambda_n} \left(H(\lambda,\alpha) \times H(\beta,\lambda) \times \left(\frac{\lambda-\alpha}{\beta-\alpha} \right) + H'(\lambda,\alpha) \right) \tag{3.4}$$

where

σ : Message loss rate for the period

λ_0 : Latency of the first message received in the period

λ_n : Latency of the last message received in the period

α : Perfectly tolerated latency

β : maximum tolerated latency (always $> \alpha$)

Observed Bandwidth as a Quality Measure

The success of operations such as file transfer can be measured in terms of the observed bandwidth. This is a special case of quality estimation and aggregation, where the quality is calculated based on the amount of time messages spend in one of MCCA's queues until they are discarded.

3.3.3 Custom Quality Estimation

Using MCCA a game can define its own quality estimation methods by setting a flow's quality estimation type to custom ("QualityEstimationType_Custom" in Program A.4) and specifying the quality estimation callback ("QualityEstimationCallback" in Program A.4 of Appendix A) while specifying a flow's quality estimation type ("setFlowQualityEstimation" in Program A.4 of Appendix A).

MCCA internally gathers the network level information of the messages, and provides them to the game, in order for it to calculate a period's observed quality ("QualityEstimationCallback:: GetQuality" in Program A.4). A game can use such information in combination with its game-level stats to report a quality back to MCCA. The reported quality is then propagated and reported as described in Section 3.3. Appendix A presents more details of the MCCA's API.

3.4 MCCA's Internal Implementation Details

MCCA internally uses a communication window for every player per class of data to manage network traffic. The communication window is variable, and it is specified upon class definition (described in Section 3.1.1). The communication window is used to 1) assign a unique ID to each message, 2) orderly queue received messages, 3) recognize missing or duplicate messages, and 4) track and acknowledge received messages. A communication window is essentially constructed from a Start ID, cursor ID, and window–size. The specified parameters are used to track the current state of the communication window: the valid ID range ("$start$" to "$start + size$", using circular increments where the ID following "$2 \times size$" is '0'), the available window-size, and the next available ID. MCCA sends acknowledgment, which is the ID of the last successfully processed

message, to advance the communication window.

The class specific communication window is used for rate limiting (i.e., limit the amount of pending network traffic) that is discussed in Section 3.1.1. Fixed timeouts are used to resend unacknowledged messages of reliable classes of network traffic, or to simply detect and report lost messages for unreliable classes. For unreliable classes of network traffic, a timer is set when an out of order message (one that is not at the head of the communication window) is received. Once the timer expires without receiving the missing messages, the communication window advances, essentially declaring the messages lost.

Finally, considering that a game services different game components (such as the communication layer) at fixed intervals, for optimization purposes, MCCA places messages requiring redundancy in time buckets to track and retransmit them once needed. Following section describes the implementation details of the MCCA's communication layer.

3.4.1 Communication Layer

The implemented communication layer attempts to satisfy each class's requirements in a best effort fashion, by sequentially serving the highest priority messages in send queues. Such prioritization is also applied for sending redundant messages (i.e., a redundant message of a higher priority class is served prior to new messages of a lower priority class). Finally, the communication layer is implemented based on the interfaces discussed in Chapter 3 and Appendix A.

The communication layer handles the communication window for each class of data, and piggy-backs acknowledgments. Messages are retransmitted, if acknowledgments are not received within a certain amount of time (time-out), for classes of service re-

quiring reliability. Furthermore, message IDs are used to correctly deliver the messages requiring in-order processing.

The communication layer frequently pings other players, to check for premature disconnections. Ping messages are used to carry other information such as the estimated qualities (described in Section 3.3). Ping messages are used to measure the Round Trip Time (RTT) between players, which are also used to build and update the two level multicast tree optimized for latency (described in Section 3.2.1). In addition, to construct the two level multicast tree optimized for available forwarding bandwidth (Section 3.2.1), each player broadcasts its own available forwarding bandwidth to other players. Based on such informations, each player builds a multicast tree optimized for itself, and sends requests (to assign forwardees to a forwarder) and responses (accept or deny). Once a player joins, leaves, or receives updated information of other players, multicast trees are adjusted.

Since game messages are relatively small (tens to hundreds of bytes) the communication layer combines the available outgoing messages that share the same target into one packet as long as the combined packet size does not exceed the UDP packet limit. This reduces generated traffic as for example packet headers and security keys add about 44 bytes of overhead in Xbox 360 [46].

3.5 Usability Issues

The experiments described in Chapter 5 demonstrate the benefits of the proposed communication architecture, MCCA, for online multiplayer games. This section focuses on challenges involved in using MCCA and discusses a few potential solutions.

Improving the efficiency, and better managing the network resources, is an important

requirement for any communication layer for online multiplayer games; however, it is not the only factor. The communication layer should also provide flexibility and usability to developers, without requiring a developer's knowledge of the communication layer's internal details. In order to evaluate MCCA with regards to these conditions, it is important to outline this thesis's assumptions in designing MCCA:

1. A game's major and frequent network traffics use the MCCA for network communication

2. Game developers have an understanding of the game behavior and the generated network traffic

3. Game developers have minimal background in network application development

In general, MCCA supports traffic classification and prioritization, and provides efficient group communication, as well as reports on network traffic loss and estimated quality. Fine-tuning the class definitions is arguably the most complex step in using MCCA, as developers need to consider the five parameters that define a traffic class: 1) order, 2) reliability, 3) redundancy level, 4) redundancy interval, and 5) window-size. Order and reliability are dependent on the network traffic's semantics, which are relatively easy to infer from game requirements, while the other three options require a better understanding of networked applications. Background in network application development and clear understanding of application requirements, help in correctly defining each traffic class; however, the following generic solutions can reduce the complexity and result in better traffic classification:

1. **Simulation based parameter search:** Game developers can accurately model and/or trace a games's network traffic based on their insight and control of the

game. Such information can be the input data for a simulator that searches different class parameters, at various deployment conditions, to find the best possible class definition. In addition, the result can specify the best class definition for different number of players and/or known network conditions, which provides flexibility to game developers, and potentially improves player experience.

2. **Machine learning algorithms:** Neural networks and machine learning algorithms are actively used in Artificial Intelligence (AI) for games, to provide higher user satisfaction, and for the game to adapt to the player's skills and habits. Considering that online multiplayer games traditionally support players with last-mile links, using neural networks in combination with MCCA to define and change class definitions at runtime would enable player specific solutions (similar to AI for games). For example, players with access to high network capacities can send redundant messages, to account for network loss, while players with lower network capacities would avoid message redundancy to reduce network traffic.

3.6 Discussion

This section compares MCCA to existing game-level optimizations (Section 3.6.1) and network QoS solutions (Section 3.6.2).

3.6.1 Comparison to Game-Level Optimizations

Game-level optimizations are a key factor in supporting higher number of players and improving the user perceived quality. In fact, certain game-level optimizations discussed in this thesis are developed on top of MCCA (Section 5.1.3). The following presents the main factors differentiating MCCA from previous research on game-level optimizations

described in Section 2.2:

1. Game-level optimizations are generally game or genre specific, as opposed to MCCA that is designed to support wide range of online multiplayer games

2. Game-level optimizations generally focus on reducing the network traffic, through predictions and are highly coupled with the game, and don't differentiate among different types of network traffics

3. Online game optimizations [3–5, 23, 54] proposed in the past, do not, to the best of our knowledge, support class specific targeted network QoS to optimize the generated network traffic at the infrastructure level for online multiplayer games

The goal of this research is to provide a generic yet flexible solution that can benefit many types of online multiplayer games. As a result, MCCA attempts to better manage network traffic, utilize network resources, and provide further communication layer and network traffic specific information for the game. Consequently, this thesis presents new game-level optimizations, only possible due to MCCA.

3.6.2 Comparison to Network QoS

MCCA takes advantage of network QoS concepts such as message prioritization, duplication, and rate limiting. However, MCCA is less restrictive compared to network QoS solutions (e.g., OverQoS [60]) to provide a classified and yet best effort communication layer for online multiplayer games. Consequently, MCCA is more dependent on the game to properly define its classes of network traffic to perform efficiently.

MCCA is targeted for online multiplayer games where the workload behavior can be predicted and is fully managed by the game. In fact, MCCA works with the game

(e.g., through game quality reports) for adapting to real-time conditions and improving the user perceived quality.

Finally, MCCA allows a game to specify its group communication and quality estimation techniques for each of its network traffic flows. Such game–defined group communication can play an important role in cheat prevention for online multiplayer games (e.g., in capture the flag type of games, important group information could be efficiently broadcasted to the group, through group members, in isolation from intruders).

Chapter 4

Evaluation Methodology

This chapter discusses the overall evaluation methodology for studying the effects of: game workload prioritization, classification, and support for targeted QoS. In addition, this chapter discusses the methodology for studying the effects of the proposed group communication and distributed quality aggregation and reporting. Chapter 5 presents the experiment setup and results, and discusses the findings.

The proposed communication architecture, MCCA (described in Chapter 3), is a standalone layer in between the game and the network layer (Figure 4.1). This layer separation is exploited, when evaluating MCCA, to extensively study various combinations of workload and network layer properties.

To evaluate MCCA, we had to model the behavior of the two layers surrounding MCCA. The rest of this chapter details the communication infrastructure (Section 4.1), and the set of workloads (Section 4.2) used for evaluating MCCA.

4.1 Communication Infrastructure

This section presents the set of communication infrastructures that could be used for evaluating MCCA. In addition, this section compares the potential communication infrastructures, and presents the implementation details and evaluation of the chosen communication infrastructure in Section 4.1.1.

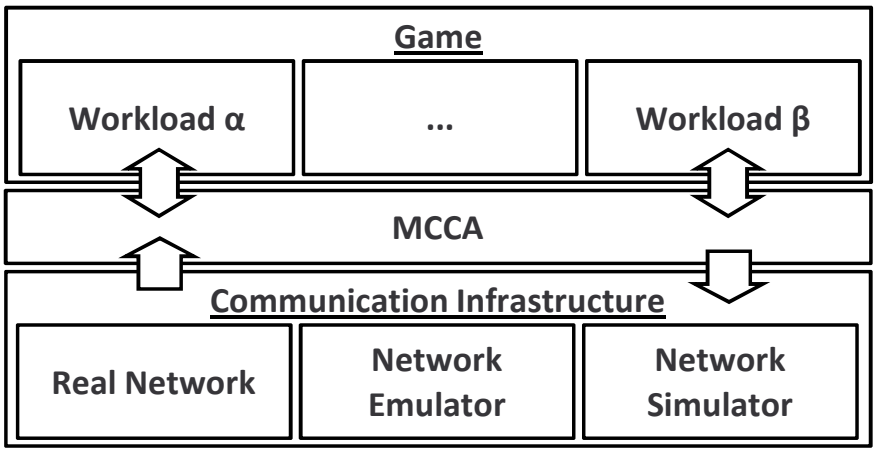


Figure 4.1: Main Deployment Layers

Clear layer separation permits the use of either of the following communication infrastructures (shown in Figure 4.1) to evaluate MCCA:

1. **Real Networks**: Provide a realistic deployment environment with higher confidence in the final results; however, it requires a lot of resources. In addition, the deployment environment is uncontrolled, which prevents reproducible experiments.

2. **Packet–Level Network Emulator**: Provides a relatively realistic deployment environment. The deployment environment can be controlled, where experiments of various deployment configurations could be performed relatively easily.

3. **Network Simulator**: Simulate the network communication among network nodes, by simulating packet or message transfers. Network simulators introduce certain error rates compared to real or emulated networks, as network upload capacity and communication latencies are all simulated. The deployment config-

urations can be easily changed, and experiments can be performed faster using virtual time.

Considering the wide range of workload and deployment configurations studied in this thesis (described in Chapter 5), we used a network simulator as the communication infrastructure, to speed-up the evaluation process (as more than 3.5 years worth of real time experiments are performed). Consequently, we developed and evaluated a network simulator to mimic the conditions that players generally experience on the Internet (Section 4.1.1). This network simulator operates at the message level since most game messages are small and fit within one packet. We evaluated the implemented network simulator (described in Section 4.1.1), and compared it to an emulator that operates at the packet level. Based on these evaluations, we infer that the network simulator introduces negligible error rates compared to a real network or a packet–level emulator.

To evaluate MCCA, the experiments presented in this thesis simulate Quake III, a Peer-to-Peer (P2P) online multiplayer game, in an environment where the game traffic competes with VoIP traffic and file transfers. VoIP traffic is selected as a workload for evaluation, primary due to its strict network requirements and the fact that a large number of online multiplayer games support VoIP as a communication mean. Furthermore, file transfer is selected for its bandwidth intensive workload that helps evaluating the effects of workload prioritization and classification.

4.1.1 Network Simulator

The primary factor affecting the feasibility of online multiplayer games is the upload capacity of each player [4], as the backbone infrastructure is generally well over provisioned for online multiplayer games. Based on this assumption, we developed and

evaluated a network simulator for point-to-point communication among players. The network simulator uses a specification of network characteristics (capacity and latency) to mimic the conditions that players generally experience on the Internet. Additionally, the simulator mimics packet loss and corruption (packet corruption is modeled as a loss). The following discusses the details of the network simulation and its evaluation .

Network Simulation

The developed network simulator works at the message level, and operates in rounds of 10μs intervals. Considering that all experiment messages (Section 4.2) are smaller than maximum IP packet size, we believe simulating at the message level is reasonable. The network simulator models the outgoing and incoming network links of all players, and manages the flow of messages based on each player's network capacity and link's configuration. At each round, the simulator calculates the time of a message leaving the outgoing link, and frees the buffer accordingly; it then applies possible loss, corruption, and delay to the message, calculates the arrival time, and, if it is the case, places the message in the receiver's queue.

The primary factor affecting the feasibility of online multiplayer games is the upload capacity at each node [4]. Bharambe et al. [4] conclude that access link bandwidth for broadband residential clients in the U.S. is modeled by a log-normal distribution (with mean of 1Mbps and a range of 256Kbps to 5Mbps), a result we use in our simulations. Since the access link capacities are generally much smaller than core links' capacity, the simulator does not model contention and cross traffic in the network core.

Network Simulator Evaluation

We evaluated the simulator by comparing the observed transfer rate and communication latency of a network traffic generator, which uses the network simulator, to the one that uses the network emulator (NetEm [44]), under the same conditions. The network emulator used for these experiments, NetEm, allows filtering of any two (IP, Port) combinations and applies network delay, loss, and rate limit. Overall, NetEm allows emulation of the same network conditions as real networks, or the presented network simulator.

The network traffic generator, mentioned above, has a sender and a receiver entities. The sender attempts to send a fixed sized packets at fixed intervals to the receiver, where all packets are marked with the time of send. The receiver reads packets and measures the observed bandwidth and latency to later evaluate the network simulator's error compare to the network emulator.

Measuring the latency for the network traffic generator running on the network simulator is accurate since it uses a virtual clock/time. On the other hand, for the network traffic generator using NetEm, the sender and receiver are placed on the same device in order to accurately measure the latency and avoid clock inconsistencies among different devices.

To measure the network simulator's relative error at different conditions, we performed experiments using different combinations of packet size, time interval, and network upload capacity (bandwidth). Each experiment is performed multiple times in order to achieve high confidence in every data point. Table 4.1 presents combinations of experiments performed. For each experiment, the sender sends packets until 5GB worth of data is successfully transfered to the receiver.

Parameter	Values
Network Capacity (Bandwidth)	256Kbps, 512Kbps, 1Mbps, 2Mbps, 4Mbps
Packet Size	64B, 128B, 256B, 512B, 1024B
Packet Interval	0ms, 2.5ms, 5ms, 10ms, 20ms, 40ms

Table 4.1: Experiment Configurations for Network Simulator Evaluation

Equation 4.1 presents the formula used to calculate the network simulator's relative error (absolute value of the difference of the two values over the network emulator's value):

$$RelativeError = abs\left(\frac{S-E}{E}\right) \tag{4.1}$$

where

S : Value from the experiment using the network simulator

E : Value from the experiment using the network emulator

In this thesis the network simulator's error refers to the simulator's bandwidth error, as it is consistently higher than the simulator's error in transfer latency. Figure 4.2 presents the network simulator's error histogram for all performed experiments. More than 85% of the experiments show less than 2% margin of error, while less than few percentages of experiments show errors as high as 8%. Overall the average error for all experiments is 0.9% with standard deviation of 1.20%, while the minimum observed error is 0.13% and the maximum observed error is 6.88%.

To better understand the simulator's strengths and weaknesses, Figures 4.3, and 4.4 present the average error distribution and 95% confidence interval with respect to the network upload capacity (Bandwidth) and packet interval respectively.

Results show that the network simulator's error increases as the network capacity increases, and it rapidly decreases as the packet interval is increased. The highest errors are observed when the network traffic generator attempts to send messages back

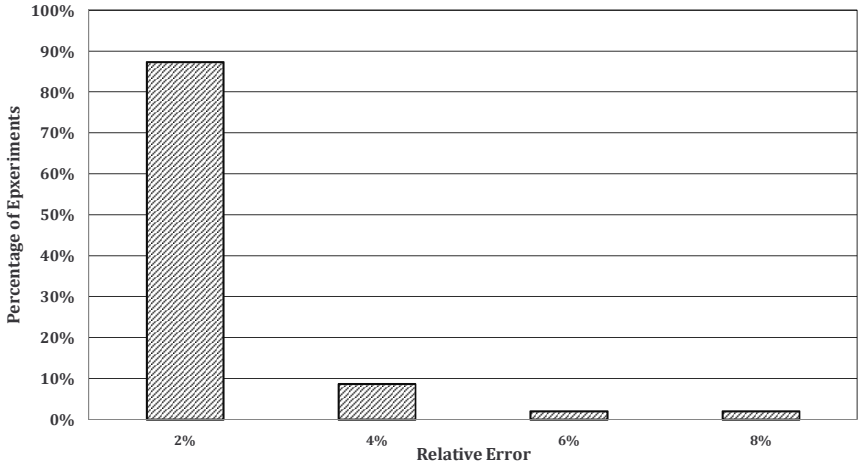


Figure 4.2: Network Simulator's Relative Error Histogram

to back (at packet interval of 0ms). Such workload generates a high processing overhead compared to network simulator, where no processing, IO, or kernel time is modeled. This processing overhead is increased as the number of I/O operations (socket send and receive), in short amount of time, is increased. Such overhead affects the overall observed bandwidth, and is higher at larger network capacities, where more packets are sent faster.

It is unrealistic to assume that any game would attempt to send packets back–to–back, as games have many other components such as graphics, audio, AI, physics and more. Due to the periodical behavior of the workloads (Section 4.2) used in experiments (presented in Chapter 5) and based on the network simulator's evaluation, the packet interval, at the worst case scenario, is anticipated to be between 2.5ms to 5ms.

Online multiplayer games are CPU intensive applications requesting a large number of IO operations frequently. Considering that the network simulator does not simulate processing overheads, the following experiment is performed to study the effects of a

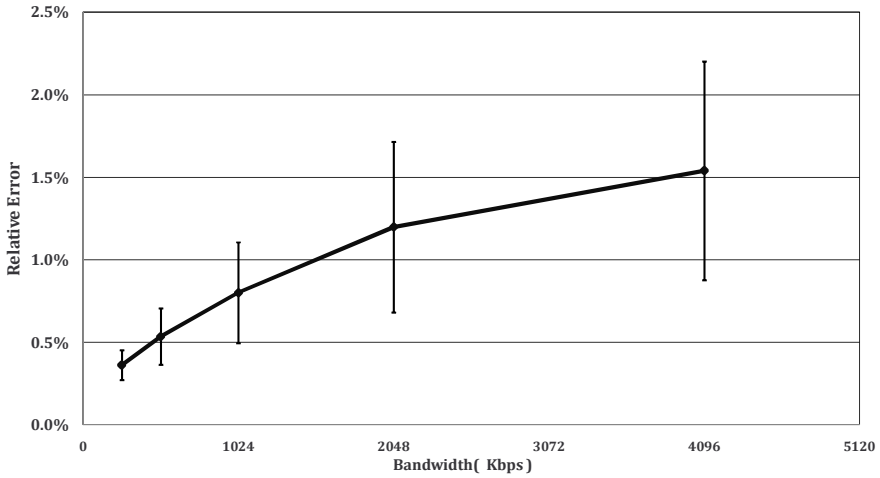


Figure 4.3: Network Simulator's Relative Error with Respect to Network Capacity (Bandwidth)

CPU intensive application on the network simulator's error. Firstly, a CPU intensive application is developed to run as a separate process (unrelated to the network traffic generator). Secondly, the CPU intensive application is run on all available cores in the background, while the experiment is performed. Experiment with these conditions are performed for 2.5ms packet intervals, showing an average error of 1.9% (about 71% increase compare to the normal scenario), still resulting in relatively low and acceptable error rates.

4.2 Workload Generators

To evaluate the communication architecture, its prioritization, classification, and the affects of optimized group communication as well as benefits of quality aggregation, the following workload generators were developed:

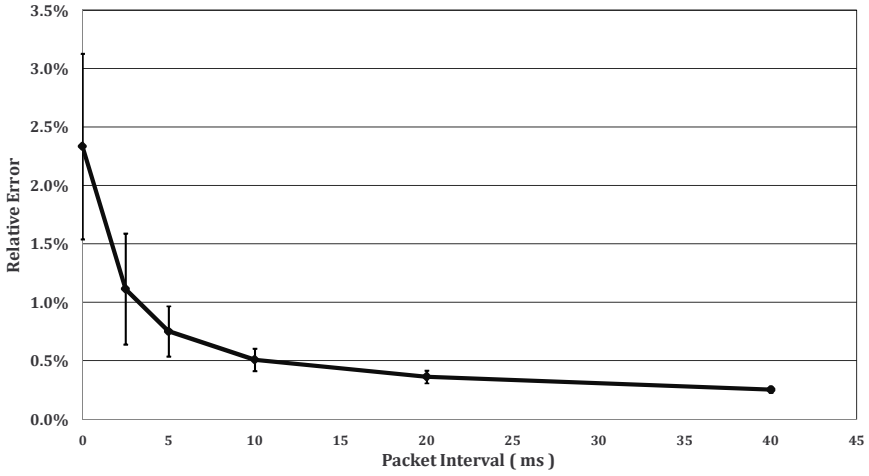


Figure 4.4: Network Simulator's Relative Error with Respect to Packet Interval

- Game Updates
- VoIP Traffic
- File Transfer
- System Messages

MCCA is targeted for online multiplayer games with multiple ongoing workloads with different behaviors, characteristics, and requirements. Consequently, aside from game updates, VoIP traffic and file transfer workloads are used for evaluation purposes. These are among the most common types of workloads for online multiplayer games, and are also different in behavior and requirements. The fourth type of workload is system messages, as it is the byproduct of managing the distributed state of players. This section discusses workload generation for each workload: game updates (Section 4.2.1), VoIP traffic (Section 4.2.2), file transfer (Section 4.2.3), and system messages (Section 4.2.4).

4.2.1 Generating Game Traffic

The game workload used in all experiments of this thesis, is the simulation of Quake III workload (as a P2P online multiplayer game) based on the model developed by Bharambe et al. [4]. Quake III presents a typical online multiplayer game, where small game updates are broadcasted regularly, and delta-encoding is used to reduce the generated network traffic. Quake III is a First Person Shooter (FPS) game with tighter latency requirements compared to other game genres (such as Real-Time Strategy (RTS) games). The following describes the delta encoding model for Quake III:

1. The game periodically sends updates at 50ms intervals.
2. The game sends a one-frame delta encoded update with mean size of 32 bytes, if the last message is acknowledged by the target player.
3. The game sends a two-frame delta encoded update with a mean size of 36 bytes, if the second last message is acknowledged by the target player.
4. The game sends a complete update (raw data) without any delta encoding, which is 196 bytes long, if none of the last two messages are acknowledged.

4.2.2 Generating Voice-Over-IP (VoIP) Traffic

Traditionally VoIP sources can be modeled as an on-off Markov process. The alternating periods of activity and silence are exponentially distributed with average durations of μ and λ respectively [16]. When the source is in the "talk" state, fixed-size packets are generated at a constant interval, while no packets are transmitted when the source is in the "silent" state. The size of the packet and the rate at which the packets are sent depends on the voice codec and compression scheme.

Source Type	Configuration Parameter
Talkative	$\mu = 5$s $\lambda = 10$s
Quiet	$\mu = 5$s $\lambda = 20$s

Table 4.2: Talkative and Quiet VoIP Source Configurations

Experiments in this thesis are based on Padungkrit et al.'s model [52] for VoIP traffic in real networks. This model adds one extra "short silence" state demonstrated in Figure 4.5. Once the source is in the "talk" state, it goes back and forth between the "talk" and "short silence" states. In addition, the VoIP traffic generator developed use the codec size, frequency, and state probabilities described by Padungkrit et al's model [52]. Consequently, VoIP messages are 78 bytes long, generated every 20ms during the "talk" state. The probability of changing state from "talk" state to "short silence" state is 32% every half a second, and the probability of returning to the "talk" state from the "short silence" state is 71% every half a second.

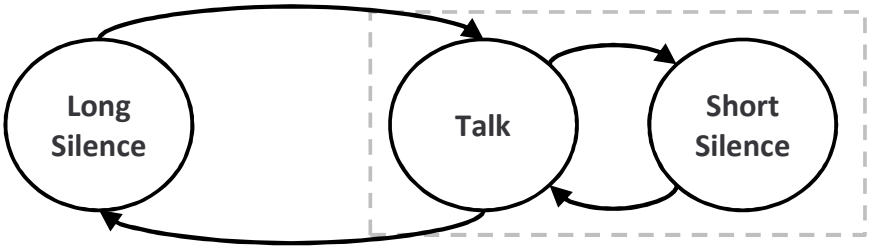


Figure 4.5: VoIP State Machine

The mean talk (μ) and mean silence (λ) durations are varied in experiments to simulate a talkative and a relatively quiet VoIP sources (Table 4.2 presents the configuration parameters).

4.2.3 File Transfer

Experiments involving file transfer (file workload) assume that the game sends a 500KB file to other players every five minutes. The file size is selected based on a normal distribution with mean of 500KB and standard deviation of 32KB. Games usually have their required files and assets loaded into memory; thus, DVD or storage device bandwidth limitations are not modeled. Furthermore, files are split into fixed sized chunks that are sent frequently until the file is fully transmitted. The file chunk size is varied in experiments to evaluate their affect on the perceived quality of VoIP and game updates.

4.2.4 System Messages

System workload is the byproduct of the messages transmitted among players to manage their distributed state. These messages are mainly consist of:

- **Pings:** Sent every few seconds to all other players as a mechanism to detect for premature disconnections. In addition pings carry useful information such as the amount of available forwarding bandwidth.

- **Control Messages:** Messages mainly used to manage the multicast tree state, such as registering or unregistering forwardees, and/or accepting or denying registration requests. Such messages are more frequent upon a player joining or leaving, and they are less frequent once players' trees are built and adjusted.

Chapter 5

Experiments and Results

This chapter presents details and results of experiments performed for evaluating MCCA and its components. Each experiment is performed multiple times, each time with a different random seed number, in order to achieve high confidence interval in the results; all figures present the 95% confidence interval. We anticipate satisfactory results with a small number of players (as player's upload capacity is plentiful), and unacceptable results with a large numbers of players (as network is severely under-provisioned). In order for the techniques explored in this thesis to have an effect, experiments focus on the region that the network is not severely over– or under–provisioned. Consequently, the number of players are steadily increased for every experiment configuration until unacceptable quality levels are achieved.

To properly evaluate MCCA as a communication architecture; number of players, characteristics of the deployment platform (loss rate, bandwidth, and latency), as well as the number and definition of game workloads are varied in experiments. In addition, different group communication mechanisms, and game-level optimizations (based on distributed quality aggregation, and flow specific reports) are presented and evaluated. The following presents the types of experiments we performed to evaluate MCCA:

1. **Various Workload Generator Combinations:** To study effects of VoIP and file transfer on online multiplayer games, and to evaluate MCCA's support for

multiple ongoing network traffics

2. **Network Traffic Class Definitions:** To evaluate MCCA's support for network traffic classification

3. **Group Communication:** To evaluate MCCA's support for efficient message dissemination

4. **Game-Level Optimizations:** To evaluate MCCA's distributed quality aggregation, and to demonstrate the potential for new types of game-level optimizations using MCCA

5. **Real-Time Changes in the Number of Players and Network Conditions:** To demonstrate online multiplayer game's ability to adapt to variable number of players and network conditions using MCCA

6. **Various Network Conditions:** To evaluate MCCA at other network conditions with higher loss rates, and lower network capacities

The types of experiments presented above are designed to: 1) individually evaluate each of MCCA's components, and 2) evaluate the MCCA as a whole. As a result, some experiments study the effects of group communication, while others evaluate network traffic classification and/or distributed quality aggregation through game-level optimizations.

The following section describes the experiment setup (Section 5.1), including details of the deployment environment (Section 5.1.1), and certain game-level optimizations (Section 5.1.3). Following which, the evaluation metric for each type of workload (Section 5.2), and the experiment naming convention (Section 5.3) are presented. This chapter then presents experiments with class definitions (Section 5.4), workload variations

(Section 5.5), real-time player and deployment variations (Section 5.6), and network conditions (Section 5.7). In addition, the study of quality variability among players (Section 5.8) are presented. Finally, MCCA's comparison with the optimal and idealized solution (Section 5.9), and the summery of the experiment results (Section 5.10) are presented.

5.1 Experiment Setup

To evaluate MCCA, the experiments presented in this chapter use different combinations of workloads described in Section 4.2. In addition, all experiments use MCCA (described in Chapter 3) as a communication layer. Finally, all experiments are performed using the network simulator described in Section 4.1.1.

The following sections present the details of the simulated deployment environment for experiments (Section 5.1.1), the details of the configuration parameters for the network traffic workloads (Section 5.1.2), and the game-level optimizations that are based on MCCA and its distributed quality aggregation (Section 5.1.3).

5.1.1 Simulated Deployment Environment

The network simulator (described in Section 4.1.1) is used as the communication infrastructure for all experiments. Following sections present the details for selecting every player's network capacity, as well as inter node latencies and loss rates. Players' upload capacity and latencies are randomly selected based on distributions found by Bharambe et al. [4]. Every experiment is performed multiple times using a different random seed number until high confidence intervals are achieved.

The default network parameters for experiments are based on the Table 5.1, with

Network Characteristic	Values
Bandwidth	μ = 1Mbps Range = [256Kbps,5Mbps],[768Kbps,3.5Mbps]
Corruption	0.1%
Latency	μ = 30ms, and σ = 30ms
Loss	0.1%, 1%, 5%

Table 5.1: Deployment Variations

the network capacity ranging from 768Kbps to 3.5Mbps, and network loss of 0.1%. Experiments with other network conditions are described in Section 5.7.

Bandwidth

The player's network capacities are randomly selected based on the log-normal distribution described in Section 4.1.1. Furthermore, by changing the variance in the model but keeping the mean the same, two network capacity distributions, one varying from 256Kbps to 5Mbps (Low Bandwidth), and another one varying from 768Kbps to 3.5Mbps (High Bandwidth), are used in the experiments.

Loss, Latency, and Corruption

To reduce inter-node latency and to combine players speaking the same language, online multiplayer games use matchmaking services that group (in the same game instance) players based on their geographical location [9]. Using data from a real game, Bharambe et al. [4] note that the mean, median, and standard deviation for inter-player RTT is 81ms, 64ms, and 63ms respectively. We use the same values for our experiments. Additionally, we conservatively fix the packet corruption rate at 0.1% and vary the loss rate from 0.1% to 5%. Table 5.1 shows the various deployment configurations under

which experiments are peformed.

5.1.2 Network Traffic Workloads

Workload generators (described in Section 4.2) are used for evaluation, where the VoIP workload generator (described in Section 4.2.2) by default simulates relatively quiet sources (specified in Table 4.2), and the file transfer workload (described in Section 4.2.3) uses a default file chunk–size of 256 bytes. Section 5.5 evaluates MCCA using other workload variations such as talkative VoIP sources and a larger file chunk-size.

Observed game and VoIP qualities are estimated based on techniques described in Section 3.3. To estimate the observed game quality, the perfectly tolerated latency (α), and the maximum tolerated latency (β) of respectively 100ms and 200ms are used (based on the result of a user study by Matthias Dick et al. [19] for Quake III).

5.1.3 Game-Level Optimizations

To evaluate MCCA's flexibility and its ability to support game-level optimizations, we developed the following game-level optimizations: 1) Optimistic Game, 2) Adaptive Game Update Interval, and 3) VoIP Optimization.

Optimistic Game

The First Person Shooter (FPS) game, described in Section 4.2.1, uses delta-encoding to decrease its generated network traffic. The presented model is a pessimistic model, which assumes that unacknowledged messages are lost. Due to network latencies and the network traffic, as the number of players grows, acknowledgments may be received too late resulting in an unnecessary increase in the network traffic. This increase is due to the game sending complete updates as opposed to much smaller one– or two–frame

delta encoded updates. This thesis explores the optimistic game model, where messages are assumed to be received, and a player only sends a complete update once it detects a two consecutive message loss for a receiving player (i.e., two consecutive message loss in the game "flow").

In this approach a receiving player missing two or more consecutive updates sends a request for a complete update, and would have an invalid state until it is received. To account for such conditions, updates received during the invalid game state contribute zero value to the estimated game quality. Finally, to better understand the game status, the percentage of players' invalid–state time are tracked for optimistic Quake III experiments.

Adaptive Game Update Interval

The presented peer-to-peer FPS generates a game update every 50ms, which is sent to all other players. Consequently, increasing the number of players results in a much higher network traffic compared to experiments with lower number of players, finally reaching conditions where 1) it takes too long to propagate a game update (updates lose their value due to large delay) or 2) updates are generated faster than they are transfered. Such circumstances result in a low game quality and are not acceptable.

This thesis proposes a variable game update interval, based on the aggregated game quality, as opposed to the fixed game update interval of 50ms. At low game qualities, a game could increase its game update interval, which reduces the amount of generated network traffic, to achieve higher sustainable game qualities. Such approach introduces a delay penalty for increasing the game update interval, as the time between an event occurrence and its corresponding game update transfer increases (Figure 3.3). Increasing the game update interval results in a lower rate of game updates, and reduces

network traffic. Such optimization is feasible in online multiplayer games, considering the game's control of the game update generation and interpretation.

This optimization can slightly reduce the game quality due to the extra delay; however, experiments show that the quality gained due to lower network traffic is much higher than the quality penalty due to the extra delay.

In experiments using the proposed adaptive game update interval, the game update interval is increased once the aggregated quality falls below the *low* quality threshold. Upon such event, the game update interval is increased periodically until the estimated quality reaches above the *okay* quality threshold. Experiments in this thesis use the *low* quality threshold of 8.0, and *okay* quality threshold of 9.0. In addition, to avoid unrealistic game update intervals, the maximum allowed update interval is set to 100ms (which in combination with the average network latency results in acceptable end-to-end delay, based on user studies of Matthias Dick et al. [19]). In addition, considering that the game update interval could be increased to higher levels than required, the game slowly reduces the update interval while the game quality is not significantly reduced. Finally, to provide a realistic model, game update intervals are rounded up to the nearest 5ms.

Aside from increasing the game update interval, online multiplayer games can use other techniques such as Area–of–Interest (AOI) and Predictive Contract Mechanisms PCM (described in Section 2.2) in combination with the distributed game quality aggregation for adapting to the number of players, and their network conditions. The following describes how AOI or PCM could implement such techniques to improve the perceived quality, and support higher number of players:

- **AOI:** Recognizing the game quality degradation, a game can further reduce the

game update frequency for players outside the area of interest to reduce the generated network traffic.

- **PCM:** PCM is a technique where updates are sent once a prediction error is higher than a certain threshold. Recognizing game quality degradation, using the MCCA's distributed quality aggregation, a game can increase the prediction error threshold to reduce the number of generated game updates (network traffic).

VoIP Optimization

The aggregated VoIP quality provides a good perception on effectiveness of the VoIP communication. VoIP at low qualities is essentially a network traffic that does not provide any useful functionality to the game or the player. A game recognizing low VoIP qualities can notify the player, and disable the VoIP communication, in order to use the network resources for other useful purposes.

Low VoIP qualities could be temporary and due to cross network traffic (i.e., other applications using the player's network resources). Consequently, a game using such optimization should enable the VoIP communication at some point in future based on player's preference and network conditions. To achieve this behavior, an experiment using VoIP optimization automatically disables VoIP at low qualities for a duration of time measured based on the last observed quality. In this model, lower VoIP qualities result in longer *off* durations.

5.2 Quality Estimation Metrics

The quality of game, VoIP, and file transfer are measured respectively in terms of the game quality estimation (described in Section 3.3.2), R-Factor (described in Sec-

tion 2.4.2), and transfer rate (KBps). This section describes the quality metric, as well as the acceptable and unacceptable quality ranges for each of the mentioned workloads.

5.2.1 Game Traffic

Game quality is estimated based on the model presented in Section 3.3.2, where α and β latencies (of respectively 100ms and 200ms) are chosen based on user studies of Matthias Dick et al. [19]. These user studies also suggest that the average player's idea of "maximum tolerable latency" for Quake III is about 150ms. Assuming a reliable network, and based on this thesis's quality estimation model for Quake III, such end-to-end latency results in an estimated game quality of 5.0.

This thesis assumes that game developers intend to ensure that no player observes a game quality of lower than the average tolerated game quality. Considering that the experiment results in Chapter 5 demonstrate the average quality, experiment results are investigated to find the smallest average game quality that satisfies the above requirement (i.e., the acceptable quality threshold). As a result, smallest average game quality of 7.0 is found to be acceptable, at which the game quality distribution satisfies the above requirement. Section 5.8 presents the details of this investigation.

5.2.2 Voice-Over-IP (VoIP)

The VoIP quality estimation is based on the E-Model presented in Section 2.4.2 that also presents a quality metric shown in Table 2.1. Experiment results in Chapter 5 demonstrate the R-Factor value for VoIP, which based on Table 2.1 represent poor and unacceptable qualities at R-Factor values of 60 or below.

5.2.3 File Transfer

Experiment results in Chapter 5 demonstrate the observed transfer rate (KBps) for experiments with file workload. Estimating the acceptable transfer rate is dependent on the characteristic of its application; however, due to lack of information on file transfer requirements, the observed transfer rate is proposed as a comparison metric.

5.3 Naming Conventions for Experiments

Considering the long description names of performed experiments (e.g., Optimistic game using adaptive game update interval with VoIP and file transfer workloads), and a many number of experiment configurations, the following naming convention is used in figures and tables for labeling experiments. All abbreviation are also listed in Table B.2 and mentioned in their respective sections. The following presents the naming convention rules:

1. 'G' represents the word "**G**ame" (an experiment with game workload)
2. 'V' represents the word "**V**oIP" (an experiment with VoIP workload)
3. 'F' represents the word "**F**ile" (an experiment with file transfer workload)
4. An 'O' before 'G' represents "**O**ptimistic **G**ame" (an experiment with optimistic game workload, described in Section 5.1.3)
5. An 'A' after 'G' represents "**A**ddaptive Game Update Interval" (an experiment using adaptive game update interval, described in Section 5.1.3)
6. An 'O' after 'V' represents "**V**oIP **O**ptimization" (an experiment using VoIP optimization, described in Section 5.1.3)

Different combinations of the mentioned naming convention rules are used to label experiments. For example, "OGAVO" represents an optimistic game using an adaptive game update interval, with VoIP workload that uses VoIP optimization.

5.4 Experiments with Class Definitions

This section presents experiments with various workload combinations and class definitions (Section 5.4.1), as well as experiments evaluating the effects and usefulness of workload window-size (Section 5.4.2), and message redundancy (Section 5.4.3).

5.4.1 Experiments with Workloads and Classification

To evaluate the effectiveness of network traffic classification and prioritization proposed in this thesis, this section presents experiments with different workload combinations. In addition, experiments involving the proposed group communications are presented to validate the intuition that class specific group communication benefits online multiplayer games. Finally, experiments with various game-level optimizations are presented to evaluate the usefulness of distributed quality aggregation for online multiplayer games.

To understand MCCA's behavior and its potential benefits, variations of each workload combination are analyzed in separate groups, where effects of network traffic classification, efficient group communication, and game–level optimizations are evaluated. The following presents experiments with the following workloads:

1. Game only workload

2. Game with VoIP workload

3. Game with VoIP and file transfer workloads

Game Only Workload

To evaluate the new game-level optimizations (only possible due to MCCA), and to understand the quality and behavior of the base scenario, this section presents the experiments with the game only workload. Experiments with game only workload include the following combinations of game-level optimizations (described in Section 5.1.3):

1. Game (G)
2. Game with adaptive game update interval (GA)
3. Optimistic game (OG)
4. Optimistic game with adaptive game update interval (OGA)

Figure 5.1 presents the result of above experiments and includes the 95% confidence interval, where every experiment (resulting in a data point) is performed many times with a different seed number, until a 95% confidence interval of lower than 5% is achieved. The figure presents the estimated observed game quality, which is calculated based on the techniques described in Section 3.3.2. As the game quality is correlated to the game update time and the invalid game state time, Figure 5.1 presents these information for their corresponding experiments.

The OGA[5] configuration provides the highest game quality (Figure 5.1a), while it results in lower game update intervals compare to GA configuration (Figure 5.1b), and lower invalid game state durations compared to OG configuration (Figure 5.1c). In

[5]Section 5.3 and Table B.2 in Appendix B describe the experiment naming convention

general, experiments that include a game-level optimization perform much better than the simple game at higher number of players.

To correctly understand the result of experiments with adaptive game update interval, it is important to fully understand its behavior (described in Section 5.1.3). The model for changing the game update interval does not produce the most optimal solution (i.e., achieving the highest game quality possible). This model attempts to find the smallest game update interval at which the game can operate properly. Additionally, in this model, the game update interval is periodically increased when the game quality is in a specific game quality range. Consequently, as a result of the presented model's behavior, the game quality for GA experiment increases between 26 players to 38 players.

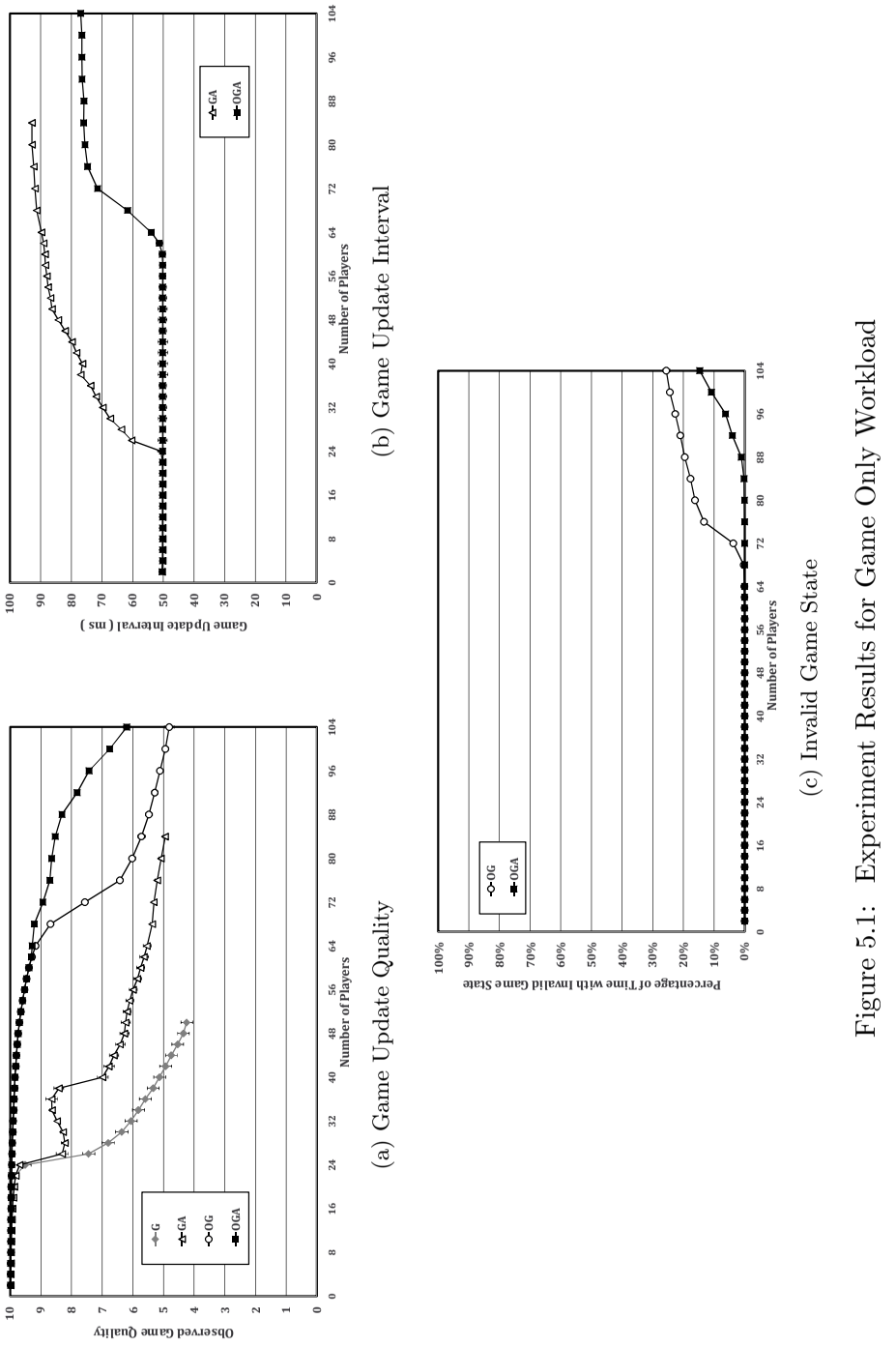


Figure 5.1: Experiment Results for Game Only Workload

Game with VoIP Workload

To study the effects of VoIP on online multiplayer games, and also evaluate MCCA for supporting a game with VoIP communication, this section focuses on experiments with game and VoIP workloads. Experiments with game and VoIP workloads include the following combinations of group communication (described in Section 3.2) and game-level optimizations (described in Section 5.1.3):

1. Game with VoIP, without any prioritization, classification, or efficient group communication (labeled GV Single Class)

2. Game with VoIP, using prioritization and classification, without any efficient group communication (GV Using Broadcast)

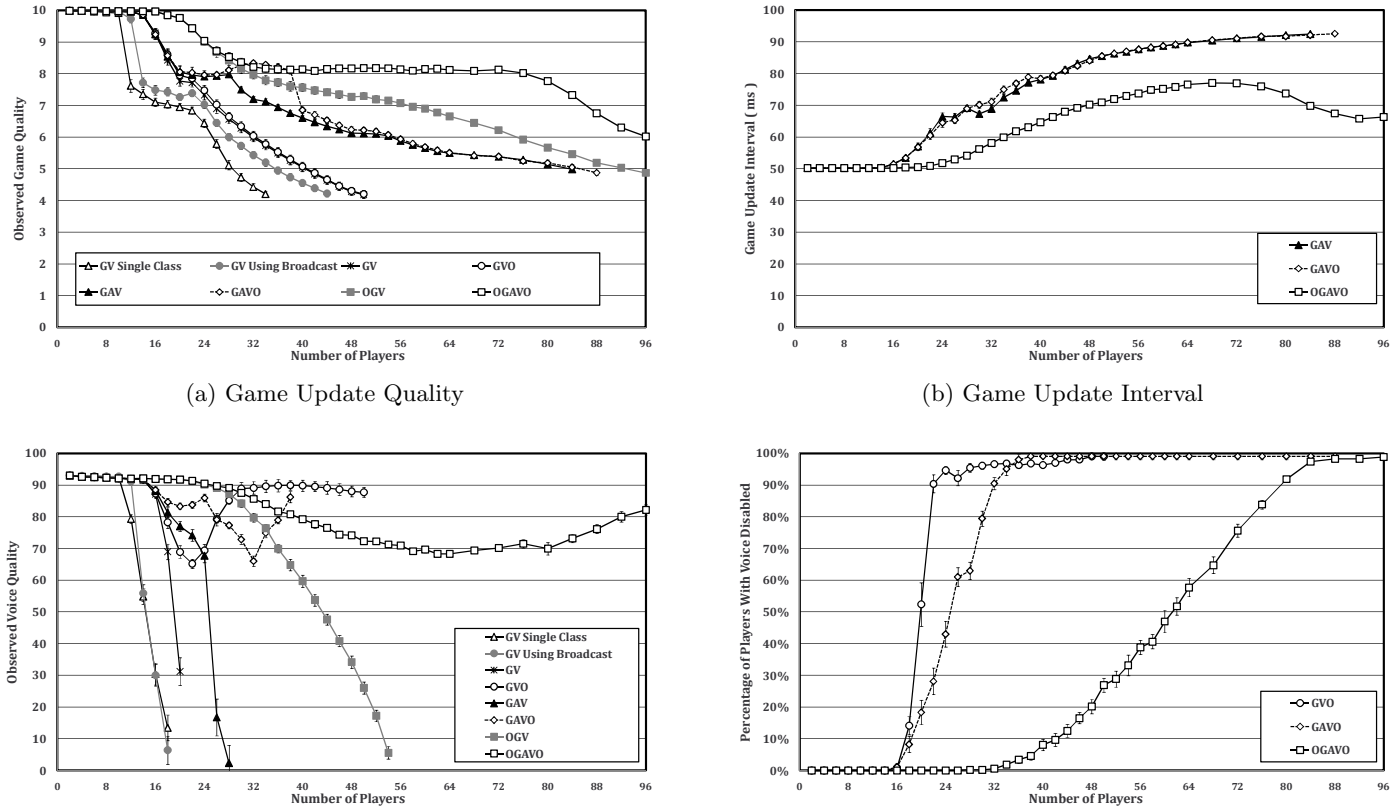
3. Game with VoIP, using prioritization, classification, and a two-level multicast tree, optimized for bandwidth, for its VoIP traffic (GV)

4. The above configuration (#3), using different game-level optimizations (GVO, GAV, GAVO, OGV, and OGAVO).

Figure 5.2 presents the result of these experiments, including the estimated game and VoIP qualities, as game and VoIP are competing traffics that affect one another. In addition, Figure 5.2b and Figure 5.2d respectively present the game update interval and the VoIP's activity (i.e., on/off) as they significantly affect the game quality.

Results in Figure 5.2a show that prioritization on its own can slightly improve game quality ("GV Using Broadcast" vs. "GV Single Class"), while in combination with efficient group communication for VoIP it can significantly improve both VoIP and game qualities ("GV" vs. "GV Single Class"). In addition, we infer that the VoIP optimization alone has little impact ("GVO" vs. "GV"), as game updates alone

saturate all available network resources. On the other hand, VoIP optimization in combination with adaptive game update interval has a much higher impact at higher number of players ("GAVO" vs. "GAV"). Finally, and as expected, the optimistic game that uses all the presented game–level optimizations outperforms all other experiments, provides higher game and VoIP qualities, and operates at lower game update intervals (Figure 5.2b) with more players having their VoIP enabled (Figure 5.2d).



(a) Game Update Quality

(b) Game Update Interval

(c) VoIP Quality

(d) Disabled VoIP Ratio

Figure 5.2: Experiment Results for Game with VoIP Workload: Figures on the left present the game and VoIP qualities. Figures on the right present the changes in game update interval, and disabled VoIP ratio, cause by their corresponding game-level optimizations; as a result, figures on the right only present the values for experiments with such game-level optimizations.

Game with VoIP and File Transfer Traffic

To study the effects of multiple types of network traffics on online multiplayer games, and to evaluate MCCA's ability to support such conditions, this section presents the experiments using game with VoIP and file workloads. File transfer requires in-order and reliable communication, and is network intensive. To support file transfer without network traffic prioritization and classification, all flows of network traffic would need to be treated orderly and reliably. Such a scenario causes unnecessary overhead, and lowers the observed QoS; however, using MCCA, the file transfers can be serviced in isolation from other network traffic flows.

Experiments with game, VoIP, and file workloads include the following combinations of group communication and game–level optimizations:

1. Game with VoIP and file transfer workloads, without any efficient group communication ("GVF Using Broadcast")

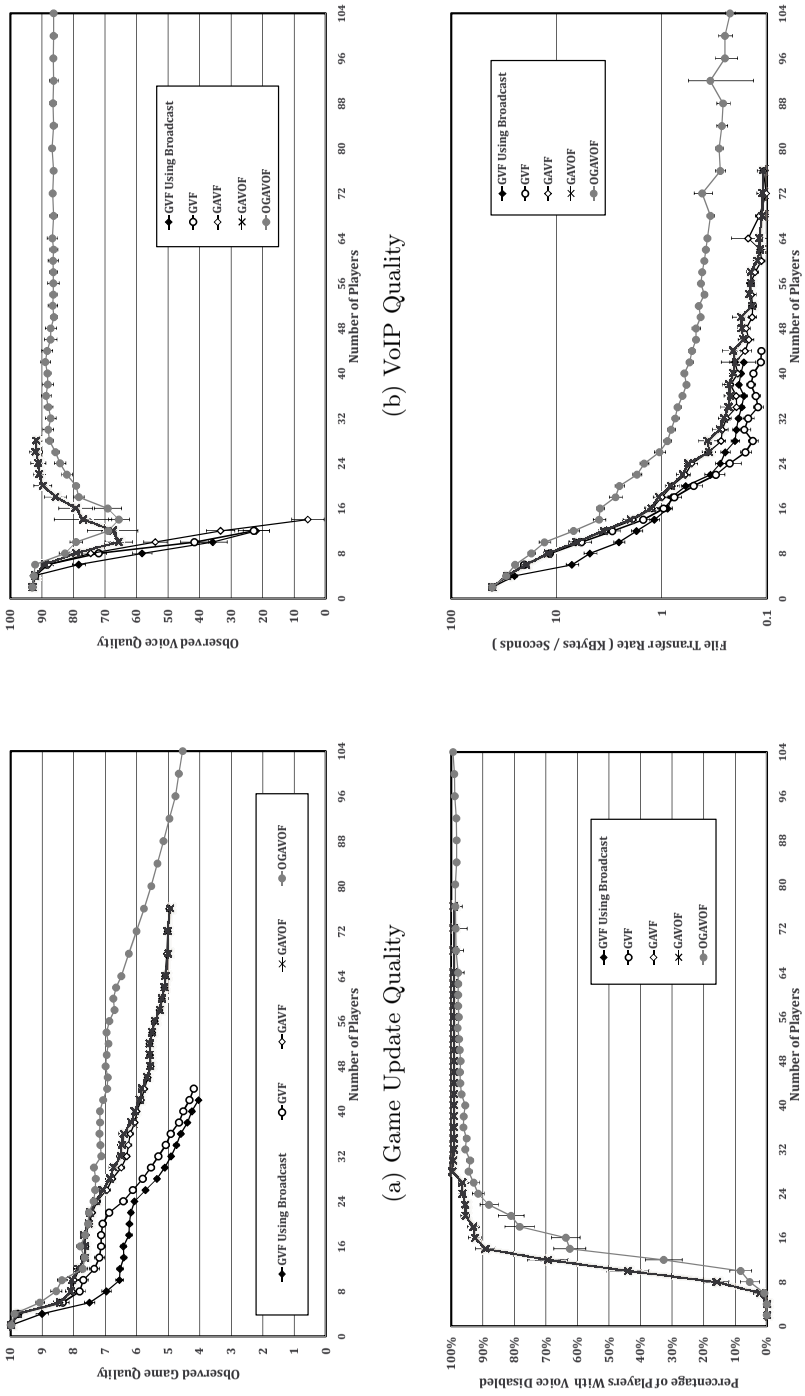
2. Game with VoIP and file workloads, using a two level multicast tree, optimized for bandwidth, for its VoIP and file transfer workloads (GVF)

3. The above configuration (#2), using various combinations of game-level optimizations (GAVF, GAVOF, and OGAVOF)

Figure 5.3 presents the results of the above experiments. Results include the estimated game and VoIP qualities, and the observed bandwidth for file transfers. In addition, Figure 5.3 presents the average percentages of players with disabled VoIP because of its impact on other workloads.

Results show that a network intensive workload such as file transfer can significantly reduce game and VoIP qualities, while it also reaches low transfer rates at higher number

of players (Figure 5.3d). Such bandwidth consuming network traffic quickly results in unacceptable game qualities as the number of players increased, while it also introduces latencies at low numbers of players degrading the game and VoIP qualities. Similarly to experiments with game and VoIP workloads, efficient group communication improves the quality (GVF vs. "GVF Using Broadcast"). The optimistic game using both game and VoIP optimizations presents the best solution (shown in Figure 5.3), as it manages to maintain the game quality while player's VoIP communications are disabled as the number of players increased, and file transfer rates are significantly reduced.



(a) Game Update Quality

(b) VoIP Quality

(c) Disabled VoIP Ratio

(d) File Transfer Rate

Figure 5.3: Experiment Results for Game with VoIP and File Transfer Workloads

5.4.2 Experiments with Workload Window-Size

To evaluate effects of the window-size for classes of network traffic, the window-size for VoIP and file workloads are varied in the following experiments:

1. Experiment with the default window-size of half a second for VoIP (message window-size of 25, considering VoIP messages are generated every 20ms) is compared to another experiment with window-size of one second for VoIP (message window-size of 50).

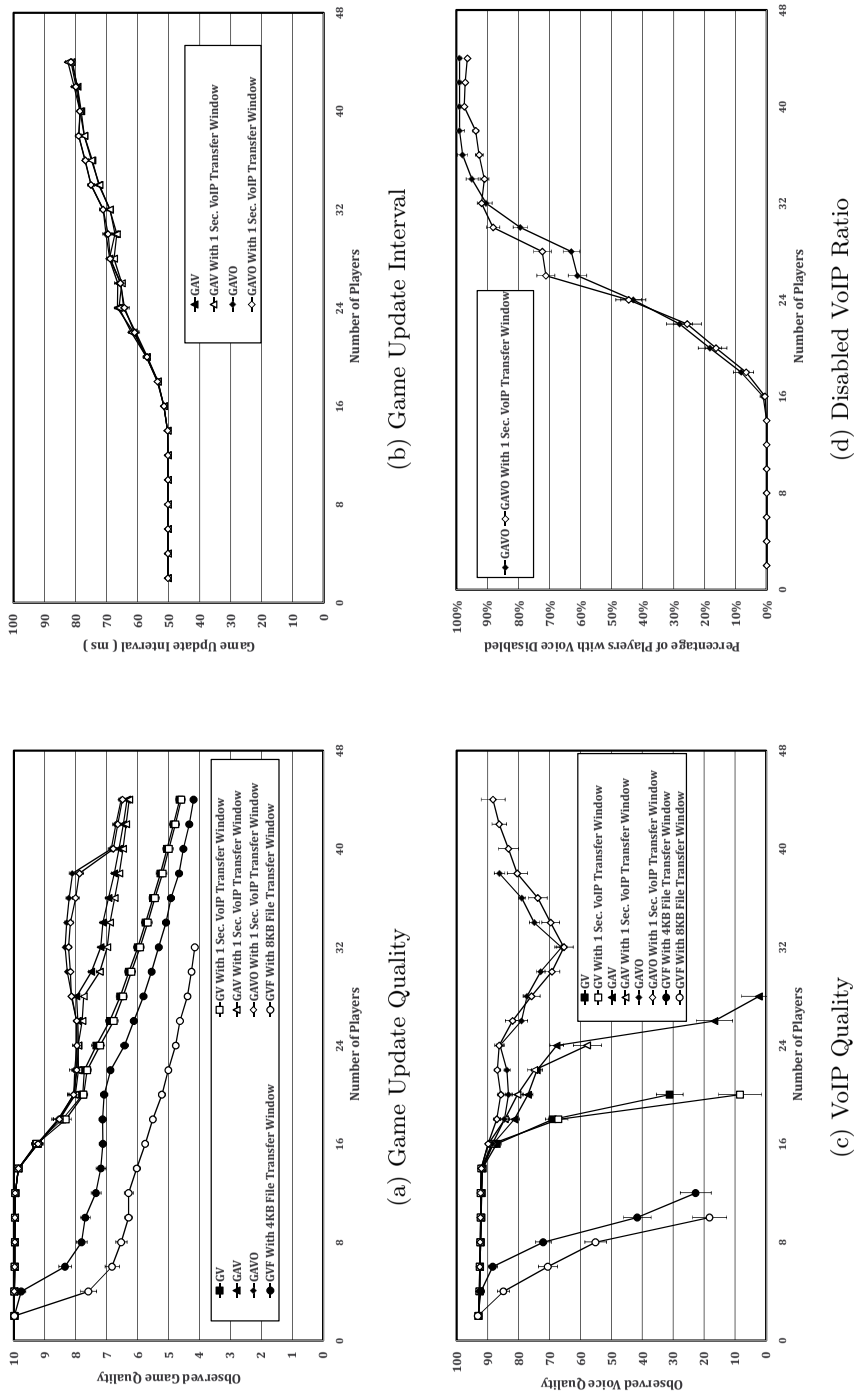
2. Experiment with the default window-size of 4KB for file transfer (message window-size of 16, considering the file chunk–size of 256B) is compared to the experiment with window-size of 8KB for file transfer (message window-size of 32).

Figure 5.4 presents the result of these experiments. Results show that a larger window-size for VoIP slightly degrades the game quality (Figure 5.4a). In addition, results show that the VoIP quality is also degraded in the experiment without game–level optimizations (Figure 5.4c), and is slightly improved in other scenarios (GAV, and GAVO). Furthermore, results show that the game update interval, and the percentage of players with disabled VoIP are higher in experiments with one second window–size (Figure 5.4b, 5.4d).

Results of experiments with variations of file workload's window-size show that both game and VoIP qualities are significantly degraded when a larger (8KB) file transfer window-size (Figure 5.4a, 5.4c) is used. Results also show that the file transfer rate is higher for the experiment with 8KB window-size at two players; however, same experiment at higher number of players result in lower transfer rates compared to the experiment with 4KB window-size (Figure 5.4e). Such behavior is due to the larger file transfer window-size's negative effect on game updates' and acknowledgments' transfer

latencies. Higher latencies result in larger game updates, as the game sends full frame updates instead of one- or two-frame delta encoded updates. Consequently, the extra higher priority traffic reduces the file transfer rate.

Results of experiments presented in this section demonstrate the benefits of correctly specifying the transfer rate (i.e., window-size in MCCA) for each class of network traffic. In addition, experiment results validate the intuition that specifying the transfer rate is essential for defining a class of network traffic, as unnecessarily large window-sizes result in lower estimated game qualities.



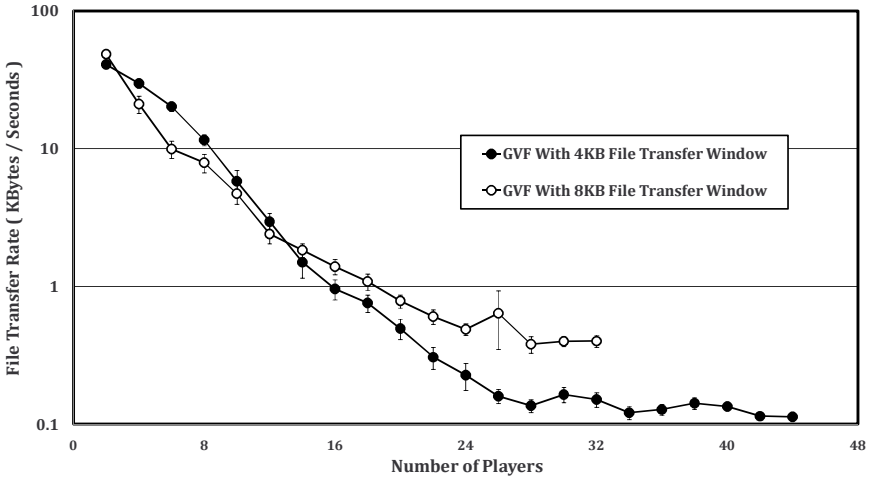
(a) Game Update Quality

(b) Game Update Interval

(c) VoIP Quality

(d) Disabled VoIP Ratio

Figure 5.4: Results of Experiments with Window-Size



(e) File Transfer Rate

Figure 5.4: Results of Experiments with Window-Size (cont'd)

5.4.3 Experiments with Redundancy

To understand effects of the redundancy level and the redundancy interval, we performed experiments using the game workload only, without any game level optimizations, using different combinations of redundancy level and interval. Figure 5.5 presents the results of these experiments. The results confirm that the redundancy overhead significantly degrades the game quality as the number of players increases. In addition, results validate the intuition that message redundancy improves the game quality at lower number of players with higher network loss rates. Finally, results show that higher redundancy intervals can slightly decrease the overhead cost and improve quality.

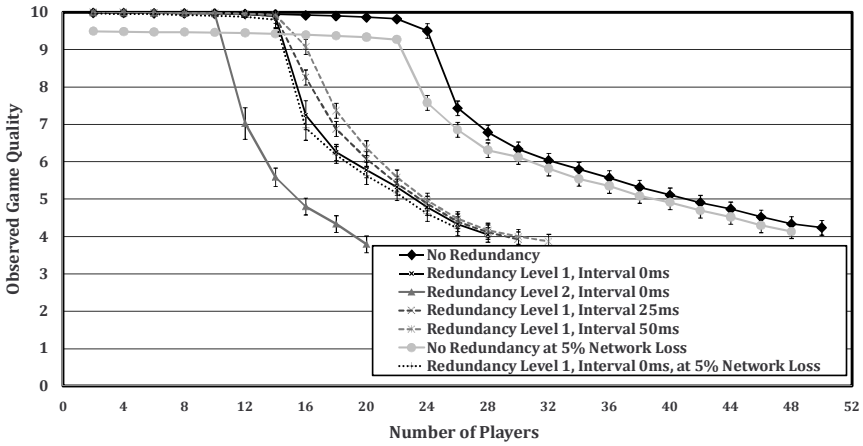


Figure 5.5: Result of Experiments with Game Redundancy

5.4.4 Summary

Experiments using MCCA consistently result in higher estimated game and VoIP qualities, and at the same time they result in higher file transfer rates. Results of experiments with VoIP and file transfers, presented in Section 5.4.1, validated the intuition that network traffic classification can reduce the generated network traffic and enable a game to efficiently use player's network resources. In addition, results show that experiments using flow specific group communication for VoIP and file transfers result in higher estimated qualities. Furthermore, experiments that use game-level optimizations (described in Section 5.1.3) result in higher game and VoIP qualities. These results demonstrate the benefits of distributed quality aggregation, and flow specific network quality estimation for online multiplayer games, as they enable new types of game-level optimizations.

Experiments in Section 5.4.2 demonstrate the benefits of correctly specifying the

window-size for each class of network traffic. Results show that the window-size that is selected based on the network traffic's requirements and behavior can reduce the generated network traffic without effecting the network traffic's estimated QoS. Finally, experiments in Section 5.4.3 study effects of the redundancy level and the redundancy interval. As expected, results show that redundancy can reduce the observed loss rate and the communication latency (by avoiding retransmissions), while it generates higher network traffics. As a result, we conclude that specifying each class of network traffic's redundancy requirements separately can help online multiplayer games in better utilizing player's network resources.

5.5 Experiments with Workload Variations

This section presents the result of experiments with the following workload variations:

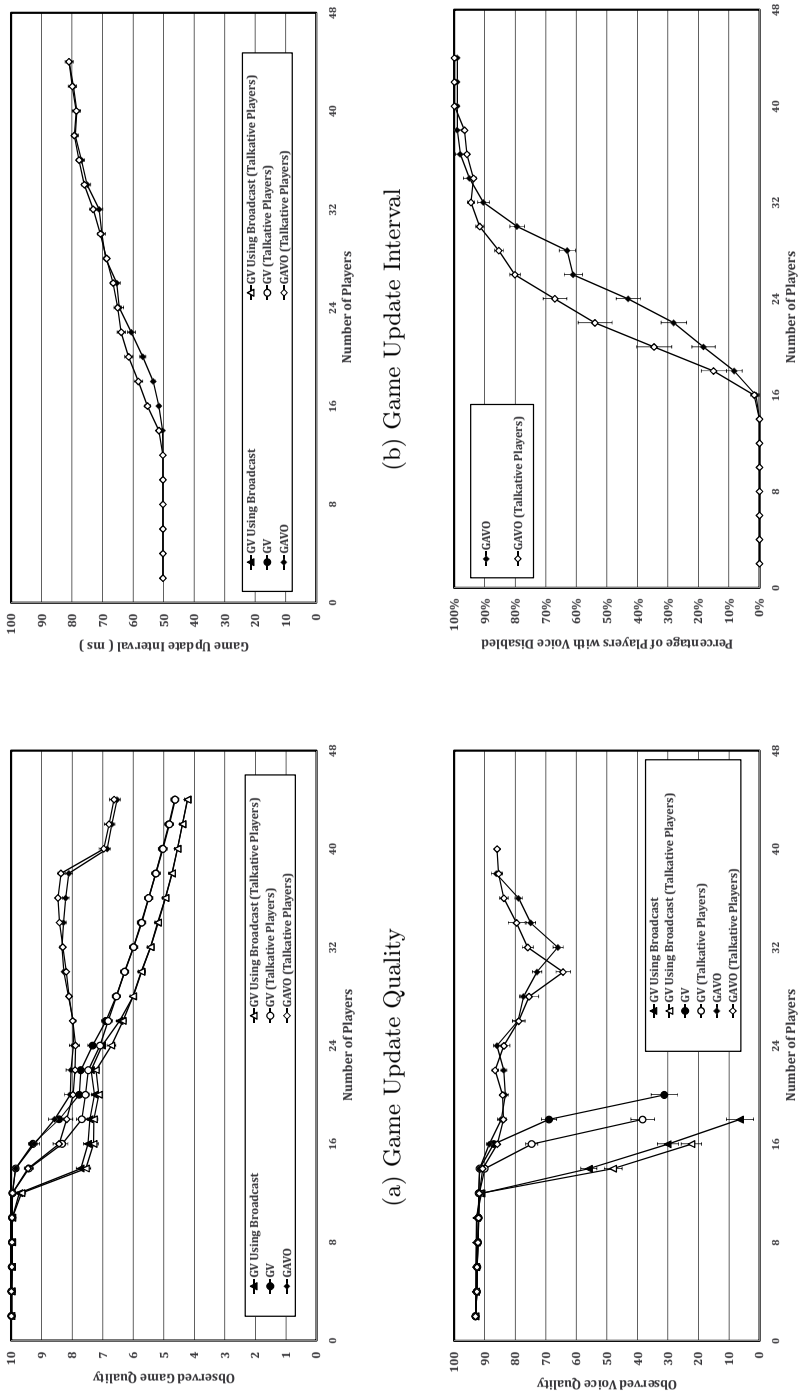
1. Talkative versus quiet VoIP sources (described in Section 4.2.2)

2. Large versus small file chunk–sizes (1KB and 256B respectively)

These experiments are necessary to better understand MCCA's behavior, in order to develop a set of guidelines for defining classes of network traffic. Figure 5.6 presents the experiment result for talkative versus quiet players, which are generally similar to experiments with variations in VoIP window-size (described in Section 5.4.2). Figure 5.7 presents the experiment result for large versus small file chunk–sizes, where results are generally similar to experiments with variations in file window-size (described in Section 5.4.2).

Results of experiments with talkative players demonstrate that the game using MCCA can relatively maintain the game and VoIP qualities at higher network traffics.

In addition, results of experiments with larger file chunk-sizes demonstrate the negative effect of large and low priority messages on higher priority classes of network traffic. Based on these results, we conclude that splitting large messages at the communication layer can help MCCA in better supporting network traffic classification.



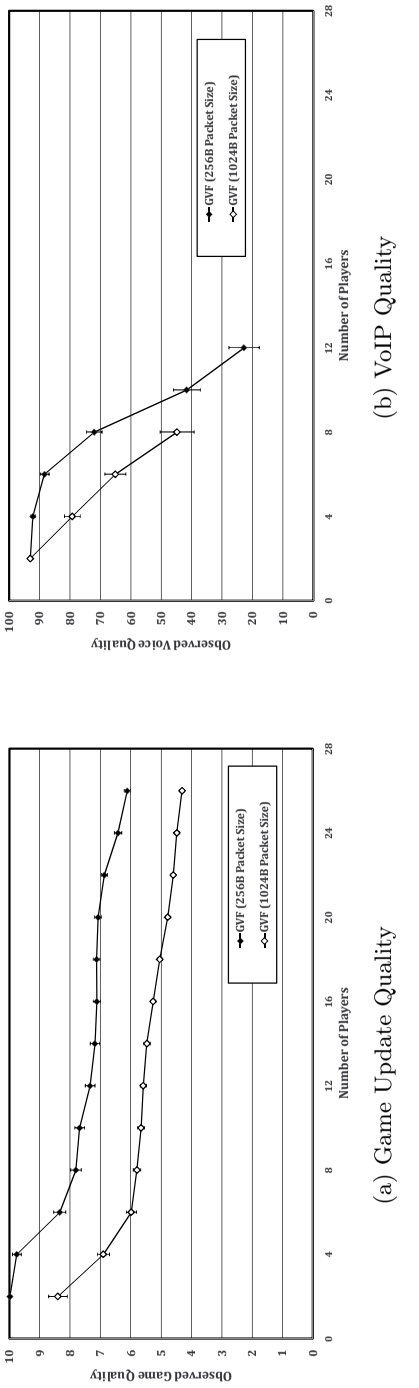
(a) Game Update Quality

(b) Game Update Interval

(c) VoIP Quality

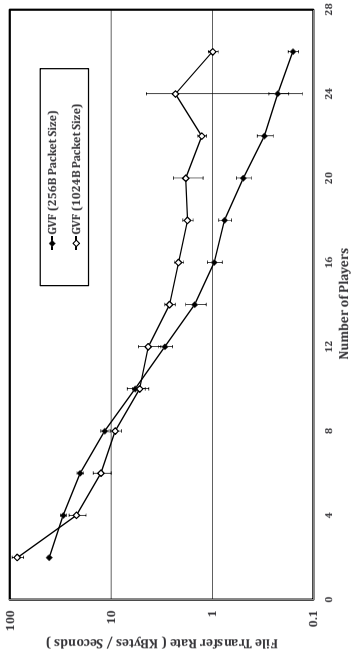
(d) Disabled VoIP Ratio

Figure 5.6: Experiment Results for Talkative vs. Quiet VoIP Sources



(a) Game Update Quality

(b) VoIP Quality



(c) File Transfer Rate

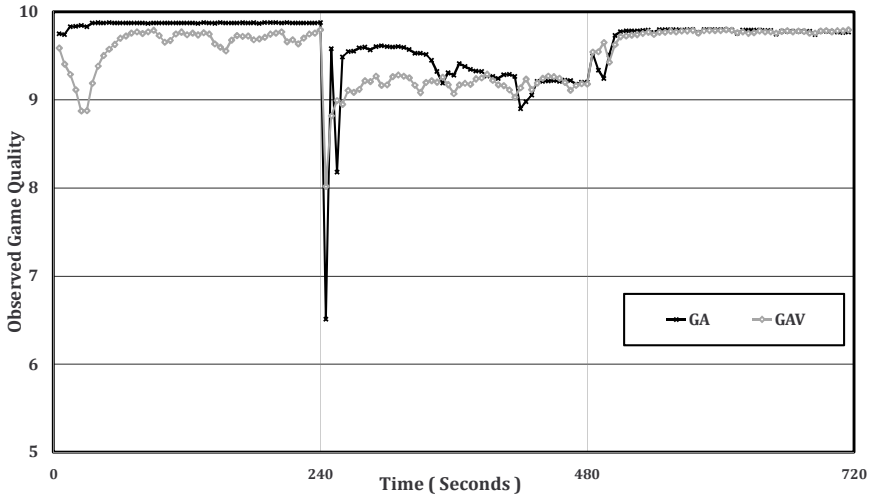
Figure 5.7: Experiment Results for Large vs. Small File Packet Sizes

5.6 Experiments with Real-Time Changes in the Number of Players and Network Conditions

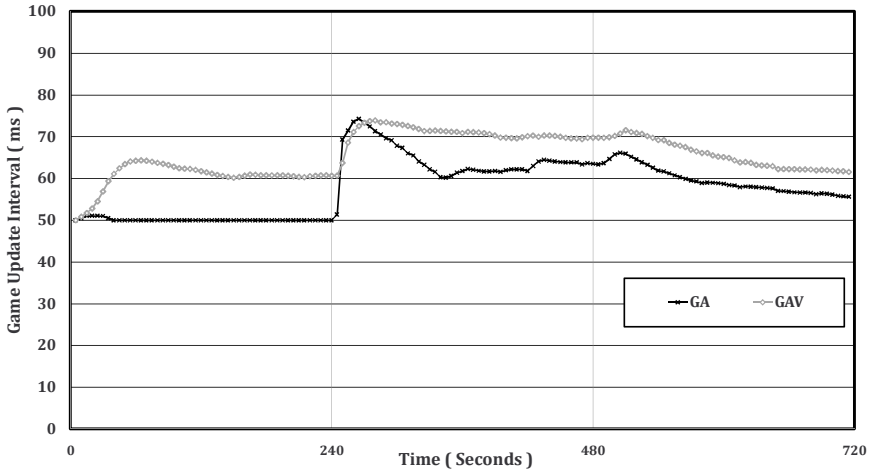
This section describes the experiments, which use an adaptive game update interval (described in Section 5.1.3), designed to study the potential benefits of distributed quality aggregation proposed by MCCA. These experiments evaluate the game quality and the game update interval for the following two scenarios: 1) experiment with real-time changes in cross network traffic (Section 5.6.1), and 2) experiment with real-time changes in the number of players (Section 5.6.2). These experiments aim to demonstrate the game's ability to adapt to network conditions, and number of players, by using MCCA's distributed quality aggregation to adjust the game's behavior and generated network traffic.

5.6.1 Cross Network Traffic

This experiment is designed to study the effects of cross network traffic on the game. The game quality and the game update interval are traced every five seconds for the lifetime of the experiment. This experiment simulates a 22 player game, where a cross network traffic that uses 30% of every player's upload capacity is introduced at the 4^{th} minute mark. The cross network traffic is then removed at the 8^{th} minute mark, and the game continues for another 4 minutes. Figure 5.8 shows the result of this experiment for a game only workload, as well as a game with VoIP workload. Results clearly demonstrate the game's ability to adapt to the network conditions, as the game quality quickly recovers (by modifying the game update interval) following a sharp reduction of the upload capacity.



(a) Game Update Quality



(b) Game Update Interval

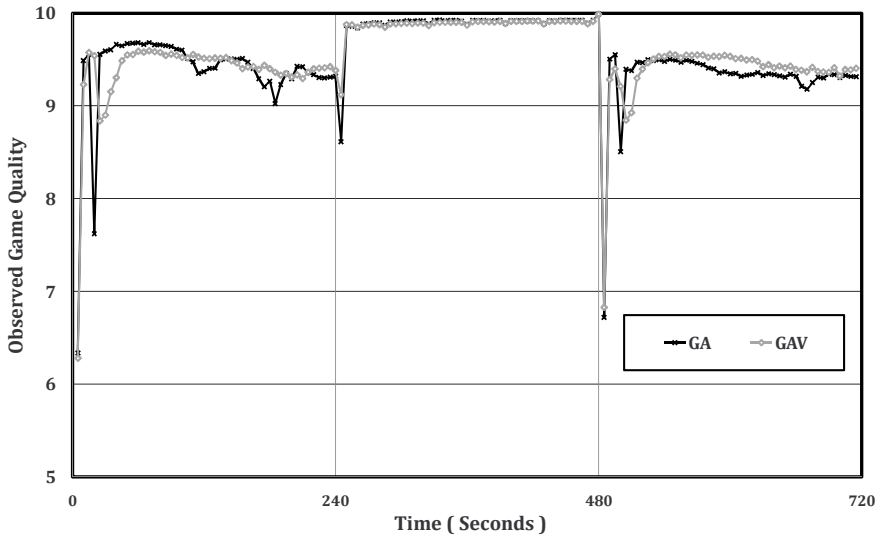
Figure 5.8: Game Adaptation to Network Conditions

5.6.2 Changes in the Number of Players

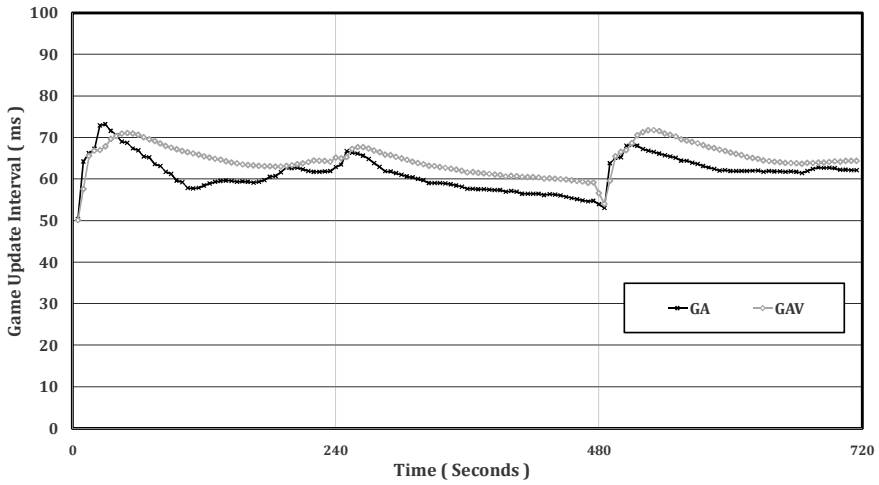
This experiment is designed to study the effects of changes in the number of players on the game (i.e., players leaving and joining the game). The game quality and the game update interval are traced every five seconds for the lifetime of the experiment. This experiment simulates a 30 player game where 18 players leave the game at the 4^{th} minute mark, and then the game continues as a 12 player game for 4 minutes. Then 18 players join the game, and then the game continues for another 4 minutes. Figure 5.9 shows the result of this experiment for a game only workload, as well as a game with VoIP workload. Results clearly demonstrate the game's ability to adapt to the number of players, as the game quality quickly recovers (by modifying the game update interval) following a large increase in the number of players.

5.6.3 Summary

Experiments presented in this section demonstrate that an online multiplayer game using the proposed distributed quality aggregation can adapt to real-time changes in network conditions and the number of players. These experiments demonstrate the vast potential for new types of game-level optimizations, such as "adaptive game update interval" (Section 5.1.3), based on the distributed quality aggregation presented in MCCA. In addition, results of experiments with players rapidly leaving and joining the game, demonstrate the proposed group communication's (Section 3.2) ability to support rapid group membership changes.



(a) Game Update Quality



(b) Game Update Interval

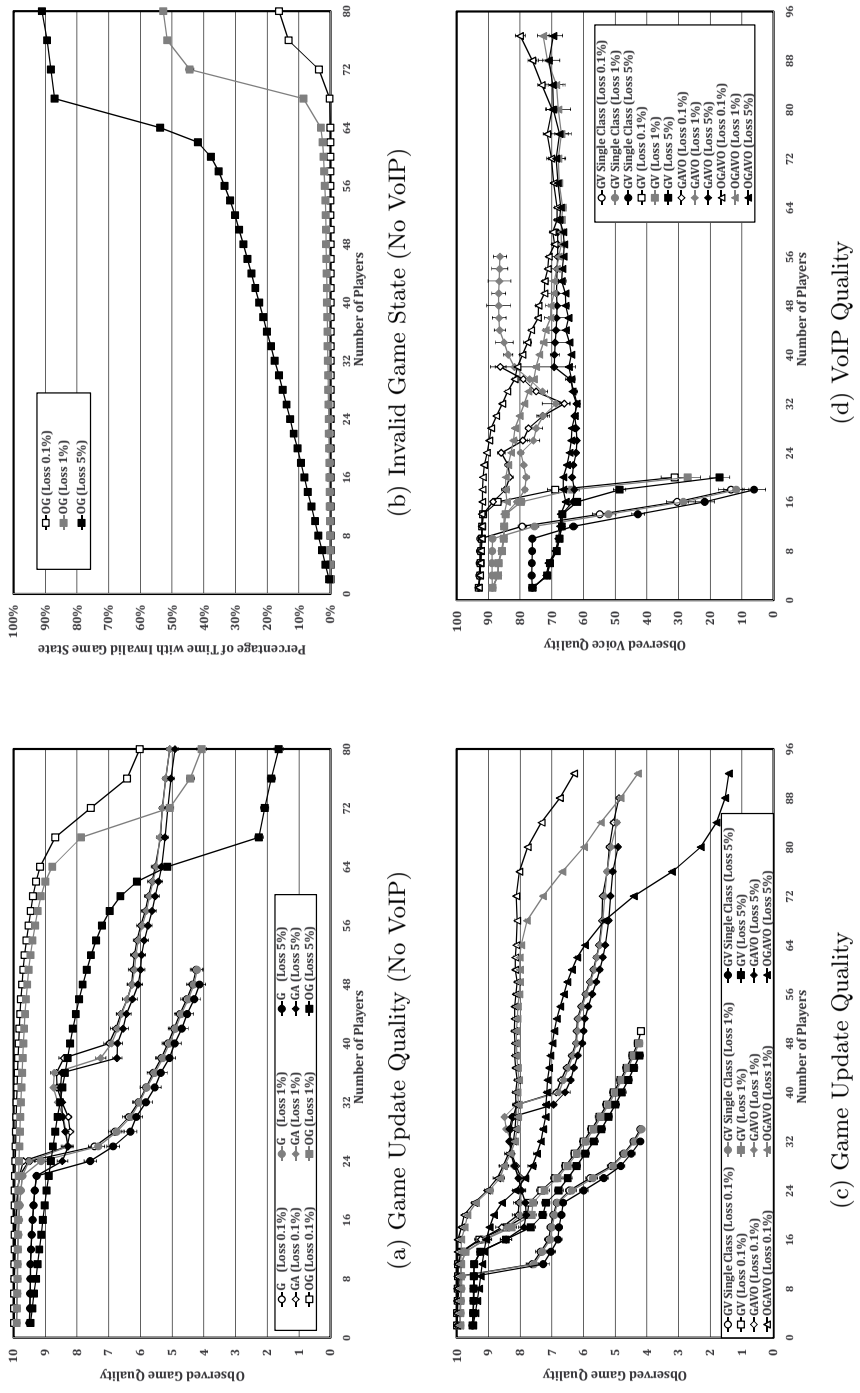
Figure 5.9: Game Adaptation to Player Presence

5.7 Experiments with Various Network Conditions

We performed the experiments presented in Sections 5.4, 5.5, and 5.6 using the default settings presented in Section 5.1.1. This section presents the experiments evaluating MCCA in other network conditions. Figure 5.10 presents the result of experiments with network loss ranging from 0.1% to 5%, and Figure 5.11 presents the result of experiments with different upload capacity ranges (presented Table 5.1).

Figures present the estimated game and VoIP qualities, average game update interval, and percentage of players with disabled VoIP due to VoIP optimization. Such figures are presented to clarify the effects of competing traffic (game and VoIP), as well as their corresponding game–level optimizations (demonstrated in form of the game update interval, and the percentage of players with disabled VoIP).

Results from experiments using various network conditions demonstrate the same trends, where network traffic prioritization, classification, and efficient group communication clearly improve quality. Similarly, game-level optimizations significantly improve the estimated game and VoIP qualities.



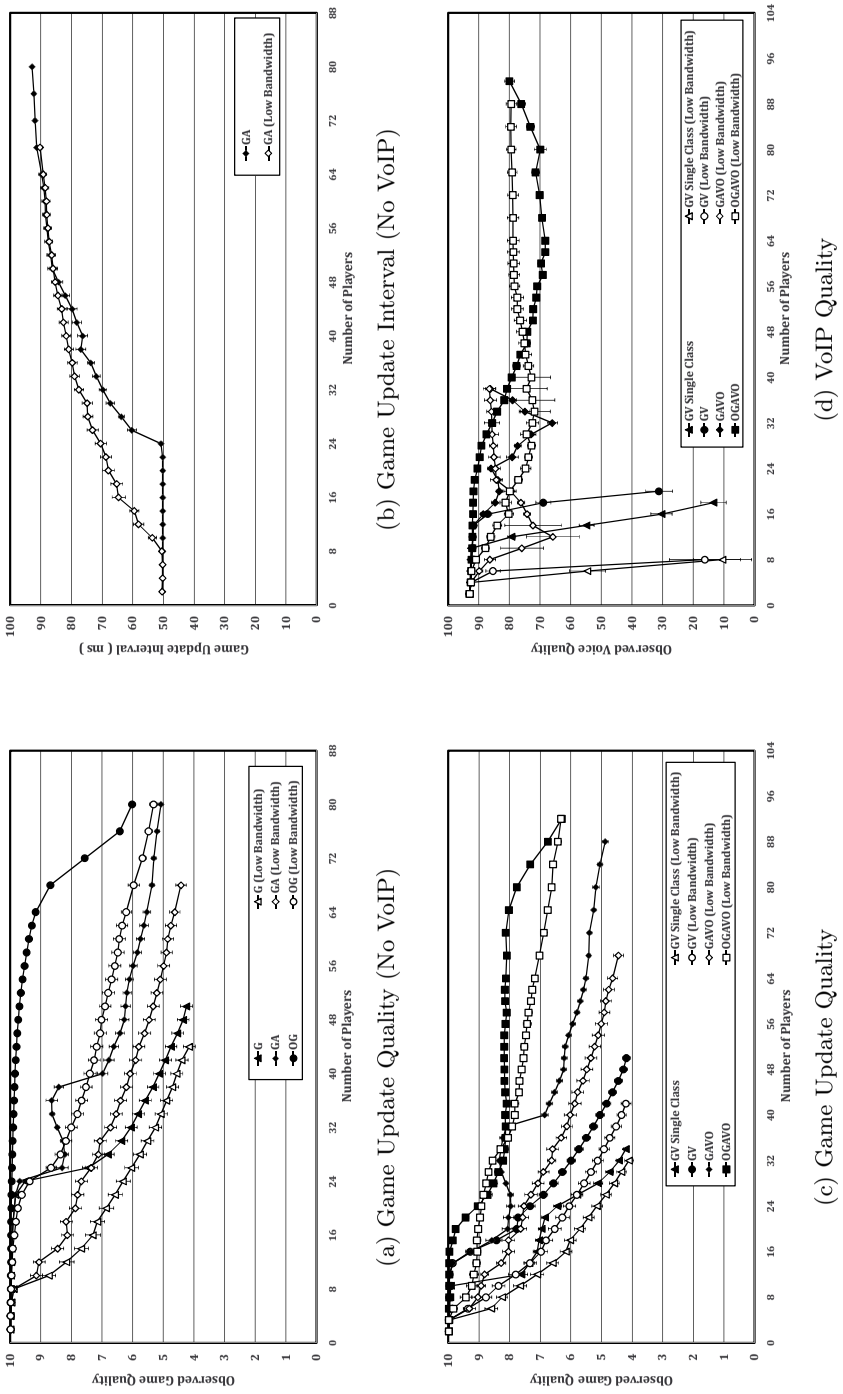
(a) Game Update Quality (No VoIP)

(b) Invalid Game State (No VoIP)

(c) Game Update Quality

(d) VoIP Quality

Figure 5.10: Experiment Results for Various Network Losses



(a) Game Update Quality (No VoIP)

(b) Game Update Interval (No VoIP)

(c) Game Update Quality

(d) VoIP Quality

Figure 5.11: Experiment Results for Various Network Capacities

5.8 Game Quality Variability Among Players

Considering that so far we have presented the average observed game quality, we investigated the game quality variability among players at various average game qualities. Figure 5.12 and Table 5.2 presents the result of this investigation, presenting the game quality distribution, where the average game quality is closest to 7.0 (the acceptable quality threshold, described in Section 5.2). This study also shows that a game using the suggested game-level optimizations, results in a less diverse game quality distribution (smaller standard deviations in Table 5.2). Consequently, we conclude that a game using MCCA can better manage its generated network traffic.

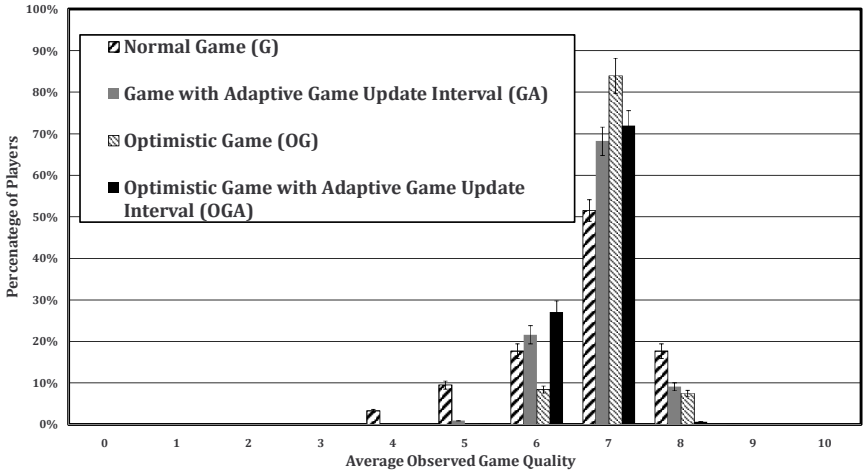


Figure 5.12: Game Quality Distribution at Unacceptable Average Game Quality

5.9 Comparison to The Optimal and Idealized Solution

MCCA is unique in providing a customizable communication layer for online multiplayer games, which complicates its comparison to other game-level or network QoS

Name	Number of Players	Average Quality	Standard Deviation
G	28	6.71	0.96
GA	40	6.85	0.53
OG	74	7.01	0.37
OGA	100	6.71	0.36

Table 5.2: Details of Experiments on Analysis of "Game Quality Distribution" (Figure 5.12)

solutions. This section compares the prototype implementation of MCCA to an optimal communication layer for Quake III.

The optimal and idealized solution is defined as an optimistic game that perfectly predicts network loss, in order to send duplicate messages, and avoid any invalid game states. Such prediction reduces the overall network traffic, by only transferring delta encoded updates, and improves the game quality by avoiding invalid game states. In addition, the optimal solution selects the best possible game update interval to achieve the highest game quality, and benefits from the exact knowledge of the network capacity and the network traffic.

Figure 5.13 presents the the game update quality for an experiment using the implemented MCCA communication layer, and the Quake III game using the optimal solution. The quality difference is mainly due to optimal solution's network loss prediction, which presents the area that MCCA requires improvement. For example, MCCA can use heuristics (similar to OverQos [60]) for message replication to provide statistical loss guarantees.

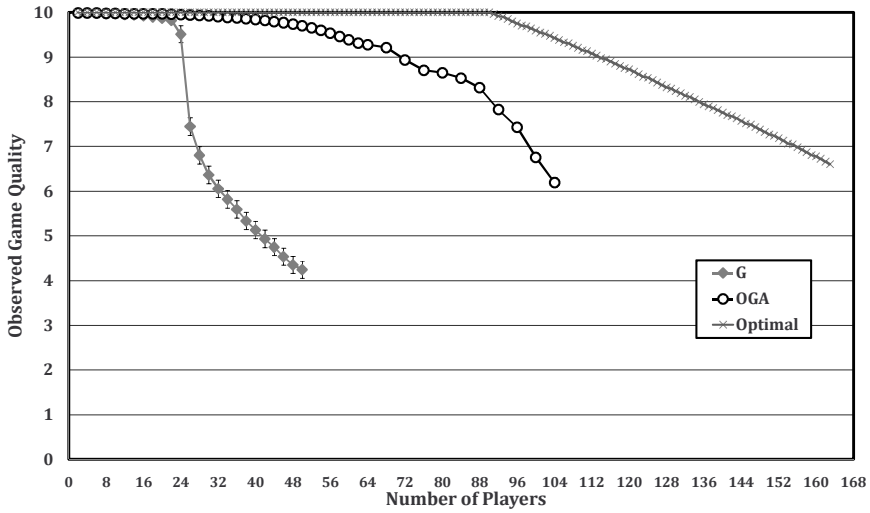


Figure 5.13: MCCA vs. Optimal Solution

5.10 Summary of Experiment Results

Experiments with various workload combinations and class definitions (Section 5.4) validate the benefits of network traffic prioritizations, classification, and flow specific efficient group communication. These results demonstrate that by using MCCA, a game can improve its estimated game quality, and use network resources more efficiently. Additionally, the experiments presented in Section 5.4.1 demonstrate MCCA's ability to support many types of network traffics with different requirements (e.g., system messages, game updates, VoIP packets, and file chunks). These workloads present a wide range of behaviors (i.e., message frequency and message size), while their latency, ordering, and reliability requirements are vastly different. Furthermore, experiments with real-time changes in the number of players and the cross network traffic (Section 5.6), demonstrate that distributed quality aggregation and reporting, such as proposed by MCCA, enables a game to adapt to dynamic network and workload conditions.

Experiments with class definition variations (Section 5.4.1), as well as, experiments with different window-size (Section 5.4.2), and message redundancy (Section 5.4.3) demonstrate MCCA's ability to support differentiated services at the communication layer. MCCA's class parameters are designed to support a wide range of class requirements, and results validate that a finely tunned class definition can reduce the network traffic, and at the same time satisfy a class's QoS requirements.

Experiments with workload variations (Section 5.5), such as the use of talkative VoIP sources, demonstrate the importance of network traffic classification and prioritization. Results show that a game using MCCA can maintain its game quality at higher VoIP traffics, while the game that does not use MCCA loses some game quality in the same conditions.

Furthermore, based on experiments with variations in file chunk–size (Section 5.5), smaller network messages are recommended in order to satisfy the game's strict latency requirements. Results show that a larger file chunk–size is potentially better for transmitting files, as it yields to higher file transfer throughput; however, a larger file chunk–size results in higher game message latencies that can severely degrade the game quality, as higher latencies result in higher network traffic, which reduces the overall file transfer throughput, at higher number of players.

Furthermore, experiments validate the intuition that network traffic classification and prioritization are more beneficial with more diverse workloads. Comparing the experiment results of 1) game only workload, 2) game with VoIP workload, and 3) game with VoIP and file workloads, shows that the experiment taking advantage of MCCA can support respectively 275%, 375%, and 600% more players.

Finally, experiments using a wide set of network deployment conditions such as higher network losses, and lower upload capacities, demonstrate the same trends in

results. Such experiments demonstrate the MCCA's strengths under wide range of network deployments.

Chapter 6

Conclusion and Future Work

This chapter revisits the material of this thesis in three respects. Firstly, Section 6.1 summarizes the problem addressed throughout the thesis. Secondly, Section 6.2 states the approach and summarizes the contributions of this work. Finally, Section 6.3 proposes a number of suggestions for the continuation of this work.

6.1 Problem with Current Approaches

This research is mainly motivated by the imbalance between the Internet infrastructure compared to the computer hardware. Over the last decade, the ability of the Internet infrastructure to carry network traffic has not improved at the same rate as the desktop technology. This imbalance has increased the perceived difference in the quality of service (QoS) offered by online multiplayer games compared to single player games. In addition, online multiplayer games are required to support many more functionalities, leading to transmission of VoIP messages, files, and more over communication networks. Such imbalance and additional workloads pose a significant challenge to online multiplayer game developers, and results in games supporting lower number of players in order to maintain quality.

In the past, researches have proposed a number of game-level optimizations for supporting a higher number of players [3–5, 23, 24, 28, 30, 36, 54, 55]; however, these

solutions are mainly game specific and focus on decreasing the game generated network traffic, or relaxing game's latency requirements. Online multiplayer games can also take advantage of research on network QoS [7, 58, 60] that provide statistical guarantees for the communication quality among players; however, such solutions don't take advantage of online multiplayer games' knowledge and control of the generated network traffic. The combination of challenges and potentials have motivated the research on a generic communication architecture targeted for online multiplayer games.

6.2 Contributions of this Work

This thesis focuses on a communication infrastructure optimized for online multiplayer games, and builds on past work in network QoS [7, 58, 60] as well as application layer multicast [12, 13, 26, 51, 56]. The main contribution of this thesis is the introduction of new mechanisms for using game-level information, such as the characteristics of a game's network traffics and requirements that drive players' quality of experience, to improve the communication layer's efficiency. In addition, this research takes the first step towards providing a general purpose game communication layer that takes advantage of game-level information, workload predictability, and flexibility. The following presents the main contributions of this research:

- Design and evaluation of MCCA, a communication architecture that supports network traffic classification, and targeted network QoS (providing a combination of reliable/unreliable, in-order/out-of-order message delivery, and mechanisms for supporting low latency communication). The experiment results validate our claim that a game using network traffic classification, prioritization, and targeted QoS can better support multiple types of network traffic such as game updates,

and VoIP.

- Design and evaluation of an efficient group communication protocol. Results show that even a randomly initialized multicast overlay, which evolves overtime can provide a more efficient network communication, mitigate network load of players with low network capacity, and handle rapid group membership changes.

- Design and evaluation of distributed network traffic quality aggregation for online multiplayer games based on a traffic's communication statistics and more. Experiment with game-level optimizations, demonstrating the game's ability to adapt to network conditions to maintain and/or improve the estimated game quality.

- Design and evaluation of new game-level optimizations based on MCCA, taking advantage of network traffic classification, group communication, and distributed quality aggregation. These examples demonstrate the potential for new types of game-level optimizations using MCCA.

- Experimental evaluation of MCCA using Quake III workload, VoIP, and file transfers, and evaluation of the affects of VoIP and file transfer on an online multiplayer game. Results also validate the following concepts: 1) game workload classification, prioritization, and targeted QoS improve the estimated game and VoIP quality, 2) classification is more beneficial at higher number and amount of network traffics, 3) application-level multicast reduces the network traffic and improves the overall quality, 4) distributed quality estimation, as a quality feedback for a game, can enable online multiplayer games to adapt to network conditions, and 5) classification and game-level optimizations, which use distributed quality estimation, result in lower quality variability among players.

In summary, this research proposes MCCA, a communication overlay that enables a game to classify its workload and benefit from targeted network QoS. Additionally, the proposed solution essentially enables new types of game-level optimizations, through distributed quality aggregation, to support higher number of players, improve quality, and/or adapt to network conditions. Finally, this thesis demonstrates that the proposed communication architecture, as a layer between the game and network infrastructure, can support most game-level optimizations proposed to date.

6.3 Future Directions

During the course of this research, a number of future directions have emerged for the continuation of this research topic. This section summarize these directions.

6.3.1 Real-World Deployment

Integrating the proposed communication architecture with an online multiplayer game, and implementing some of the discussed game-level optimizations. Such evaluation provides higher assurance in the results. In addition, having a working implementation of the solution, open to the research community, can better facilitate the research on online multiplayer games.

6.3.2 Incorporating Observed Network Conditions in MCCA

Over time, the communication layer gathers information about each player's network conditions as well as statistics such as loss, latency, and available upload capacity. Such information can be incorporated in the communication layer to better prioritize traffic, provide network QoS, and use network resources. For example, recognition of a lossy

communication can help in more efficient management of message redundancy.

6.3.3 Mechanisms for Simplifying Class Definition

Engineering class definitions pose a new challenge to game developers, and taking steps in reducing such complexity is essential. Developing guidelines for splitting and defining game workload is an essential research in reducing MCCA's complexity for developers, while the following general purpose solutions can be explored to provide a dynamic solution for game developers:

- **Simulation Based Parameter Search:** Given the developers's control over the generated network traffic, a simulator could be developed to find the best class definition for the given workloads. Game's network traffic traces and informations on the target deployment can be used in the simulator, which is responsible for finding the best possible configuration at various number of players and network conditions.

- **Neural Networks:** Integrating neural networks with MCCA to define, and manage network traffics, to achieve the highest game quality. Such solution would essentially reduce the developer's difficulty in defining different classes of network traffic, and can dynamically converge to the best configuration for the number of players and their network conditions.

6.3.4 Cheat Detection and Prevention

One of the main challenges, for online multiplayer games, is cheat detection and prevention. Extending the MCCA's network traffic classification, the communication layer can support a set of mechanisms for targeted cheat detection and prevention.

Bibliography

[1] Suman Banerjee, Seungjoon Lee, Ryan Braud, Bobby Bhattacharjee, and Aravind Srinivasan. Scalable resilient media streaming. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 4–9, New York, NY, USA, 2004. ACM.

[2] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003®. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151, New York, NY, USA, 2004. ACM.

[3] Eric J. Berglund and David R. Cheriton. Amaze: A multiplayer computer game. *IEEE Software*, 2(3):30–39, 1985.

[4] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: enabling large-scale, high-speed, peer-to-peer games. *SIGCOMM Comput. Commun. Rev.*, 38(4):389–400, 2008.

[5] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. Colyseus: a distributed architecture for online multiplayer games. In *NSDI'06: Proceedings of the 3rd*

conference on Networked Systems Design & Implementation, pages 12–12, Berkeley, CA, USA, 2006. USENIX Association.

[6] Ashwin R. Bharambe, Sanjay G. Rao, Venkata N. Padmanabhan, Srinivasan Seshan, and Hui Zhang. The impact of heterogeneous bandwidth constraints on dht-based multicast protocols. In *in The Fourth International Workshop on Peer-to-Peer Systems*, 2005.

[7] Steven Blake, Mark Carlson, Elwyn Davies, Nortel Uk, Borje Ohlman, Dinesh Verma, Zheng Wang, and Walter Weiss. An architecture for differentiated services. *IETF RFC*, 2475, 1998.

[8] Blizzard entertainment, world of warcraft reaches new milestone: 10 million subscribers. http://eu.blizzard.com/en/press/080122.html.

[9] Blizzard entertainment. wow pvp battlegrounds. http://www.worldofwarcraft.com/pvp/battlegrounds.

[10] Jean-Chrysotome Bolot. End-to-end packet delay and loss behavior in the internet. In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 289–298, New York, NY, USA, 1993. ACM.

[11] Michael S. Borella. Measurement and interpretation of internet packet loss. *Journal of Communication and Networks*, 2:93–102, 2000.

[12] Miguel Castro, Peter Druschel, Anne-marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *IPTPS '03*, 2003.

[13] Miguel Castro, Peter Druschel, Anne-marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20:2002, 2002.

[14] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. An empirical evaluation of tcp performance in online games. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 5, New York, NY, USA, 2006. ACM.

[15] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. In *in Proceedings of ACM Sigmetrics*, pages 1–12, 2000.

[16] Chen-nee Chuah, Randy H. Katz, Jean Walr, and George Shanthikumar. Providing end-to-end qos for ip-based latency-sensitive applications. Technical report, Berkeley, 2006.

[17] R. G. Cole and J. H. Rosenbluth. Voice over ip performance monitoring. *SIGCOMM Comput. Commun. Rev.*, 31(2):9–24, 2001.

[18] Alberto Dainotti, Antonio Pescape, and Giorgio Ventre. A packet-level traffic model of starcraft. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 33–42, Washington, DC, USA, 2005. IEEE Computer Society.

[19] Matthias Dick, Oliver Wellnitz, and Lars Wolf. Analysis of factors affecting players' performance and perception in multiplayer games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–7, New York, NY, USA, 2005. ACM.

[20] Johannes Färber. Network game traffic modelling. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 53–57, New York, NY, USA, 2002. ACM.

[21] Wu-chang Feng, Francis Chang, Wu-chi Feng, and Jonathan Walpole. Provisioning on-line games: A traffic analysis of a busy counter-strike server. In *in Internet Measurement Workshop*, pages 151–156, 2002.

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

[23] Stefan Fiedler, Michael Wallner, and Michael Weber. A communication architecture for massive multiplayer games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 14–22, New York, NY, USA, 2002. ACM.

[24] Laurent Gautier and Christophe Diot. Design and evaluation of mimaze, a multiplayer game on the internet. In *ICMCS '98: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, page 233, Washington, DC, USA, 1998. IEEE Computer Society.

[25] Growth of gaming in 2007 far outpaces movies. http://arstechnica.com/news.ars/post/20080124-growth-of-gaming-in-2007-far-outpaces-movies-music.html.

[26] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[27] John A. Hamilton, David A. Nash, and Udo W. Pooch. *Distributed Simulation*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[28] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. Von: A scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, 2006.

[29] IEEE. Ieee standard for distributed interactive simulation application protocols. *IEEE Standard 1278.1-1995*, September 1995.

[30] IEEE. Standard for modeling and simulation high level architecture. *IEEE standard 1516-2000*, September 2000.

[31] ITU-T. Methods for subjective determination of transmission quality. *ITU-T standard P.800*, August 1996.

[32] ITU-T. Objective quality measurement of telephoneband (3003400 hz) speech codecs. *ITU-T standard P.861*, August 1996.

[33] ITU-T. Perceptual evaluation of speech quality (pesq): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. *ITU-T standard P.862*, February 2001.

[34] ITU-T. The e-model, a computational model for use in transmission planning. *ITU-T standard G.107*, March 2005.

[35] Wenyu Jiang and Henning Schulzrinne. Modeling of packet loss and delay and their effect on real-time multimedia service quality. In *Proceedings of NOSSDAV '2000*, 2000.

[36] Bjorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games, 2004.

[37] Thomas J. Kostas, Michael S. Borella, Ikhlaq Sidhu, Guido M. Schuster, Jacek Grabiec, and Jerry Mahler. Real-time voice over packet-switched networks. *IEEE Network*, 12:18–27, 1998.

[38] J Lakkakorpia, A Heinerb, and J Ruutuc. Measurement and characterization of internet gaming traffic. *Helsinki University of Technology, Networking Laboratory*, 2002.

[39] Tanja Lang, Grenville Armitage, Phillip Branch, and Hwan-yi Choo. A synthetic traffic model for half-life. In *in Australian Telecommunications, Networks and Applications Conference (ATNAC)*, 2003.

[40] Ian Marsh. Measuring Internet telephony quality: Where are we today? In *Proceedings of IEEE Globecom: Global Internet*, Rio De Janeiro, Brazil, December 1999.

[41] Aaron McCoy, Declan Delaney, Seamus McLoone, and Tomas Ward. Dynamic hybrid strategy models for networked multiplayer games. In *In Proceedings of the 19th European Conference on Modelling and Simulation (ECMS)*, 2005.

[42] Aaron McCoy, Tomas Ward, Seamus Mcloone, and Declan Delaney. Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications. *ACM Trans. Model. Comput. Simul.*, 17(4):16, 2007.

[43] Sean McCreary and Kc Claffy. Trends in wide area ip traffic patterns: A view from ames internet exchange. In *In ITC Specialist Seminar on IP Traffic Modeling, Measurement and Management*, 2000.

[44] Net:netem - the linux foundation. http://www.linuxfoundation.org/en/Net:Netem.

[45] James Nichols and Mark Claypool. The effects of latency on online madden nfl football. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 146–151, New York, NY, USA, 2004. ACM.

[46] Vance ONeill. Adding creamy nougat and a crisp candy coating to the network xrnm & qnet. In *XNA GameFest*, 2007.

[47] Venkata N. Padmanabhan and Kunwadee Sripanidkulchai. The case for cooperative networking. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 178–190, London, UK, 2002. Springer-Verlag.

[48] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29, New York, NY, USA, 2002. ACM.

[49] Lothar Pantel and Lars C. Wolf. On the suitability of dead reckoning schemes for games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 79–84, New York, NY, USA, 2002. ACM.

[50] Vern Paxson. End-to-end routing behavior in the internet. *SIGCOMM Comput. Commun. Rev.*, 26(4):25–38, 1996.

[51] Daniel Pittman and Chris GauthierDickey. A measurement study of virtual populations in massively multiplayer online games. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 25–30, New York, NY, USA, 2007. ACM.

[52] Padungkrit Pragtong, Kazi M. Ahmed, and Tapio J. Erke. Analysis and modeling of voice over ip traffic in the real network. *IEICE - Trans. Inf. Syst.*, E89-D(12):2886–2896, 2006.

[53] 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 *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 152–156, New York, NY, USA, 2004. ACM.

[54] Sanjay Rao and Srinivasan Seshan. Mercury: a scalable publish-subscribe system for internet games. In *In Proceedings of the first workshop on Network and system support for games*, pages 3–9. ACM Press, 2002.

[55] Phillip Rosedale and Cory Ondrejka. Enabling player-created online worlds with grid computing and streaming. Technical report, Gamasutra, September 2003.

[56] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.

[57] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in warcraft iii. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 3–14, New York, NY, USA, 2003. ACM.

[58] Scott J. Shenker, Lixia Zhang, Deborah Estrin, Sugih Jamin, John Wroclawski, and Shai Herzog. Integrated services in the internet architecture: an overview. internet rfc 1633, 1994.

[59] Sony online entertainment. planetside faq. http://planetside.station.sony.com/faq.vm.

[60] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. Overqos: An overlay based architecture for enhancing internet qos. In *1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[61] Video game industry growth still strong: Study. http://www.reuters.com/article/industryNews/idUSN2132172920070623.

[62] D. Waitzman, C. Partridge, and S. E. Deering. Distance vector multicast routing protocol, 1988.

[63] M. Yajnik, Sue Moon, J. Kurose, and D. Towsley. Measurement and modelling of the temporal dependence in packet loss. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 345–352 vol.1, 1999.

[64] YVG, video game sales break records. http://videogames.yahoo.com/feature/video-game-sales-break-records/1181404.

Appendix A

MCCA's API

Program A.1 Types and Enumerations

```
typedef int PlayerId, ClassId, FlowId;

enum Error {
  Error_OK,
  ...
};

enum MessageType {
  MessageType_Unicast,
  MessageType_Broadcast
};

enum ForwardingPolicyType {
  ForwardingPolicyType_Star,
  ForwardingPolicyType_TwoLevelBandwidthTree,
  ForwardingPolicyType_TwoLevelLatencyTree,
  ForwardingPolicyType_Custom
};

enum QualityEstimationType {
  QualityEstimationType_None,
  QualityEstimationType_ObservedBandwidth,
  QualityEstimationType_VoIP
  QualityEstimationType_Game,
  QualityEstimationType_Custom
};
```

Program A.2 Interface for Receiving Message and Notifications

```
class ObservedUserQualityMap {
  int GetNumPlayers( );

  Error GetQuality(
    __in  PlayerId playerId,
    __out double* quality );
};

class FlowCallback {
  virtual void OnReceive(
    __in ClassId classId,
    __in FlowId flowId,
    __in PlayerId from,
    __in MessageType messageType,
    __in void* data,
    __in unsigned int dataSize ) = 0;

  virtual void OnNConsequetiveLosses(
    __in ClassId classId,
    __in FlowId flowId ) {};

  virtual void OnObservedQualityUpdate(
    __in ClassId classId,
    __in FlowId flowId,
    __in ObservedUserQualityMap qualityMap ) {};
};
```

Program A.3 Custom Forwarding Policy Creator

```
class NetworkTopology {
  int GetNumPlayers( );

  void GetPlayerNetworkInfo(
    __in  PlayerId playerId,
    __out double* bandwidth,
    __out double* roundTripTime );
};

class CustomForwardingPolicy {
  Error CreateTopology(
    __in  const NetworkTopology* networkTopology,
    __out ForwardingPolicy* forwardingPolicy );

  Error UpdateTopology{
    __in  const NetworkTopology* networkTopology );

  Error OnPlayerJoin(
    __in const NetworkTopology* networkTopology,
    __in PlayerId joinedPlayerId );

  Error OnPlayerLeave(
    __in const NetworkTopology* networkTopology,
    __in PlayerId leftPlayerId );
};
```

Program A.4 Quality Aggregation Interface[1]

```
class LatencyHistogram {
  void Begin();

  bool Next(
    __out double* latencyRangeLow,
    __out double* latencyRangeHigh,
    __out double* average,
    __out int* count );
};

class QualityEstimationCallback {
  double GetQuality(
    __in const LatencyHistogram* latencyHistogram,
    __in double averageLatency,
    __in double observedLossRate,
    __in double averageJitter,
    __in int numberOfMessages );
};

Error setFlowQualityEstimation(
  __in ClassId classId,
  __in FlowId flowId,
  __in QualityEstimationType qmType,
  __in QualityEstimationCallback* qmCallback = NULL,
  __in double qmLatencyHistogramInterval = 0.05 );
```

[1]Refer to Program A.2 for the interface of the *FlowCallback*, and the aggregated quality callback

Program A.5 Consecutive Loss Reporting Interface[1]

```
Error setOnNConsecutiveLossesCallback(
  __in ClassId classId,
  __in FlowId flowId,
  __in int numLosses );

Error clearOnNConsecutiveLossesCallback(
  __in ClassId classId,
  __in FlowId flowId );
```

[1]Refer to Program A.2 for the interface of the *FlowCallback*, and the consecutive loss callback

Appendix B

Abbreviations

Acronym	Description
AI	Artificial Intelligence
AOI	Area of Interest
API	Application Programming Interface
CLVL	Controlled Loss Virtual Links
DVMRP	Distance Vector Multicast Routing Protocol
FPS	First Person Shooter
IO	Input/Output
IP	Internet Protocol
ITU-T	Telecommunication Standardization Sector
MCCA	Multi-Class Communication Architecture
MMORPG	Massively Multiplayer Online Role-Playing Game
MOS	Mean Opinion Score
P2P	Peer-to-Peer
PCM	Predictive Contract Mechanisms
PESQ	Perceptual Evaluation of Speech Quality
PSQM	Perceptual Speech Quality Measure
QoS	Quality of Service
RPF	Reverse Path Forwarding
RTS	Real Time Strategy
RTT	Round Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice-Over-IP
WoW	World of Warcraft

Table B.1: Technical Terms and Acronyms

Acronym	**Description**
G	Game
GA	Game with Adaptive Game Update Interval
OG	Optimistic Game
OGA	Optimistic Game with Adaptive Game Update Interval
GV	Game + VoIP Workload
GVO	Game + VoIP Workload with Optimization
GAV	Game with Adaptive Game Update Interval + VoIP Workload
GAVO	Game with Adaptive Game Update Interval + VoIP Workload with Optimization
OGV	Optimistic Game + VoIP Workload
OGVO	Optimistic Game + VoIP Workload with Optimization
OGAV	Optimistic Game with Adaptive Game Update Interval + VoIP Workload
OGAVO	Optimistic Game with Adaptive Game Update Interval + VoIP Workload with Optimization
GVF	Game + VoIP Workload + File Transfer Workload
GVOF	Game + VoIP Workload with Optimization + File Transfer Workload
GAVF	Game with Adaptive Game Update Interval + VoIP Workload + File Transfer Workload
GAVOF	Game with Adaptive Game Update Interval + VoIP Workload with Optimization + File Transfer Workload
OGVF	Optimistic Game + VoIP Workload + File Transfer Workload
OGVOF	Optimistic Game + VoIP Workload with Optimization + File Transfer Workload
OGAVF	Optimistic Game with Adaptive Game Update Interval + VoIP Workload + File Transfer Workload
OGAVOF	Optimistic Game with Adaptive Game Update Interval + VoIP Workload with Optimization + File Transfer Workload

Table B.2: Experiment Names and Acronyms