LATENCY REDUCTION IN ONLINE MULTIPLAYER GAMES USING DETOUR ROUTING

by

Cong Ly

B.Sc., Simon Fraser University, Burnaby, BC, Canada, 2002

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2010

APPROVAL

Name:	Cong Ly
Degree:	Master of Science
Title of Thesis:	Latency Reduction in Online Multiplayer Games Using Detour Routing
Examining Committee:	Dr. Tamara Smyth Chair
	Dr. Mohamed Hefeeda, Senior Supervisor
	Dr. Ramesh Krishnamurti, Supervisor
	Dr. Joseph Peters, Examiner
Date Approved:	May 26, 2010

Declaration of
Partial Copyright Licence

Abstract

Long network latency negatively impacts the performance of online multiplayer games. In this thesis, we propose a novel approach to reduce the network latency in online gaming. Our approach employs application level detour routing in which game-state update messages between two players can be forwarded through other intermediate relay nodes in order to reduce network latency. We present results from an extensive measurement study to show the potential benefits of detour routing in online games. We also present the design of a complete system to achieve the potential, which is called Indirect Relay System (IRS). The experimental and simulation results show that IRS: (i) significantly reduces end-to-end round-trip times (RTTs) among players, (ii) increases number of peers a player can connect to and maintain good gaming quality, (iii) imposes negligible network and processing overheads, and (iv) improves gaming quality and player performance.

Keywords: online multiplayer games; video games; latency reductions; detour routing

To my son Alexander Jordan.

"Anyone who has lost track of time when using a computer knows the propensity to dream, the urge to make dreams come true and the tendency to miss lunch."

— *Tim Berners-Lee*

Acknowledgments

First and foremost, I would like to express my deepest and most sincere gratitude to Dr. Mohamed Hefeeda, my senior supervisor, for his patience, motivation, and continuous support. Dr. Hefeeda has an infectious enthusiasm for his research that is simply contagious. He is always willing to share his considerable knowledge when needed. This thesis would not have been possible without him.

I would like to thank Dr. Ramesh Krishnamurti, my supervisor, for all those times I bothered him with algorithm questions. If it was not for his enthusiasm in algorithm I would still be afraid to open a textbook with 'algorithm' in the title. I would also like to express my gratitude to Dr. Joseph Peters, my thesis examiner, for being on my committee and reviewing this thesis. I would also like to thank Dr. Tamara Smyth for taking the time to chair my thesis defense.

A big thank you to my many student colleagues at the Network Systems Lab for providing a stimulating and fun environment in which to learn and grow. I am thankful to Cheng-Hsin Hsu, Yi Liu , Ahmed Hamza, Mohammad Alkurbi, and R. Cameron Harvey for their friendship. I am especially grateful to Cheng-Hsin Hsu for his guidance, sharing his knowledge, and encouragement. The knowledge that I have gained through the many insightful discussions with Cheng during the past few years are invaluable. It has been an honor for me to have worked with such talented people.

I am indebted to my family for their love, and endless support. I owe my loving thanks to my wife Davina, and son Alexander. I will never forget all the kicking from Alexander at 5 a.m. in the morning to remind me to get up and work on my papers and thesis; especially on nights when daddy went to bed at 1 a.m. Without their encouragement, loving support, and understanding, it would be impossible for me to finish this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we provide a brief introduction about online multiplayer games. Then, we introduce the network latency delay problem we aim to address in this thesis and summarize the contributions. The organization of this thesis is given at the end of the chapter.

1.1 Introduction

Online multiplayer games have become increasingly popular in the past few years, and several market forecasts predict that online games will continue to grow in terms of market revenues, number of users, and generated Internet traffic volume. The latest Nielson report reveals that 37% of the US population played online games in 2009 [4], and market research forecasts that the online game industry will grow from $21.33 billion in 2008 to more than $48.9 billion in 2011 [5]. Online multiplayer games such as World of Warcraft (WoW), Halo, Quake, and Counter-Strike have attracted millions of players around the world [6]. The network traffic from online games currently constitutes a sizeable fraction of the Internet backbone traffic [7]. These numbers indicate that efficiently providing high-quality online gaming experience to a huge number of players is important not only to the success of the gaming industry, but also to network administrators and Internet Service Providers (ISPs) [8].

Providing high-quality online multiplayer gaming experience is challenging due to the best-effort nature of the Internet. For large-scale distributed systems in the Internet, high latency, insufficient bandwidth, and packet loss are the three most common obstacles for providing good quality-of-service, and need to be carefully addressed. Online multiplayer

games generate low bit rate streams with highly periodic traffic patterns for frequent game-state updates, which are less sensitive to bandwidth and packet loss. This is because the bit rates of these traffic streams are much smaller than the capacity of broadband access links, and packet losses can be absorbed by frequent game-state updates. In contrast, online multiplayer games require real-time interactions, and high latency becomes the main challenge.

In this thesis, we explore the potential of using detour routing to reduce network latency among players in online multiplayer games. More precisely, we propose to reduce the end-to-end RTT (round-trip time) between any two players by sending the game-state updates over detour paths through other players. The proposed system *directly* reduces RTTs among players in online games, while most of the earlier works mitigate the negative impacts of latency by either applying latency compensation techniques [9, Chapter 2], or matching players exclusively with nearby players in terms of RTTs [10, 11]. Through an actual implementation and extensive trace-driven simulations, we show that compared to direct links used in current online games, the system significantly reduces RTTs among players, while imposing negligible network and processing overheads. For example, in the real experiments, the system achieves more than 100 msec RTT reduction in 40% of game sessions, while each player on average sends one additional message every 16 sec which is indeed negligible compared to the frequent game-state updates exchanged in online games.

1.2 Problem Statement and Thesis Contributions

In this section, we state the research problem studied and addressed in this thesis. We also summarize the thesis contributions.

1.2.1 Problem Statement

Like many other large-scale distributed systems, online multiplayer games may suffer from the high network latency, limited bandwidth, and high packet loss ratio of the best-effort Internet. Fortunately, online multiplayer games are *not* very sensitive to limited bandwidth and high packet loss ratio. This is because online multiplayer games generate low bit rate and highly periodic traffic streams that consist of frequent game-state updates. These low bit rate traffic streams can easily fit into the bandwidth of current broadband access links, and packet losses are somewhat mitigated by the frequent game-state updates. However,

since online multiplayer games involve real-time interactions from players, high network latency becomes the main challenge for providing high-quality gaming experience. For example, in first-person shooter games, higher network latency leads to fewer number of frames rendered per second, which in turn results in sluggish responsiveness and affects players' shooting accuracy. More precisely, this problem can be stated as follows:

Problem 1. *Consider online multiplayer gaming networks in which players form gaming sessions. Players in the same session continuously exchange game-state updates to maintain a consistent view of the game. Long network latency among players negatively impacts the on-time delivery of game-state updates and hence the gaming quality observed by the players. Design an efficient system to reduce the network latency among players. The system should impose minimal overhead on players, and it should increase the number of potential players that can be matched with each player.*

1.2.2 Thesis Contributions

We propose a novel approach to solve the network latency reduction problem in online gaming. Our approach employs application level detour routing in which game-state update messages between two players can be forwarded through other intermediate relay nodes in order to reduce network latency. In particular, we explore whether detour routing yields better gaming experience. We study the magnitude of the expected performance improvement in different types of online multiplayer games when a detour path includes up to $1, 2, 3, \ldots,$ or k^* relay clients, where k^* is the number of relay clients that yields the minimum delay between the two ends of the detour path. We then propose a new overlay routing system designed for online multiplayer games, which can efficiently locate the best detour path between any two players through a third player, so that the end-to-end RTT is minimized. More concretely, the contributions of this thesis can be summarized as follows:

- We conduct a measurement study to collect RTT values among players of a popular online game. The trace files contain more than 18.8 million pairwise RTT measurements collected from online game players distributed over almost 8,000 subnets. We filter out unreliable measurements, and make the traces available to the research community.
- Using the traces, we quantify the potential of the detour routing in online games. We show that more than 40% of players can observe 100 msec or more RTT reduction

by routing game-state updates through 1-hop detour paths. We also show that, with detour paths, players can join online game sessions that were not available to them because of the long network latency of the direct paths.

- We analyze the expected impact of the detour routing on player performance in different online games. We report significant improvements in various player performance metrics, such as lap completion time in car racing games and hit ratio in first-person shooter games. The results indicate that game developers can employ detour routing to improve the gaming quality, attract more players, and increase their revenues.

- We show that most of the improvements in player performance metrics can be achieved by employing detour paths with only one intermediate relay node.

- We design and implement the proposed detour routing, which we call the Indirect Relay System (IRS)[1]. IRS employs a distributed network coordinate system and a computationally efficient algorithm to locate the optimal detour paths. The results from real experiments show that the proposed system improves the online gaming quality from several aspects, while incurring negligible network and processing overheads.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides brief background on online multiplayer games and detour routing for online multiplayer games, and summarizes the related works in the literature. In Chapter 3, the potential performance improvements of online multiplayer games using detour routing are quantified in detail. The design and implementation of the Indirect Relay System (IRS) is presented in Chapter 4. Deployment, evaluation, and experimental results are presented in Chapter 5. Chapter 6 concludes the thesis and outlines potential extensions for this work.

[1]This is a joint work with my colleague Cheng-Hsin Hsu under the supervision of Dr. Mohamed Hefeeda.

Chapter 2

Background

In this chapter, we provide background about online multiplayer games, and detour routing. We then summarize previous works in the literature that are closely related to the work in this thesis.

2.1 Online Multiplayer Games

In online multiplayer games, players are linked by a network (most likely the Internet) and are generally in different geographic locations. Players in the same gaming *session* share a *game world*, which consists of a collection of game objects and states. Interactions between players and game objects are communicated via game-state updates. In this section, we briefly discuss the different types of online multiplayer games. This is followed by a discussion of the impacts of network latency on online multiplayer games. We then discuss a general model for online gaming networks.

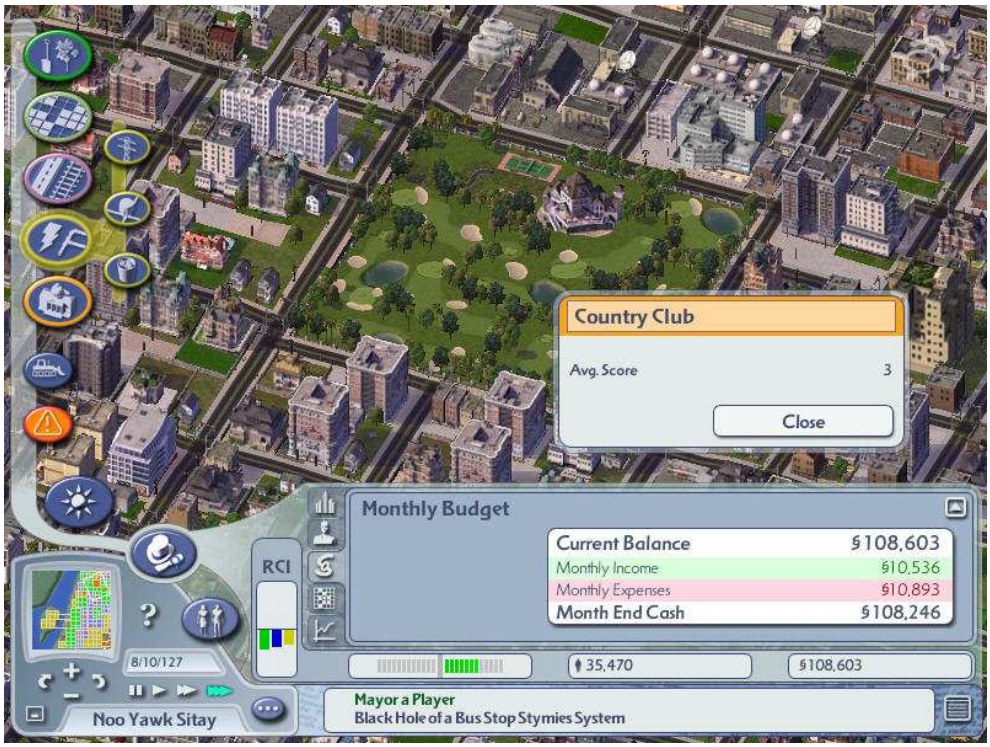
Game Classification. Online multiplayer games are roughly classified into two types: avatar games and omnipresent games [16]. In avatar games, a player controls a single character that exists at a precise location in the shared virtual space and can only interact with near-by objects. Avatar games include shooter games, role-playing games, action games, and sports games. These games are further categorized into first-person avatar games in which a player views through the character's eyes as illustrated in Fig. 2.1(a) and third-person avatar games in which a player sees the character from a distance as illustrated Fig. 2.1(b). In omnipresent games, a player concurrently controls a group of characters, and can interact with objects that are close to any of these characters. Omnipresent games

(a) Team Fortress 2 [12]

(b) Grand Theft Auto IV [13]

Figure 2.1: Avatar Games: (a) First-person shooter, and (b) third-person avatar.

(a) Starcraft 2 [14]



(b) SimCity 4 [15]

Figure 2.2: Omnipresent Games: (a) real-time strategy, and (b) omnipresent simulation.

Table 2.1: Latency Thresholds

Game Type	Example Genres	Latency Thresholds
Avatar	FPS, Racing	100 msec
	Sports, RPG	500 msec
Omnipresent	RTS, Sim	1,000 msec

include real-time strategy games and simulation games as illustrated in Fig. 2.2.

Network Latency. Players of different types of games can tolerate different amounts of quality degradation caused by network latency, which can be quantified by the gaming performance of players [16]. This is because games have various degrees of *tightness* on the delivery deadline of game-state updates. For example, high network latency leads to fewer pictures rendered per second, which has a significant negative impact on players' lap completion times in racing games. In contrast, players of real-time strategy games continuously monitor many gaming objects on a large map, and may be less sensitive to irregular movements of a single object. More precisely, several experimental studies show that while players of omnipresent games can tolerate up to 1 sec RTT, players of avatar games have more stringent latency requirements: 100 and 500 msec RTT for first-person and third-person avatar games, respectively as summarized in Table. 2.1. Network latencies higher than these thresholds lead to significant drops in gaming performance, and could turn players away from the online games. Therefore, game developers *must* take network latency into consideration when developing new games to provide high-quality gaming experience to players.

General Model. In this thesis, we consider a fairly general model for online gaming networks, where several players form a session and exchange game-state updates. Players outside a session are not interested in the game-state updates of that session. This model is general because it can be readily mapped to different types of games. For example, in avatar games, upon agreeing on the game settings such as the map and rules, several players start a game and exchange game-state updates. That is, they form a session in our model. In large-scale simulation games, thousands of players are playing in a virtual world. While there is no equivalence concept to the game session in avatar games, players that are far away from each other in the virtual world do not interact with each other. Therefore, several previous works have proposed to divide the virtual world into smaller segments through a process known as segmentation [17]. Segments are smaller regions of the virtual world.

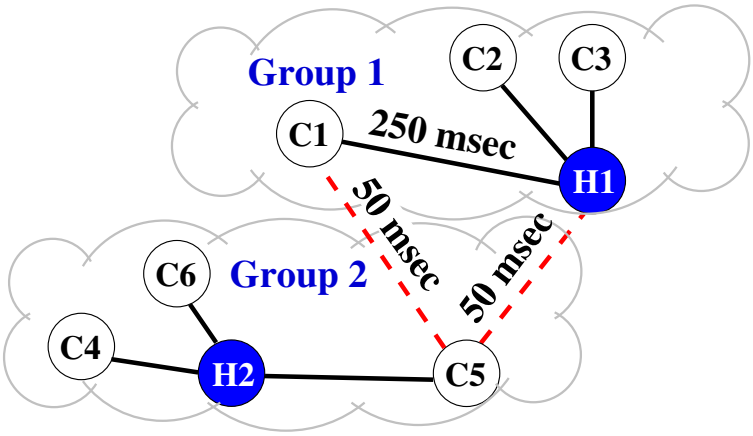


Figure 2.3: Internet routing may not be optimal in terms of latency.

Players in the same segment exchange game-state updates, and thus each segment is similar to a game session in our model.

While this model is applicable to various online games, we will only consider avatar games in the rest of this thesis for concrete discussion. In avatar games, each player runs a copy of the game software on his/her machine, and this machine is referred to as a client. Once a player decides to play the game, he/she needs to find other players through a centralized server called the *lobby server*, which is also known as the master server. Lobby servers are usually provided and maintained by companies that develop online games, and their locations are hardcoded in the game software [18]. With the help of the lobby server, several clients form a gaming session, which has a common set of game settings including number of players, map, and gaming rules. Clients in the same session frequently exchange game-state updates. Such message exchanging is usually done over an overlay network. To prevent inconsistency in the game state, game-state updates must be validated before being trusted. In each game session, one of the clients is chosen as the host, which is responsible for validating the game-state updates, and thus is also known as the authoritative client in that session. The host runs the main gaming logic, and forwards valid updates to all clients in the same session.

2.2 Detour Routing for Online Multiplayer Games

Network latency reduction can be accomplished by finding *detour paths* with one or more intermediate clients through which the gaming traffic is routed. These intermediate clients are called *relays*. A detour path can be defined as an application-level routing path that results in end-to-end delay (including the overhead) shorter than that of the direct Internet path. Therefore, a detour path *never* leads to network latency worse than the direct path between source and destination clients. A k-hop detour path goes through k relays, where $k \geq 0$. When $k = 0$, the direct Internet path between source and destination clients is used without any relays.

The direct path between a pair of clients may have longer RTT than a detour path between them, because of the triangle inequality violations (TIVs) in the Internet [19]. TIVs exist in the Internet for at least two reasons: (i) a congested or failed router on the direct path can inflate the RTT of that path, and (ii) inter-domain policy routing may be inefficient in terms of latency. The second reason is because ISPs maintain various commercial relationships, which can be classified into three types: customer-to-provider (c2p), peer-to-peer (p2p), and sibling-to-sibling (s2s) [20]. For a c2p link, a customer autonomous system (AS) pays a provider AS for carrying Internet traffic, and thus the customer AS does not transport cross traffic for its provider AS. For a p2p link, two peering ASes exchange traffic that has originated from or is destined to them or their customer ASes only. An s2s link connects two ASes belonging to the same ISP and they can freely exchange any traffic. Inter-domain routing chooses IP routes that conform to the above routing policies, and these routes are usually called *valley-free* [21]. Since policy routes must be valley-free, certain routes may be feasible for the detour paths but invalid for the policy routing. This may result in detour paths that are shorter than direct paths. With TIVs in the Internet, we can minimize the pairwise network latency among gaming clients in the same group by locating the best detour path for each pair of clients. To utilize detour paths, each client maintains a detour routing table. The table of a client has entries for other clients with which this client exchanges messages. For example, assume that client C1 in Fig. 2.3 sends messages only to the host H1. In this case, the detour routing table of C1 will have an entry <H1, C5>, which means that game-state updates sent to H1 should be sent through the relay client C5.

2.3 Related Work

The work in this thesis is closely related to two general approaches to dealing with network latency: detour routing for latency reduction, and online multiplayer games latency compensation mechanisms.

2.3.1 Latency Compensation Mechanisms

Network latency causes delays which make it nearly impossible for players to exchange their current positions and game-state at every game loop. Therefore, coping with network latency is critical to the quality of user experience in online games [16]. Several latency compensation mechanisms have been proposed by the research community and used by the gaming industry. These mechanisms are roughly categorized into two groups: time manipulation and matchmaking. Time manipulation mechanisms compensate for latency by adjusting the timestamp of game-state updates. Time manipulation mechanisms can further be classified into two approaches: time delay such as lockstep and event-locking, and time wrap such as dead reckoning.

Lockstep and event-locking are two popular mechanisms adopted by the gaming industry. The lockstep algorithm [22] controls consistency among clients with varying latency. With this algorithm, clients send out game-state updates at fixed time intervals, and a client is blocked until receiving updates from all other clients. In event-locking [23], clients send game-state updates to a server, and the server relays them to all clients in the same session. While lockstep and event-locking are suitable on local-area networks, they perform poorly in the Internet [24].

In dead reckoning [25, 26], clients extrapolate the behavior and state of gaming objects and thus can continue rendering frames even if game-state updates are late. A dead reckoning vector is commonly used to provide game-state updates about player and object movements. A dead reckoning vector typically contains the current position of the player in terms of x, y, and z coordinates as well as its trajectory. Clients are required to maintain two parallel models, an extrapolated path often called ghost model and an actual path that represents the true state as computed from player inputs. The ghost model represents an approximation of the true entity state. While dead reckoning *may* be invisible to the players, in real time there is a deviation at the receiving player between the actual and the

extrapolated trajectories. To maintain consistency, clients agree on some thresholds of prediction errors for various types of game-states. Game-state corrections are sent by the server or host if these thresholds are exceeded. Under this principle, several improvements have been proposed for the dead reckoning system, e.g., the authors of [27] propose to augment it with synchronized clocks in order to improve the consistency of gaming objects. While time manipulation mechanisms attempt to *hide* network latency from players, they may cause negative side effects. In particular, time delay leads to poor responsiveness and time wrap results in game-state inconsistency and irregular moves due to game-state corrections. Therefore, time manipulation mechanisms may not work in networks with long and varying latency.

Matchmaking based latency compensation [10, 28, 29] techniques *prevent* a new player from starting a game with other players that have high expected network latency to that new player. Chambers et al. [28] propose to use IP-to-geolocation databases to filter out far-away players and redirect players to close-by ones. Htrae [10,29] combines IP-to-geolocation databases and network coordinate systems to predict network latency, and prevents players with high network latency from matching. Htrae favors players that are geographically close in proximity, e.g., a player in Japan may never be matched with another player in Canada. The matchmaking based mechanism effectively reduces the number of players each player can play with, and it does not support *specific matches*. Specific matches refer to those games formed before-hand, e.g., among friends. In contrast, IRS enables each player to be matched with more players and supports specific matches.

In summary, latency compensation mechanisms mitigate the network latency issue by either hiding it from players or preventing players with long latency from being matched; whereas the proposed IRS directly reduces the network latency.

2.3.2 Detour Routing for Latency Reduction

The idea of routing game-state updates over detour paths is a kind of application-layer overlay routing. Several overlay networks have been proposed to improve the Internet routing performance from various aspects. For example, RON (resilient overlay network) [30] uses overlay routing to find network paths other than those reported by the Internet routing protocols. RON allows end systems to quickly recover from link congestion and/or path outage. OverQoS [31] uses overlay routing to provide QoS enhancements in the application layer. The main goal of these works is not to reduce network latency for distributed applications.

Several works use graph algorithms to optimize application-level multicast networks. For example, Yang et al. [32] consider the problem of placing proxies in overlay networks to minimize the average or maximum latency between two nodes in the same multicast group. They formulate this problem as an ILP (integer linear programming) problem, and solve it using an ILP solver. Solving ILP problems is computationally expensive, and may not be applicable to dynamic systems such as online games. Vik et al. [33] compare several tree-based and mesh-based algorithms to construct overlay networks connecting all users in a multicast group. These algorithms try to minimize the overall network latency, while connecting all nodes in the graph. However, online game players often form small groups, and network latency between two players in different groups is not important. Furthermore, constructing and maintaining such an overlay network may incur high overhead, and thus is not feasible in online gaming networks. Similar to the IRS system proposed in this thesis, several recent peer-to-peer (P2P) networks, e.g., [34–36], also employ detour paths to reduce network latency.

Several previous works, such as [37], have pointed out that the direct path between two IP addresses may lead to longer network latency than a detour path through a third IP. More recently, overlay routing through detour paths has been used in a few peer-to-peer (P2P) networks, such as [34, 35]. The authors of [34] study the problem of improving voice quality of P2P VoIP systems. They propose a system to use inferred AS maps and ping probes to find detour paths for VoIP sessions. While inferring a complete AS map takes a tremendous amount of time, the resulting AS map may not be accurate [38]. In contrast, IRS employs light-weight network coordinate to find the *optimal* detour path between any two IP addresses rather than an AS map. Moreover, the algorithm in [34] finds the detour paths by performing a breadth-first search from the two peers in a VoIP session, and stops whenever a few detour paths with *good enough* network latency have been found. In contrast, IRS capitalizes on the centralized gaming server, and employs an efficient algorithm to find the *optimal* detour path between any two IP addresses.

Lumezanu et al. [35] propose to construct symbiotic overlay networks, in which every peer associates with other peers only when they can mutually help each other to reduce network latency to some Internet servers. That is, as an incentive mechanism, the system only associates mutual advantage peers that may be on each other's detour paths. This incentive mechanism is not applicable in online gaming networks, in which software is completely controlled by game developers. In addition, while both IRS and the system in [35] employ

decentralized network coordinate systems to find detour paths, their system only performs *local* search for *a* detour path, while we find the globally *optimal* detour path. Moreover, the authors of [35] consider general Internet applications and find detour paths to Internet servers that are not part of their overlay, while the game-state updates are always destinated to other game clients.

In summary, detour routing is similar to overlay routing, which has been shown to improve the performance of several Internet systems and applications, including multicast [32, 33], file sharing, network routing [30, 31], P2P VoIP [34], and cooperative overlay networks [35, 36]. However, these works have objectives different from that of the detour routing in online games.

Chapter 3

Potential of Detour Routing

In this chapter, we explore the potential performance improvements of online multiplayer games using detour routing. We rigorously analyze the expected impact of the detour routing on player performance in different online multiplayer games.

3.1 Measurement Study

In order to conduct our measurement study we need a pairwise RTT dataset of online games to quantify the potential benefits of detour routing. Although network latency is the dominating factor of online gaming quality, there are no publicly available datasets of pairwise RTTs among a large group of game clients. Existing RTT datasets are either sparsely constructed without pairwise measurements, such as the dataset used in [11], or not publicly available due to the proprietary nature of the industry, such as the Xbox dataset used in [10]. Therefore, it is necessary to collect our own RTT dataset with pairwise RTT

Table 3.1: Summary of the measurement.

Description	**Value**
Start Time	4 August, 2009
End Time	17 August, 2009
No. PlanetLab Nodes	551
No. Client IP addresses	28,924
No. Subnets	8,063
No. Responsive Subnets	7,795
Raw RTT Measurements	18,884,321

measurements among representative gaming clients. Table 3.1 summarizes some information about the measurement study. The RTT dataset were collected in two steps, which are detailed as follows.

Collecting IP Addresses of Potential Game Clients. First, we need to identify the IP addresses used by game clients. The `Qstat` utility [39] allows us to achieve this. Qstat is an open source command-line utility that allows users to browse the information of individual gaming sessions, so that they can join the interesting sessions. Qstat supports various gaming protocols, and it works as follows. Once a user specifies the online gaming protocol and lobby server address, Qstat connects to the lobby server and requests a list of current gaming sessions. Each session is represented by its host, which is the client running the game logic. Qstat then iteratively connects to each host, requests session information, and displays the information to users. It is worth noting that Qstat does not give the IP addresses of *all* clients in gaming sessions, instead only the host IP addresses are returned. This is because modern online gaming protocols prevent hosts from disclosing client IP addresses to other clients, in order to avoid possible denial-of-service (DoS) attacks. In this step, Qstat is used to collect the IP addresses of all hosts. The IP addresses can represent all gaming clients. It is reasonable to assume the set of collected IP addresses is representative for two reasons. First, servers may also be gaming clients. Second, almost all online games allow players to *host* sessions, and become the servers themselves when there are too few hosts in the near-by network.

After running `Qstat` several times on many machines with various arguments, there were two striking observations. First, the lobby server `h2master.streampowered.com` of *Counter-Strike: Source* returned the largest number of IP addresses during the experiments. Therefore, this lobby server was used throughout the experiments. Second, the lobby server returns different sets of IP addresses to `Qstat` running on different machines. A possible explanation for this is that the lobby server implements a matchmaking algorithm that only returns close-by IP addresses in the sense of geographical and/or network proximity. In order to collect IP addresses of game clients around the world, we need to run `Qstat` at many locations. More specifically, `Qstat` was deployed on more than 550 PlanetLab nodes, and each PlanetLab node ran `Qstat` 60 times. After combining all collected IP addresses together, it yielded 28,924 distinct game client IP addresses.

Measuring RTTs among Clients. Next, we describe the process for measuring the RTT between any two client IP addresses. Since we have no control over the game

clients, the pairwise RTT measurements must be done using the `King` utility [40]. `King` supports measuring the RTT between any two arbitrary IP addresses, and is built upon two observations. First, most workstations are close to their authoritative DNS (domain name system) servers. Second, many DNS servers support recursive queries, which can be used for RTT estimation. Modifications were made to the original King utility to allow it to log the Unix timestamp of each measurement.

Measuring all pairwise RTTs among the considered 28,924 client IP addresses would take a prohibitively long time. To accelerate the measurement process without losing accuracy, the client IP addresses were clustered into traditional *Class C* (/24) subnets. Measurements are then taken between each pair of subnets. We then use the RTT between two subnets as the RTT between any two gaming client IP addresses in these two subnets. We can do this without affecting the accuracy because `King` measures RTT between two clients using their authoritative name servers, and clients on the same Class C subnet most likely share the same authoritative name server. The clustering process resulted in 8,063 subnets. For each subnet, a random client IP is chosen to be a representative for a subnet. The representative client is used to conduct pairwise RTT measurements among 8,063 other representative client IP addresses. Client IP addresses within the same subnet were excluded from the measurements. Absence of these measurements, however, does not bias the quantification of the potential of detour routing. This is because clients on the same subnet are *unlikely* to be on each other's detour paths.

Even after the clustering, the number of required RTT measurements is still large, and conducting these measurements from a single machine takes a long time. Furthermore, running too many copies of `King` on several workstations on the same subnet may incur high measurement errors, because a large number of query packets can lead to network congestion. To conduct the RTT measurements with high accuracy, scripts were developed to run `King` on the 550+ PlanetLab nodes mentioned above for two weeks. On every PlanetLab node, the script ran 12 measurement processes. Each process repeatedly measured the RTT of two randomly chosen representative client IP addresses, and wrote the successful measurement results into a data file. The measurement processes worked independently, and the data files were merged before the analysis. Over this two-week experiment, a total of 18,884,321 RTT measurements were collected, equivalent to about 230 GB of raw data.

Data Processing and Trace Creation. The collected raw data did not contain RTT measurements for some subnet pairs. Only 7,795 out of 8,063 subnets had RTT

measurements. This is mostly because not all `King` measurements were successful, since `King` relies on recursive queries to estimate RTTs, and not all name servers support recursive queries. The experimental study in [40] indicates that only about 76% of web server and 79% of P2P file sharing clients have authoritative name servers that support recursive queries. Next, the RTT measurements were sorted. In the case where a pairwise subnet had more than one RTT result, the median RTT was used. This resulted in 13,414,831 distinct pairwise RTTs. These pairwise RTTs are stored in a matrix, which in this thesis we refer to as the *RTT matrix*.

3.2 Notation and Performance Metrics

In this section, we define the notation and performance metrics used in the experiments.

3.2.1 Notation

For a pair of clients s and t, let $d(s,t)$ be the RTT value between them. Next, consider a realistic relay overhead $\alpha = 24$ msec as suggested by Ren et al. [34]. We then define $T_k(s,t)$ as the minimum RTT of all k-hop detour paths between clients s and t, where $k \geq 0$. More precisely, we define $T_k(s,t)$ by induction as:

$$T_k(s,t) = \begin{cases} d(s,t), & k = 0; \\ \min\limits_{\forall \text{ client } r} \Big\{ d(s,r) + \alpha + T_{k-1}(r,t) \Big\}, & k \geq 1. \end{cases} \tag{3.1}$$

For any $k \geq 0$, we define $T_k^*(s,t)$ as the k-hop shortest distance between s and t, which has the minimum RTT among all detour paths with up to k intermediate clients. Specifically, the k-hop shortest distance can be written as:

$$T_k^*(s,t) = \min_{0 \leq k' \leq k} \Big\{ T_{k'}(s,t) \Big\}, \tag{3.2}$$

where $k \geq 0$. Last, we use $T^*(s,t)$ to denote the k^*-hop shortest distance, which is the absolute minimum RTT among all possible detour paths with any lengths.

Mathematically, we write:

$$T^*(s,t) = \min_{k \geq 0} \Big\{ T_k^*(s,t) \Big\}. \tag{3.3}$$

The RTT matrix and Eq. (3.2) are used to recursively derive the 1, 2, and 3-hop shortest distances. A modified Dijkstra's algorithm that takes relay overhead α into consideration was used to compute k^*-hop shortest distance.

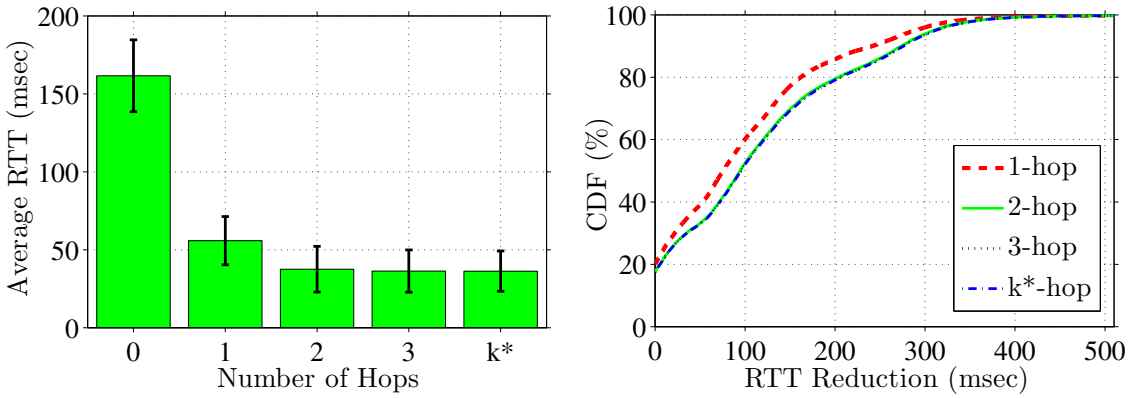
3.2.2 Performance Metrics

Next, we define the performance metrics used to evaluate the potential of detour routing as follow.

Pairwise RTT Reduction. The pairwise RTT reduction between clients s and t is defined as the RTT difference between their k-hop shortest distance and their direct distance. To derive the overall k-hop (or k^*-hop) RTT reduction, we can simply iterate through *all* pairs of known clients, and compute the shortest distance for each of them. This, however, may take a prohibitively long time. To cope with this computational complexity, a sample set of client pairs is used. That is, a random set of 500,000 pairs of clients are chosen and used to compute 0-hop (direct path), 1-hop, 2-hop, 3-hop, and k^*-hop shortest distances for each pair. k-hop shortest distances with $k \geq 4$ is not considered because there was no clear improvement when increasing k from 3 to 4. This is most likely because of the significant overhead incurred by 4-hop, $\alpha = 4 * 24 = 96$ msec. When choosing the random client pairs, only those pairs with valid RTT values were considered. This is because it is not possible to fairly compute the RTT reduction of client pairs with no direct distance.

Session RTT Reduction. In online games, several clients join a game session, and the gaming quality is determined by the client with the *highest* RTT to the host. This is because each client needs to send game-state updates to the host and wait for validated updates. Therefore, clients with smaller RTTs may suffer from long RTTs of other clients. To quantify the potential RTT reduction in real gaming sessions, we analyze the session information reported by Qstat. The distribution of number of players per session is calculated and used in calculating the session RTT reduction.

Reachability. Reachability is defined as the number of players a game client can connect to, and maintain good gaming quality. As mentioned in Chapter 2, different game types have different thresholds on RTTs. These thresholds in turn define the reachability of a client. For each threshold, we iterate through every client, and identify all other clients that have RTT values shorter than the threshold to the subject client. These clients are defined as *reachable* clients. This process is repeated for 0-hop (direct), 1-hop, 2-hop, 3-hop, and k^*-hop shortest distances.



(a) Average RTT reduction.

(b) RTT reductions.

Figure 3.1: Distribution of RTT reduction.

3.3 Results for Potential Improvements

This section presents the potential of detour routing using the RTT matrix described in Chapter 3.1.

RTT Reduction. Fig. 3.1(a) illustrates the computed average RTT among all sample pairs for 0-hop (direct), 1-hop, 2-hop, 3-hop, and k^*-hop. The confidence intervals of 95% are also shown on the graph. This figure shows that a significant reduction in the average RTT is possible using detour routing. For example, with only one relay node, the average RTT drops from about 160 msec to less than 60 msec. The figure also shows that most of the gain is achieved by using only one relay node. Next, we compute the CDF (cumulative distribution function) of the RTT reduction across all player pairs. The results are shown in Fig. 3.1(b) for 0-hop, 1-hop, 2-hop, 3-hop, and k^*-hop detour routing. A conservative approach is used in choosing the relay overhead, with $\alpha = 24$ msec in this figure. This overhead value is conservative because one could optimize the implementation of the game software to do much faster processing of the relayed or tunnelled through packets. In Chapter 4.2 we will show that an actual relay delay as low as 6.2 msec is achieveable. The results in Fig. 3.1(b) illustrates that 80% of player pairs observe RTT reduction using detour routing. Most importantly, the figure shows that about 40% of player pairs observe at least 100 msec RTT reduction with just 1-hop. Since the minimum RTT requirement for first-person avatar games is 100 msec, this is a significant reduction in the RTT among players. In summary, Fig. 3.1 shows that using detour routing significantly reduces RTTs among

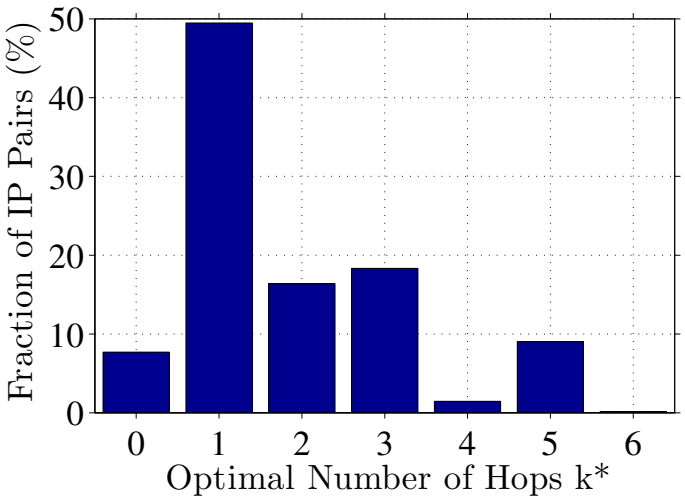
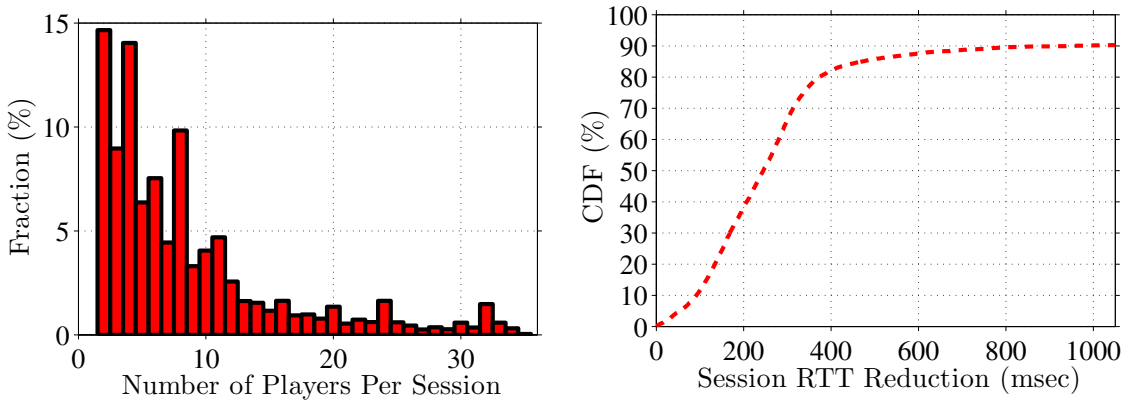


Figure 3.2: Distribution of the optimal number of hops in detour paths.

gaming clients.

Optimal Number of Hops (k*). The optimal number of hops is defined as the number of relay nodes traversed by the optimal detour path. The optimal number of $k*$ hops is computed using Dijkstra's shortest path algorithm implemented in the Boost Graph Library (BGL) [41]. We choose to use BGL because it has a Perl interface that is easily integrated into our scripts. Fig. 3.2 illustrates the distribution of the number of hops in the optimal detour paths. The results shows that almost 50% of optimal paths can be found within 1-hop.

Session RTT Reduction. As mentioned, the pairwise RTT reduction results presented above are very *conservative* because we assume each pair of clients forms a gaming session. In reality, many clients join a session, and the gaming quality is determined by the client with the *highest* RTT to the authoritative server. Hence, the potential of detour routing should be even more significant. To quantify the potential RTT reductions in real gaming sessions, we also need to analyze the session information reported by Qstat. Fig. 3.3(a) illustrates the distribution for the number of players per gaming session. Using this distribution, 10,000 gaming sessions were simulated with different numbers of players. The computed expected session RTT reduction using 1-hop detour routing is shown in Fig.3.3(b). This figure clearly shows that 1-hop detour routing achieves more than 100 msec RTT reductions in 90% of the gaming sessions!



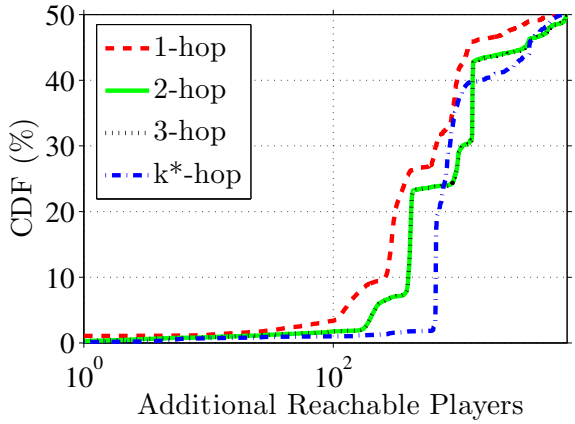
(a) Distribution of players over sessions.

(b) Session RTT reduction.

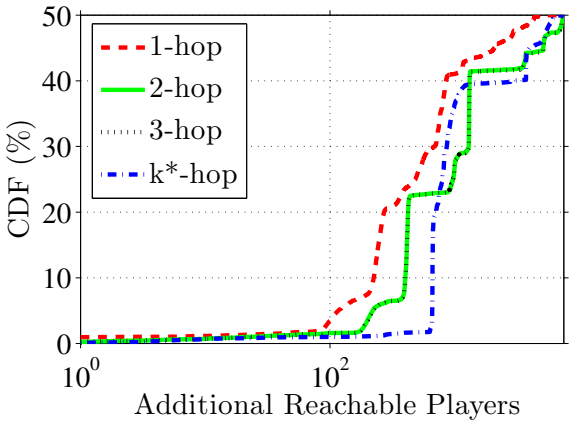
Figure 3.3: Session RTT reduction due to detour routing.

Reachability. As mentioned in Chapter 2.1, different types of games have different thresholds on RTTs, and these thresholds in turn define the reachability of clients. In order to study the reachability we need to compute the reachability of various RTT thresholds: 50, 100, and 200 msec. For each threshold, we need to iterate through every client IP address, and identify all other client IP addresses that have an RTT value shorter than the threshold to the subject client. We define these clients *reachable* clients, and count the number of such clients. This process is repeated for 0-hop (direct path), 1-hop, 2-hop, 3-hop, and k^*-hop. Next, the difference between the results achieved by k-hop (k>0) detour routing and direct paths (0-hop) is computed. Fig. 3.4 shows the CDF curves for 50 msec, 100 msec, and 200 msec thresholds. These figures show, for example, with 200 msec threshold 60% of clients can reach at least 100 additional players with just 1-hop detour routing. This means that employing detour routing helps players to find more potential players to start gaming sessions, while maintaining good gaming quality. This is in contrast to previous works on player matching, such as [10, 11], which aim to *constrain* players' reachability to maintain reasonable gaming quality.

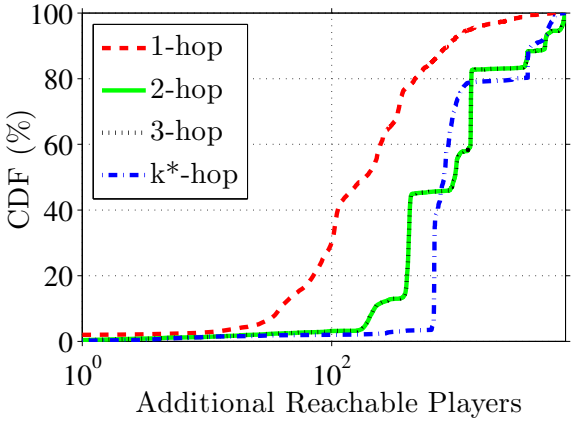
Optimality and Complexity Trade-off. Despite the advantages discussed above, it is not easy and at times costly to construct optimal overlay routing [33]. There are several reasons that make overlay routing not practical: (i) to find an optimal detour path we must construct a complete graph and explore all possible paths, (ii) the number of players participating in an online game network can exceed hundreds of thousands [16], and (iii)



(a) 50 msec threshold



(b) 100 msec threshold



(c) 200 msec threshold

Figure 3.4: Additional players allowed by detour routing.

Table 3.2: Expected player performance in first-person shooter games. Based on data from [1].

Latency (msec)	Hit Ratio	Kill Count	Death Count
50	49%	41.00	10.75
75	54%	39.75	14.00
100	42%	38.25	16.25
200	29%	35.75	16.75
250	29%	27.75	17.00
300	19%		

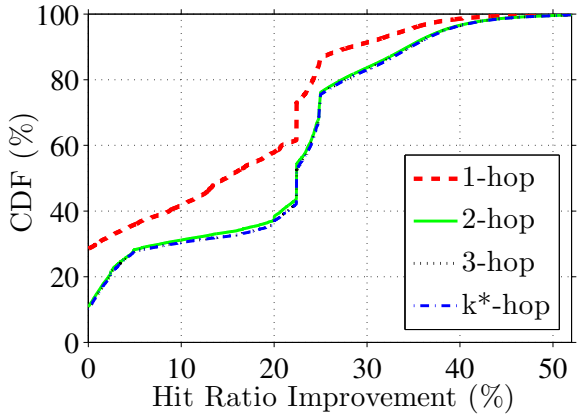
the tremendous amount of computing power required to deal with a dense graph this size is not within reach of the general public. The task is, hence, to find the best trade-off between the specified RTT reductions and the complexity of the algorithm.

3.4 Impact on Actual Games

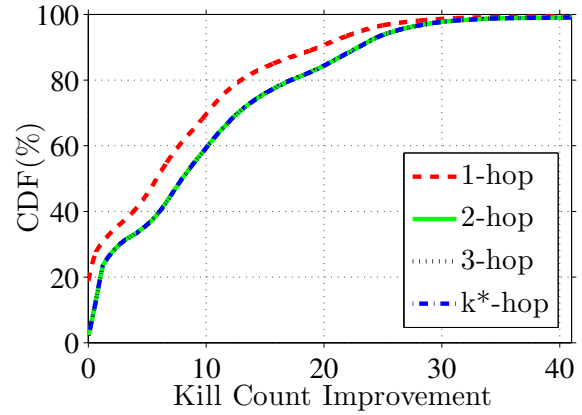
Player performance is an indication for gaming quality: low frame rates, sluggish responsiveness, and irregular moves affect players' ability to interact with other players, and thus often lead to worse player performance. The player performance metrics are defined in the context of each game type, and could be quite different from one to another. For example, higher shooting accuracy in first person shooter games is important, while shorter finish time in car racing games is desired. Furthermore, the same game type may have multiple player performance metrics. Many subjective user studies, e.g. [1–3, 16, 42, 43], define performance metrics for various games. In this section, we will analyze the impact of detour routing on several player performance metrics defined in previous works. We do not conduct a new subjective study. Rather, we build on the extensive results already available in the literature, and quantify the potential gain using these existing metrics.

3.4.1 First-Person Shooter Games

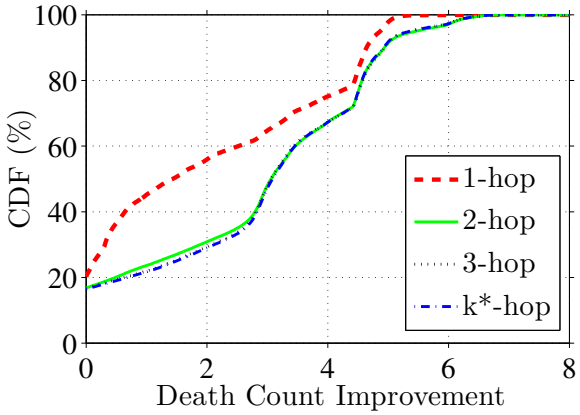
Beigbeder et al. [1] study the effect of network loss and latency on player performance in Unreal Tournament, which is a popular first person shooter game. In first person shooter games, each player controls an avatar, and sees a virtual world through its eyes. Players move in the virtual world, and try to kill other players and/or bots controlled by computer



(a) Hit Ratio



(b) Kill



(c) Death

Figure 3.5: Improvements in first-person shooter games achieved using detour routing.

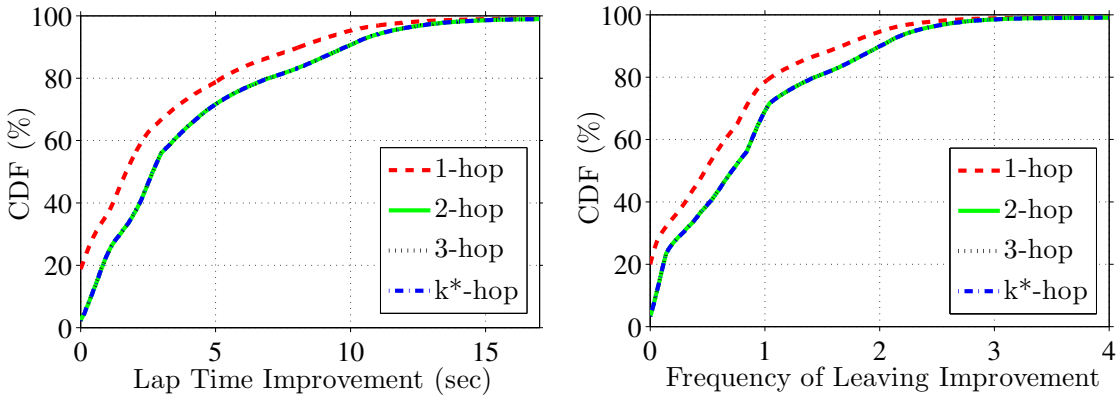
algorithms. There are several modes for the Unreal Tournament game. The base mode is called *Deathmatch*, where each player tries to kill as many other players as possible. If a player is killed in a Deathmatch game, he/she would rejoin the game for a limited amount of time. The score of each player is determined by the number of players he/she kills. Players have a wide selection of weapons to use. These weapons can be classified into high, medium, and low precision. High precision weapons are more vulnerable to network latency, as small aiming inaccuracy can easily lead to missed shots.

Based on [1], three player performance metrics were chosen that are most affected by the network latency. First, we consider *Hit Ratio*, which is the ratio of hit shots over the total fired shots. The hit ratio is measured using high precision weapons during 10-minute games. Next, we consider *Kill* and *Death* metrics, which are the number of enemies killed by the player and the number of deaths a player observed, respectively, in 5-minute Deathmatch games. Table 3.2 illustrates the average player performance under various network latency values, which is derived from the user study in [1]. This table shows that Death count increases, while Hit Ratio and Kill count decrease when network latency increases. The data in this table were used as reference to interpolate and extrapolate the expected impacts on player performance. Fig. 3.5 illustrates the impact of latency reduction on the player performance in first-person shooter games. In particular, Fig. 3.5(a) depicts the improvements in the Hit Ratio. This figure shows that about 60% of players gain at least 10% hit ratio improvements for 1-hop, and some of them can improve their hit ratio by as much as 40%. Fig. 3.5(b) shows the improvements in the player's ability to kill others. This figure shows that using just 1-hop detour routing, 70% of players can improve their Kill count. Lastly, Fig. 3.5(c) shows a similar improvement in Death count.

3.4.2 First-Person Car Racing Games

Pantel and Wolf [2] explore the impacts of latency on player performance in the Virtual RC Racing game. In car racing games, each user drives a car running on a racing track for several laps. The goal is to finish a target number of laps as soon as possible. To achieve that, players need to follow the racing track as closely as possible, because missing the track means sudden speed reduction and higher chances for collisions. High network latency results in irregular frame updates, which in turn increases the chance for players to leave the track, and thus have a longer finish time.

Using a subjective study, Pantel and Wolf [2] summarize the players' gaming experience



(a) Lap Time

(b) Frequency of Leaving

Figure 3.6: Improvements in first-person racing games achieved using detour routing.

Table 3.3: Expected player performance in car racing games. Based on data from [2].

Latency (msec)	**Lap Time (sec)**	**Frequency of Leaving Track**
0	7.50	0.05
50	8.00	0.20
100	9.00	0.50
150	10.00	0.90
200	12.00	1.10
250	14.00	1.50

as follows. With RTT at 50 msec, players are not aware of any imposed latency. At 100 msec, players can feel the unresponsiveness when steering, but do not observe rendering issues. At 200 msec, players clearly see the frame rate dropping, and the cars are harder to control. Finally, at 500 msec, the gaming quality becomes so bad, and players would rather stop playing. This subjective study clearly indicates network latency has great impacts on the user satisfaction in racing games.

For objective metrics, two player performance metrics were selected from the study in Pantel and Wolf [2]. First, we consider *lap time*, which is the average time a player finishes a lap. Each player runs five laps, and the average lap time is computed across all users in the user study. Second, we consider *frequency of leaving the track*, which is defined as the number of times a player accidentally leaves the track for each lap. This is computed by replaying each player's five lap race to count the number of times he/she leaves the

Table 3.4: Expected player performance in MMOPRG. Based on data from [3].

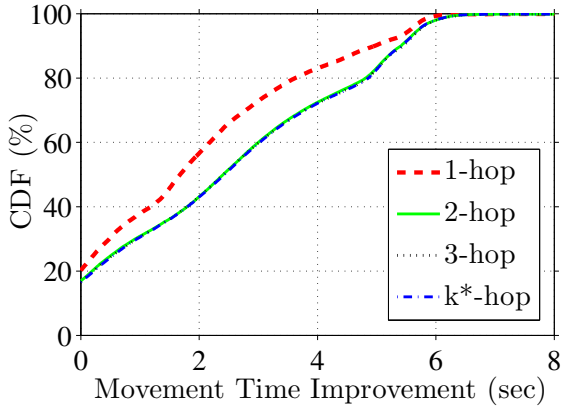
Latency (msec)	**Movement (sec)**	**Combat (sec)**	**Health (%)**	**Mana (%)**
0	204	108	90	45
250	210	112	90	35
500	211	120	72	22.5
750	212	130	80	10
1250	220	155	60	22.5
2000	233	151	35	20
3000	222	167	31	10
5000	244	184	22.5	0

track. Table 3.3 summarizes the player performance under different network latency values, which is extracted from the figures in Pantel and Wolf [2]. The data in this table is used to calculate the expected impacts on player performance. The results are shown in Fig. 3.6. Fig. 3.6(a) illustrates that for 1-hop detour routing 70% of players achieve 1 sec or more lap time reduction, while 20% of players achieve 5 sec or more. This is a significant reduction as typical lap time in racing games is less than 14 sec and the winning lap time is often within 1 msec of the 2^{nd} place [16]. Furthermore, this figure reveals that 80% of players can experience reduction in lap time. Fig. 3.6(b) shows significant reductions in the frequency of players leaving the game due to poor latency. It shows that with 1-hop detour routing the player leaving frequency can be reduced by at least 1 in 20% of the cases.

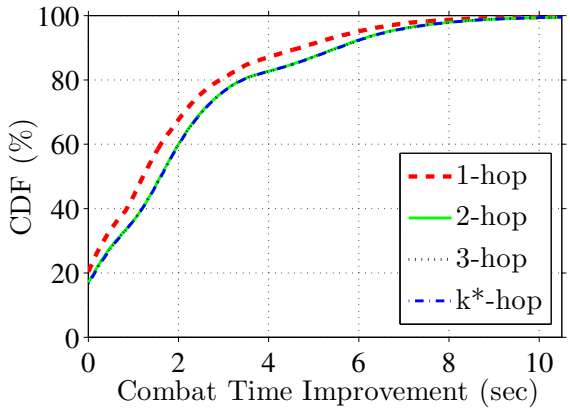
3.4.3 Third-Person Avatar Games

Fritsch et al. [3] study the impact of network latency on player performance in Everquest 2, which is a massive multi-player online role-playing game (MMORPG). Everquest 2 supports many concurrent players by dividing the virtual world into multiple zones, which allows the server to implement load balancing. Players in Everquest 2 do not have a clearly defined goal, and their general objectives are to gain experience points, upgrade to the next level, get better equipment, and enter restricted areas. The performance of an Everquest 2 player is largely determined by how fast he/she can gain experience points, which is correlated to how fast the player can move as well as take down monsters. Smaller network latency allows players to move faster and fight more efficiently.

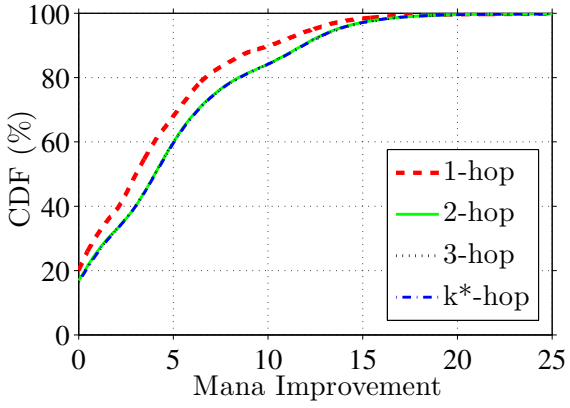
The performance metrics in [3] are based on two challenges: Movement and Combat.



(a) Movement time



(b) Combat Time



(c) Mana

Figure 3.7: Improvements in third-person avatar game Everquest 2 achieved using detour routing.

Table 3.5: Expected Player Performance in Sports Games.

Latency (msec)	**Attempt Gain (yards)**
0	4.85
125	4.60
250	5.00
500	4.75
750	3.75
1000	3.25
1500	3.00
2000	3.30

In Movement challenge, players must kill three monsters and then run through a hostile area to a destination. In Combat challenge, players must kill a fairly strong monster. Four performance metrics from the work in [3] were choosen as reference in computing the expected impacts on player performance. First, we consider *Movement Time*, which is the time by which a user completes the movement challenge. Second, we consider *Combat Time*, which is the time by which a user kills the monster in the combat challenge. Third, we consider *Health* and *Mana*, which are the residue of the health and magic points, respectively, after a user finishes the monster challenge. It is worth noting that shorter Movement and Combat times indicate that the user can move faster, and high Health and Mana values mean that the user can fight more efficiently. Table 3.4 summarizes the expected player performance under various network latency values, which is extracted from figures in [3]. The results are presented in Fig. 3.7. Fig. 3.7(a) illustrates the improvement in player's movement time. This figure shows that with 1-hop detour routing, 65% of the players can reduce Movement Time by at least 1 sec. Fig. 3.7(b) shows the Combat Time improvements. Increased latency decreases a player's ability to cause damage, thus it increases the time it takes to complete a fight. This often causes frustration to the player. This figure shows that with detour routing, 30% of players can achieve Combat Time reduction of 2 sec or more. More significantly, some players can achieve as much as 8 sec in Combat Time reduction. Fig. 3.7(c) illustrates improvement in the player's Mana. It shows that 50% of the players can increase their Mana by at least 4%. Similiar results were also observed for Health.

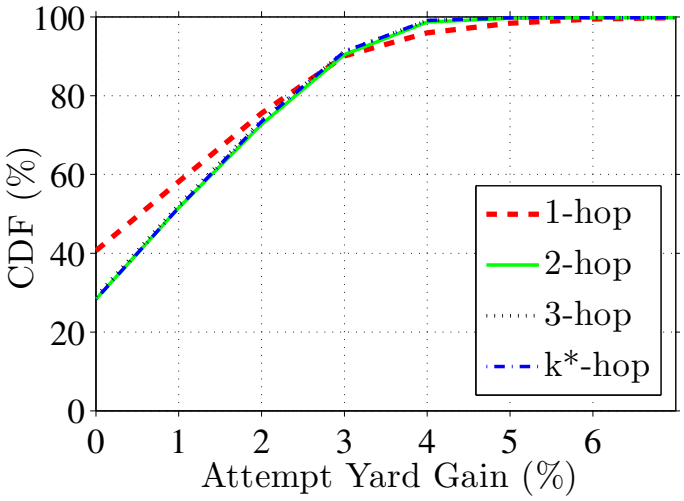


Figure 3.8: Yard improvements in Madden NFL Football achieved using detour routing.

3.4.4 NFL Football Game

Nichols and Claypool [42] study the impact of network latency on Madden NFL Football, which is a network game running on Sony PlayStation 2. Two main player performance metrics are identified in this game: Running and Passing performance. Only the Running performance has been quantified in the study in terms of average attempt gain in yards. The experimental results show that longer network latency results in smaller yards gained, and the precise mapping is extracted from the figures in [42] and given in Table 3.5. This table shows that the attempt gain performance decreases when network latency increases, and as high as $(4.85 - 3.30)/4.85 = 30\%$ can be achieved. The data in this table is used to compute the expected impact on player performance, and plot the results in Fig. 3.8. This figure depicts the relative improvement on the number of yards gained per attempt. The figure shows that 40% of players achieve 1% or more of improvement.

3.4.5 Omnipresent Games

Claypool [43, 44] explores the effect of network latency on player performance in several omnipresent real-time strategy (RTS) games: Warcraft III, Age of Mythology, and Command and Conquer. In online RTS games, several players start a game session on a map agreed upon by them. Players first harvest resources, such as lumber and oil, and then build various types of buildings. Constructing and upgrading buildings allows a player to produce

certain military units and/or develop new weapon technologies. In addition to buildings, players also produce military units in order to defend his/her territory, occupy and control larger portions of the map, and invade other players' territories. Players of RTS games can form allied forces, or simply play free-for-all games. A player exits the game if all his/her buildings are destroyed or seized, and the game terminates when there is only one player left, who becomes the winner of the game.

Claypool [43] identifies three basic tasks as the player performance metrics in RTS games, which are build time, exploration time, and combat score. *Build time* refers to the time used by players to construct all types of buildings and research all technologies. Shorter build time usually means faster production of troops, which leads to better chances of defeating other players. *Exploration time* refers to the time required by a player explores the whole map. Players need to explore the map because, in RTS games, the parts of maps where his/her own individual units does exist are not visible to the player. Shorter exploration time results in more response time for battles. *Combat score* is the score difference between two players after battles, where only one player is under imposed network latency.

The experimental results reveal that the correlation between these three performance metrics (in all three RTS games) and the network latency is fairly weak. Moreover, even long network latency imposes relatively insignificant impact on these metrics. This is probably because the player performance depends more on the chosen strategy, and players can quickly adapt to lags caused by long network latency in these relatively slow paced games. For these reasons, in this thesis we do not consider the player performance in RTS games.

3.5 Summary

In this chapter, we rigorously analyzed the potential of using detour paths in various online multiplayer games to reduce network latency among players and the impact of this RTT reduction on the actual player performance of different types of online games. Futhermore, the results show that with detour routing, players can join online game sessions that were not available to them because of the long network latency of the direct paths. In particular, the results show that more than 40% of players can observe 100 msec or more RTT reduction by routing game-state updates through 1-hop detour paths. This suggests that simple 1-hop detour paths can achieve most of the benefits resulting from reduced latency.

Chapter 4

Indirect Relay System

In the previous chapter we showed the potential of using detour routing to reduce latency in online multiplayer games. This chapter presents the Indirect Relay System (IRS) which uses 1-hop detour routing, and analyzes its performance from several angles.

4.1 Overview

At first glance, creating a complete mesh among players in the same group seems to minimize the network latency in online games. This, however, is not true because the Internet routing is *not* optimal in terms of network latency [37]. Thus, sending updates directly from a player to another player may lead to longer network latency than sending them through a few *relay* players. For illustration, Fig. 4.1 shows several players in an online game, in which edges represent network connections, and they are annotated with their RTT (round-trip time) values. In this figure, observe that the triangle of players C1, C5, and H1 violates the triangle inequality. That is, the length of the side (C1, H1) is longer than the sum of the other two sides (C1, C5) and (C5, H1). This is called a triangle inequality violation (TIV). It is clear that routing game-state updates from C1 to H1 through C5 leads to shorter end-to-end RTT than directly sending these updates from C1 to H1. The path C1-C5-H1 is called a *detour path*. Recent works, such as [19], report that TIVs are not due to measurement errors, and more than half of arbitrarily chosen IP pairs belong to some TIVs [45]. Therefore, detour paths with shorter RTTs than direct paths can easily be found in the Internet.

One way to find the best k-hop detour path is to iterate though all possible relay clients. This approach, however, incurs high measurement overhead. This is because RTT values of

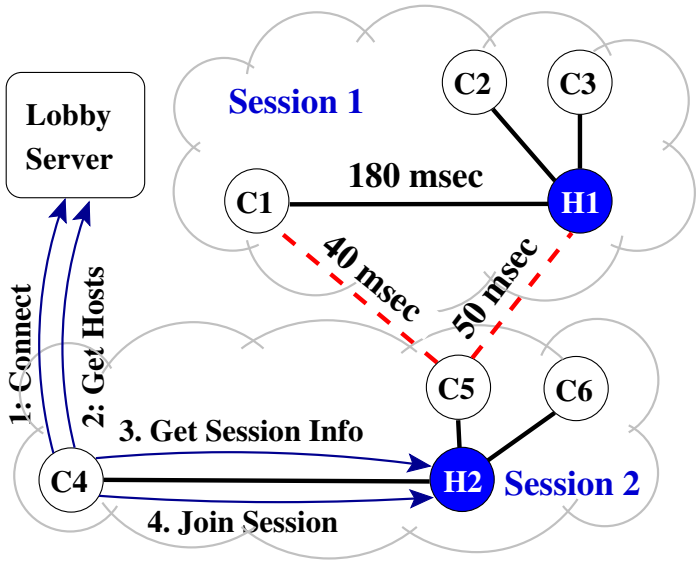


Figure 4.1: Game sessions in online game systems.

too many client pairs need to be measured, which may not be feasible in online games. To reduce the number of RTT measurements, we may consider network coordinate systems. A network coordinate system assigns each client a point in a coordinate space, such that computing the distance between the coordinates of two points gives the RTT *estimate* between the clients associated with these two points. The coordinates of each client are derived from a few RTT measurements between that client and its *neighbors*, which are chosen by a bootstrap service when the client joins the coordinate system. With coordinates, we can compute RTT estimates between any pair of clients when searching for the best detour path. This significantly reduces the measurement overhead.

One may think that finding detour paths using network coordinates can be as simple as follows. We derive the coordinates of individual clients, and compute the RTT estimates between any two clients using their coordinates. Next, use the RTT estimates to find the k-hop optimal detour path. This simple approach, unfortunately, does *not* work, because most of the coordinate spaces satisfy the triangle inequality [46]. Thus, RTT estimates computed using network coordinate systems form *no* triangle inequality violations (TIVs). Lumezanu et al. [35] observe that because network coordinates cannot properly embed RTT measurements with TIVs into the resulting coordinates, the RTT estimates computed using coordinates of any two points of a TIV would show a nontrivial estimation error. This

means that by checking the estimation errors between any two clients, we can determine whether the link between them is part of a TIV. This enables us to *indirectly* use network coordinate systems to locate detour paths without imposing high measurement traffic.

The proposed Indirect Relay System (IRS) utilizes the triangle inequality in the Internet to form detour paths for faster delivery of game-state updates. Consider a gaming network with a lobby server and M clients. Let $D(s,t)$ be the RTT measurement between any two clients s and t, where $1 \leq s,t \leq M$. Let $O(r)$ be the round-trip relay delay of client r, where $1 \leq r \leq M$. We say that client r leads to a detour path from s to t if and only if $D(s,r) + O(r) + D(r,t) < D(s,t)$. The goal of the IRS system is to efficiently find detour paths between two clients s and t, and utilize the detour paths to reduce RTT between them. To achieve this, the IRS system supports the following three operations between s and t:

1. Identify up to K most promising relay clients using a network coordinate system, where K is a system parameter.

2. Rank these potential relay clients based on end-to-end RTT measurements, which allow client s to find the best detour path to reach t.

3. Monitor the network and relay client conditions and dynamically switch detour paths if the active one is congested or the relay client fails.

The IRS system has two components: IRS Server and IRS Client. The IRS Server is implemented as a module in the lobby server to manage coordinates of clients and assist clients to utilize detour paths in order to reduce the RTT between any two clients. The IRS Client implements a network coordinate system and runs on game clients. The IRS system can work with any network coordinate system, such as Vivaldi [46]. Each game client c maintains a neighbor set $\mathbf{n}_c$, and randomly probes clients in $\mathbf{n}_c$. The client then adjusts its coordinates based on the RTT measurements and the coordinates of its neighbors. The number of neighbors of each client N is a system parameter. Client c periodically (every T sec) sends updates of its coordinates ($\mathbf{x}_c$) and RTT measurements ($D(c,n)$, where $n \in \mathbf{n}_c$) to the IRS server. This enables the IRS server to maintain a current view of the gaming network, and to determine the likelihood of any two clients being part of a detour path. Then, the IRS server uses an efficient algorithm to find the most promising detour paths between two given clients, which is presented in the next section. The overhead of the

algorithm is controlled by heuristically setting thresholds on the changes in the coordinates (Δx) and RTT measurements (Δd) below which the updates are not sent.

4.2 Design of the IRS

In this section, we first show the steps to identify potential detour paths between any two gaming clients. Next, we explain the employed ranking procedure and present the dynamic management of detour paths. We also discuss the handling of security concerns. Last, we give high-level pseudocode, and analyze its complexity.

4.2.1 Identifying Potential Detour Paths

To identify potential detour paths, one approach is to employ network coordinate systems, which enable us to derive pairwise RTT measurements without imposing significant probing overhead. A network coordinate system assigns each client a point in a coordinate space such that computing the distance between the coordinates of two points gives the RTT estimate between the clients associated with these two points. The coordinates of each client are derived from a few RTT measurements between that client and its neighbors, which are chosen by a bootstrap service when the client joins the coordinate system or through gossip protocols.

Our approach is to employ an indirect way to use network coordinate systems in order to identify potential detour paths. This method is based on the following observation, which is also used in [35]. Since network coordinates cannot properly embed RTT measurements with TIVs into the resulting coordinates, the RTT estimation of two points of a TIV would suffer from a nontrivial estimation error. For example, Fig. 4.2 shows a TIV between C1, C5, and H1, where the numbers next to the links are RTT estimations and the numbers in parentheses are estimation errors. The same TIV is also shown in Fig. 4.1 with real RTT measurements. We first consider the long side (C1, H1), its RTT measurement is abnormally long from the perspective of the network coordinate system, and thus the RTT estimation should be shorter than the RTT measurement, or equivalently the estimation error should be a nontrivial negative value. Otherwise, this TIV is *successfully* embedded by the network coordinate system, which is *impossible* because coordinate spaces satisfy the triangle inequality. Similarly, consider the short sides (C1, C5) and (C5, H1), their RTT measurements are abnormally short from the perspective of the network coordinate system,

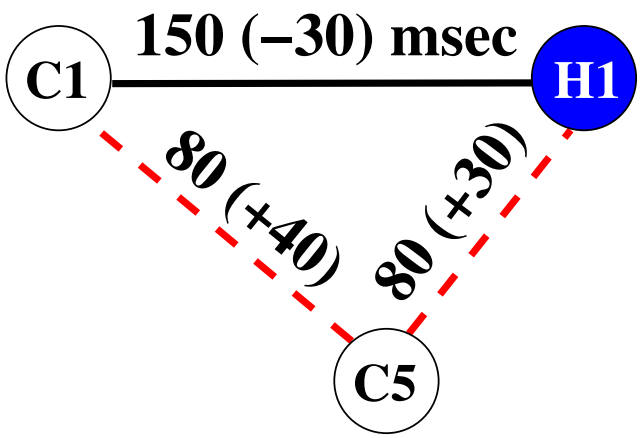


Figure 4.2: Locating TIV using network coordinate systems.

and thus the RTT estimates should be longer than the RTT measurements, or equivalently the estimation errors should be nontrivial positive values. This is shown in Fig. 4.2, where the link between C1 and H1 has a negative estimation error of -30, while the other two links have positive estimation errors of $+40$ and $+30$.

The component that identifies detour paths using the above observation runs on the IRS server. It uses the RTT measurements and network coordinates collected from the clients to find the most promising relay clients. First, we define the relay candidates $\mathbf{r}$ as the set of all clients whose RTT measurements from s or t were previously reported to the IRS server. That is, a client r is in $\mathbf{r}$ if and only if $D(s,r)$ and/or $D(r,t)$ are known to the IRS server. For a given pair of s and t, the IRS server evaluates the *likelihood* of each client r in $\mathbf{r}$ for being the best relay client of the detour path between s and t using the likelihood function:

$$\hat{E}(r) = \begin{cases} E(s,r) = D'(\mathbf{x}_s, \mathbf{x}_r) - D(s,r), \text{ if } D(s,r) \text{ is known;} \\ E(r,t) = D'(\mathbf{x}_r, \mathbf{x}_t) - D(r,t), \text{ if } D(r,t) \text{ is known;} \\ \frac{E(s,r)+E(r,t)}{2}, \text{ if } D(s,r) \text{ and } D(r,t) \text{ are both known,} \end{cases} \tag{4.1}$$

where $D'(\mathbf{x}_s, \mathbf{x}_r)$ is the estimated RTT between s and r using their network coordinates $\mathbf{x}_s$ and $\mathbf{x}_r$. Based on the aforementioned observation on TIVs, relay clients with higher likelihood function values have higher chances to be on better detour paths. The IRS server uses the likelihood function to find the K most promising relay clients.

4.2.2 Ranking Detour Paths

While the IRS server maintains historical RTT measurements to identify potential detour paths, these RTT measurements may be out-dated due to network dynamics. Fortunately, this problem can be mitigated by conducting on-demand, end-to-end, RTT measurements through the K potential relay clients from s to t. Other than more up-to-date measurements, conducting actual end-to-end RTT measurements has an additional benefit. These end-to-end measurements allow us to factor in the round-trip relay overhead $O(r)$, which is dynamic and depends on the current load of the relay client r. This in turn allows us to avoid overloading clients with limited resources as these clients have high $O(r)$ values, and thus high end-to-end RTT measurements. When the end-to-end RTT measurements are done, the source client s ranks the potential detour paths in ascending order of their RTTs. It then uses the first detour path as the active detour path, and keeps other detour paths as backups.

Notice that the RTT measurements need not be empty probing packets. Instead, actual game-state updates may be used for measuring RTTs, which reduces the network overhead incurred by the IRS system.

4.2.3 Relay Overhead

Sending game-state updates through a relay game client leads to additional latency and traffic overheads. To quantify actual relay overheads, two popular multiplayer online games `Starcraft 2` and `Counter Strike: Source` were used to measure overheads. The setup of our experiments is illustrated in Fig. 4.3. First, the games were played on a commodity PC with 2.8 GHz Intel CPU for 30 minutes, and the game-state updates were captured and structured into a trace file. Next, a traffic generator is used to replay the captured game-state updates toward the PC, and at the same time a new 30 minutes game session is played. A relay utility that receives game-state updates from and sends them back to the traffic generator is used to emulate the packets forwarding functionality. This utility measures the latency overhead as the difference between the time an update arrives at the PC's network adapter and the time it goes onto the network adapter's outgoing queue. Concurrently running the online multiplayer game and relay utility on the same PC allows us to measure realistic overheads. The results of this experiment shows an average latency overhead of $O(r) = 6.2$ msec in `Starcraft 2` and $O(r) = 6.5$ msec in `Counter Strike: Source`. This

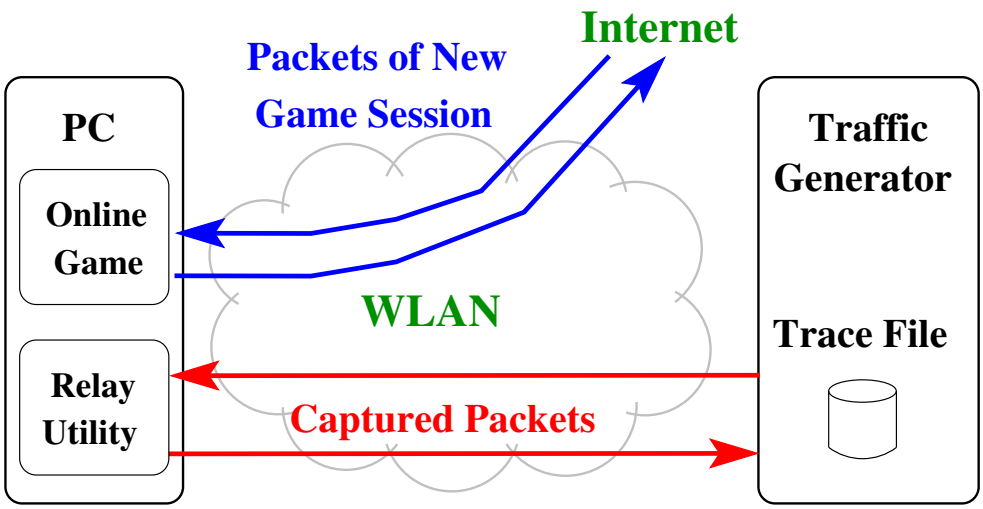


Figure 4.3: The setup for measuring relay latency and traffic overheads.

experiment reveals that, even with the latest online multiplayer games, the latency overhead is insignificant. The experiments also indicate that game-state updates on average incur 40 kbps traffic overhead in `Counter Strike: Source` and 42 kbps in `Starcraft 2`, which are insignificant to broadband access links. The observed traffic overhead measurements are inline with those reported in the literature [47].

4.2.4 Managing Network Dynamics

The IRS system may be affected by network dynamics, such as network congestion, overloaded relay clients, and disconnected relay clients. The outcome of these events is excessive lateness of game-state updates, which results in degraded gaming quality. To cope with network dynamics, the IRS system provides an interface for online games to report excessive lateness of updates, or *lags*. When a lag occurs, the IRS system switches over to the next backup detour path. In addition, the IRS system may conduct a new set of end-to-end RTT measurements in the background to cope with the new network and client conditions. Switching over to backup detour paths leads to several benefits. First, it helps the clients to recover from lags due to network congestion or client failure. Second, it reduces the load on relay clients that cannot keep up with forwarding game-state updates, which prevents the IRS system from overloading relay clients.

4.2.5 Handling Security Concerns

The IRS system carefully handles two types of attacks: denial-of-service (DoS) and man-in-the-middle. DoS refers to the attack where an attacker client floods many packets to his/her opponent in order to inflate the RTT between the victim client and its host. The victim client in turns suffers from sluggish responsiveness and may even be dropped from the game session [48], which gives the attacker client advantages. In the IRS system, an attacker client may direct the packet flood toward the victim client's active relay client for a DoS attack. This is because the game-state updates between the victim client and its host pass through the relay client. The IRS system addresses such DoS attacks as follows. First, the IRS system never discloses relay candidates of a client to others. Therefore, an attacker client cannot find the victim client's relay client. Second, even if the attacker client accidentally locates the victim client's active relay client, and starts a DoS attack by flooding packets to that active relay client, the victim client would quickly notice a network lag and switch to backup detour paths. Therefore, victim clients can recover from such DoS attacks. Last, any clients that suffer from packet floods would report high RTT measurements to the IRS server. The IRS server, therefore, wouldn't choose them as relay candidates for newly joined clients. With these three mechanisms, employing the IRS system does not increase the clients' chance of DoS attacks.

In man-in-the-middle attacks, an attacker client makes two connections to victim clients and relays modified or delayed game-state update messages between them in order to gain advantages. In the IRS system, a client can maliciously report very low RTT measurements to attract others using it as a relay client and conduct man-in-the-middle attacks. To handle such attacks, the IRS client provides an interface for online games to selectively send sensitive data, such as shared keys, over direct paths to avoid potential eavesdropping. This allows online games to send encrypted game-state updates using methods such as Monch et al. [49], and prevents attacker clients from modifying game-state updates. If an attacker client delays game-state updates, the target IRS client would notice a network lag and switch to backup detour paths. Hence, the IRS system efficiently handles both types of man-in-the-middle attacks.

SRTT: Shortest RTT Algorithm

1. // **Server**, input: src s, dst t, RTT $D(s,t)$, and K
2. **let r** be the set of all relay client candidates
3. **compute** $\hat{E}(r)$ for all $r \in \mathbf{r}$
4. **sort r** on $\hat{E}(r)$ in descending order
5. **keep** the first K relay clients of **r** // best ones
6. **send r** to client s

1. // **Client**, input: dst t, **r**
2. **foreach** $r \in \mathbf{r}$
3. **conduct** RTT measurements from s to t via r
4. **endfor**
5. **conduct** RTT measurement directly from s to t
6. **sort** $\mathbf{r} \cup \{\varnothing\}$ based on their RTT measurements
7. **use** the best relay client in **r** for detour path
8. **fall back** to the next detour path when lag happens

Figure 4.4: The proposed algorithm.

4.3 The Shortest RTT (SRTT) Algorithm

Fig. 4.4 gives the high-level pseudocode of the proposed algorithm, which we call Shortest RTT (SRTT) algorithm. The algorithm consists of two parts: server and client. The server first finds all potential relay clients, and sorts them on their likelihood function values in lines 2–4. It eliminates the clients with low likelihood function values from the set in line 5, and sends the remaining potential relay clients to source s. Upon receiving the potential relay clients, in lines 2–4, client s goes through the relay clients and conducts end-to-end RTT measurements through each of them. In line 5, the RTT of the direct path is measured. Client s then sorts the detour and direct paths using the RTT measurements in line 6 and picks the best one of them in line 7. The client switches over to backup detour paths in line 8 if lags are reported.

4.4 Overhead Analysis

The proposed IRS system incurs low processing and network overheads. The processing overhead on each client is dominated by line 6, which takes $O\Big((K+1)\log(K+1)\Big)$ operations as $|\mathbf{r} \cup \{\varnothing\}| = K+1$. Since K is a small system parameter, the processing overhead on clients is negligible. The processing overhead on the server is dominated by line 4, as line 3 computes $\hat{E}(r)$ using the closed-form formula in Eq. (4.1). Therefore, the *worst case* processing time is $O(M \log M)$, where M is the number of clients in the gaming network. The average number of relay candidates is typically close to the number of neighbors N, and the *average* processing time at the server is $O(N \log N)$, where N is a small system parameter, e.g., Dabek et al. [46] state that using $N = 32$ in Vivaldi leads to good performance. Since the average and maximum processing overheads on the server are low, and the SRTT algorithm only runs at session initialization time, a reasonable lobby server can serve a large number of clients.

The network overhead between clients and the server is small as each client updates the server at most once every T sec, and each update consists of the coordinates of the reporting client and on average N RTT measurements to its neighbors. Since N is a small system parameter, each update can be packed into a single packet. Since T is in the order of seconds, the network overhead is negligible. The network overhead among clients is also small. First, a relay client contributes a small bandwidth (about 40 kbps as reported in Chapter 4.2.3) toward every client using it as the relay client. Second, in typical network coordinate systems, a client sends control messages to its neighbors infrequently. For example, as presented in Chapter 5, experimental results using Pyxida [50] show that each client sends a probing message every 16 secs on average. Hence, the network overhead incurred by the IRS system is negligible.

4.5 Implementation

The implementation of the IRS system consists of about 3,700 lines of Java code. The IRS system consists of two parts: client and server. The IRS client runs on game clients. The IRS server may run on the lobby server or on a standalone machine. Running the IRS server on a standalone machine allows multiple lobby servers to share the same IRS server via remote procedure calls and enables load balancing. Detailed discussions of the IRS client

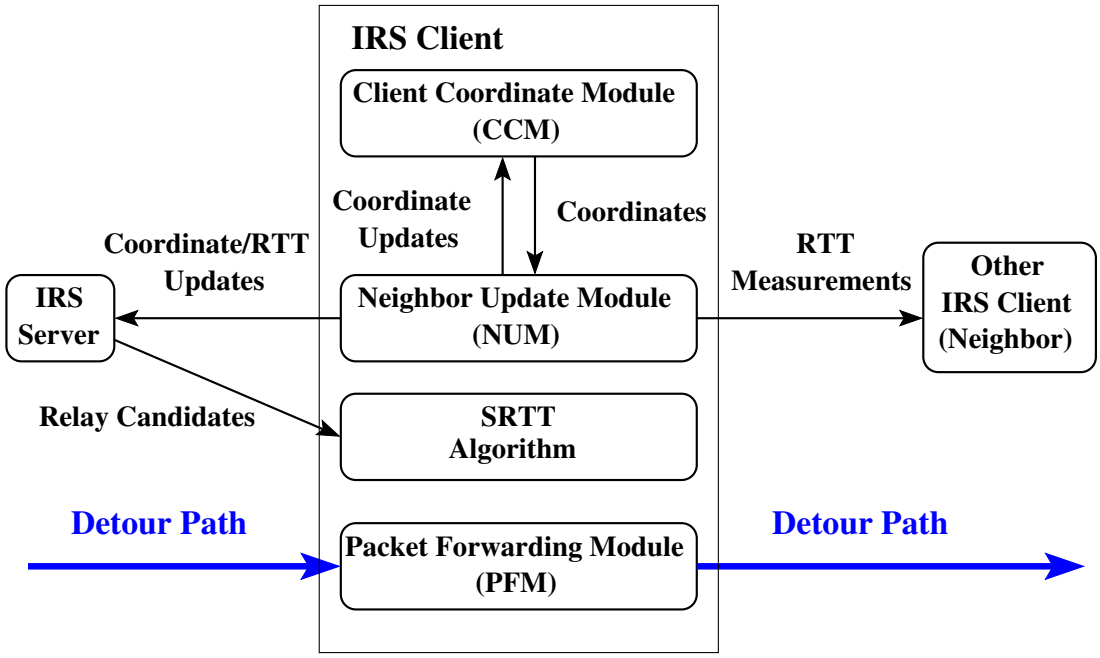


Figure 4.5: IRS Client architecture.

and server are provided in the following subsections.

4.5.1 IRS Client

The IRS client consists of three modules: (i) neighbor update module (NUM), (ii) client coordinate module (CCM), and (iii) packet forwarding module (PFM). Fig. 4.5 illustrates the IRS client architecture. The neighbor update module is responsible for the control messages. It maintains communication channels with the IRS server and the neighboring clients. When a new client joins the IRS system, its neighbor update module connects to the IRS server and requests a list of neighbors. Upon getting the list of neighbors, the neighbor update module connects to the neighbors and schedules periodic RTT measurements to them. The time intervals between RTT measurements are adaptive so that neighbors that have stable network coordinates are assigned longer measurement intervals. This is to reduce the number of RTT measurements and network overhead of the IRS system. The neighbor update module is also responsible for sending the coordinates and RTT measurements to the IRS server.

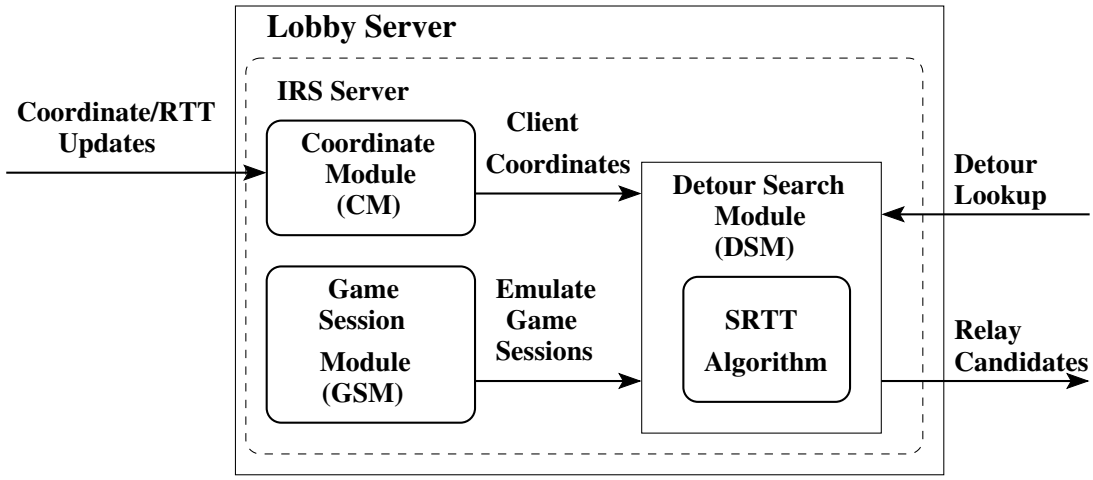


Figure 4.6: IRS Server architecture.

While the neighbor update module schedules the RTT measurements, it does not implement the network coordinate system itself. Instead, the network coordinate system is implemented in the client coordinate module . The neighbor update module gets the latest coordinates from the client coordinate module whenever the neighbor update module decides to send a coordinate update to the IRS server, or receives an RTT measurement request from a neighbor. The client coordinate module implementation is based on the open source Pyxida project [50], which implements Vivaldi [46] algorithm. Specific modifications were made to the Pyxida implementation to make it suitable for the IRS system. For example, the IRS system alleviated the need for neighbor discovery by using the neighbor update module. Thus, neighbor discovery using gossip messages was removed. This eliminates the convergent time needed by Pyxida [35].

While the neighbor update module and client coordinate module are in the control plane of the IRS client, the packet forwarding module is in the data plane and maintains the detour paths. That is, all game-state updates are sent to the packet forwarding module, and re-transmitted to the destination client. Following the results from Feng et al. [47], the packet's sizes are randomly set between 25 and 100 bytes to emulate real life game traffic. The purpose of using *synthetic* game-state updates is to measure end-to-end RTTs, which include the actual relay overhead. That is, round-trip relay overhead $O(r)$ *is* part of RTTs reported in the experimental results. The packet forwarding module switches over to backup detour paths whenever network lags occur.

4.5.2 IRS Server

The IRS server consists of three modules: (i) coordinate module (CM), (ii) detour search module (DSM), and (iii) game session module (GSM). Fig. 4.6 illustrates the IRS server architecture. The coordinate module is essentially a database and manages the coordinates and RTT measurements sent by clients. The coordinate module provides the coordinates and RTTs to the detour search module. The detour search module implements the server-side of the SRTT algorithm and provides detour path lookup service to IRS clients. Upon receiving a detour lookup request from an IRS client, the detour search module invokes the SRTT algorithm to compute a set of potential detour paths, which are sent back to that client.

While the coordinate module and detour search module are sufficient to provide detour path lookup service, a game session module was implemented in the IRS server to emulate players, who may join game sessions. To emulate typical game matches, the game session module periodically creates a new random game session every 15-60 sec, and each game session lasts between 3 to 10 minutes. The game session module is programmed to generate random game sessions as follows. First, game sessions information are collected and analyzed in Sec. 3.1 and used to derive an empirical PMF (probability mass function) for the number of players per session. Then this probability distribution is used to find a random number of players for each game session. Next, we let k be the resulting number of players. The game session module randomly chooses k IRS clients from all active clients, and it selects a random host from these k clients. Once the clients are determined, the game session module emulates this game session by finding detour paths from individual clients to the host. The game session module achieves this by sending multiple lookup requests to the detour search module. The game session module collects statistics on the detour and direct paths, and saves them in a log file for evaluation.

Chapter 5

Deployment and Evaluation of IRS

In the previous chapter, we presented details of the proposed IRS system. In this chapter, we present experimental results from deployment on PlanetLab and home computers with DSL and cable modem access links.

5.1 PlanetLab Deployment

IRS clients were deployed on more than 500 PlanetLab nodes. PlanetLab [51] is a planetary-scale research testbed for deploying and evaluating networking services. It is designed to subject network services to real world conditions. The IRS server was deployed on a workstation at the NSL lab. Table. 5.1 provides a summary of IRS server parameters used in the deployment. To rule out time-of-day variations on network conditions, the GSM module was instructed to perform the same experiment five times, with each lasting more than nine hours. More than 3,000 game sessions with length 3 to 10 mins are randomly

Table 5.1: IRS server parameters.

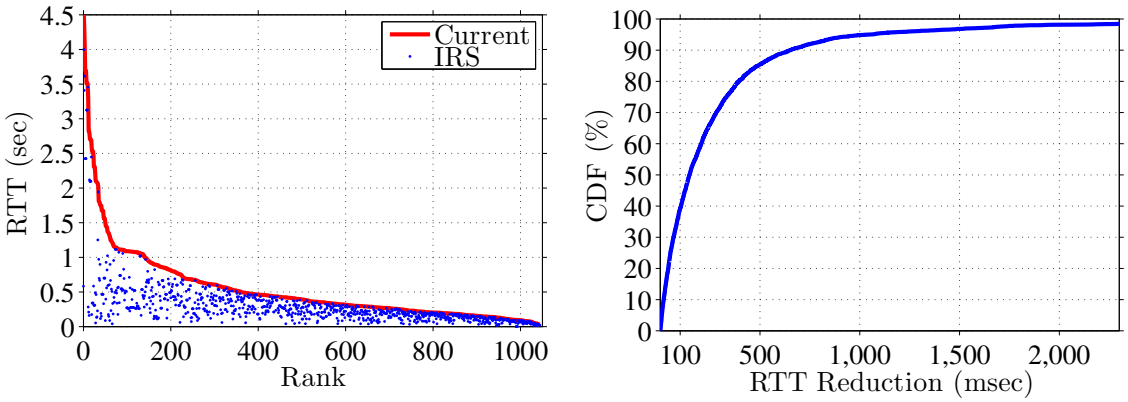
Parameter	**Value**
K	32
Δd	64 msec
Δx	64 msec
T	60 sec
N	32
Session Time	3 to 10 min
Session Spawn Rate	20 to 30 sec

created with the number of players per session following an empirical driven probability distribution. The performance of the IRS is consistent across all experiments. The results of the experiment were selected from an experiment conducted between 1:05pm and 10:17pm on January 7th 2010 (PDT). For each game session, real RTTs (including relay overhead) of the detour paths were collected and computed by the DSM and saved into a log file, which was then post-processed to quantify the performance of the IRS system. For comparison, the IRS server also logged RTTs of the direct paths to be used in computing the performance of the current gaming networks. In the figures, IRS denotes results achieved by the IRS implementation, and Current denotes results without the IRS implementation.

The following performance metrics are used in the PlanetLab experiments. For each game session, we measure the end-to-end RTT between each client and the host. The *session* RTT is defined as the highest RTT from any client to the session host. Given that game-state updates must be validated by the host, the session RTT determines the gaming quality and we report session RTTs if not otherwise specified. The IRS server also keeps track of the number of probing and update packets, which represent the amount of overhead imposed by the IRS system. Last, we consider player performance as a performance metric, as longer RTT results in worse gaming experience, and thus worse player performance. Two first-person avatar games, a shooter game and a racing game, were selected as performance representatives. Empirical functions given in [16] were used to map the RTT of each session to the player performance in hit fraction and average lap time. Due to the dynamic nature of the PlanetLab nodes and the varying loads on them, the IRS clients running on some nodes were disconnected from the Internet during the experiments. Results from failed clients were removed from the data set.

5.1.1 Experimental Results from PlanetLab

RTT Reduction. RTT reductions achieved by the IRS implementation are shown in Fig. 5.1. In Fig. 5.1(a) sessions are sorted on RTTs with and without the IRS system in descending order. The first 1,000 sessions are shown. This figure shows that the IRS system significantly reduces RTTs for many sessions. RTTs of some sessions are reduced from more than 3 sec to less than 0.3 sec, which is more than 10 fold improvements. The IRS system never results in longer RTTs than the current system: the IRS system resorts to the direct path if no better detour path is found. In order to quantify the overall RTT reduction we need to compute the RTT reduction of all game sessions. Fig. 5.1(b) shows the CDF



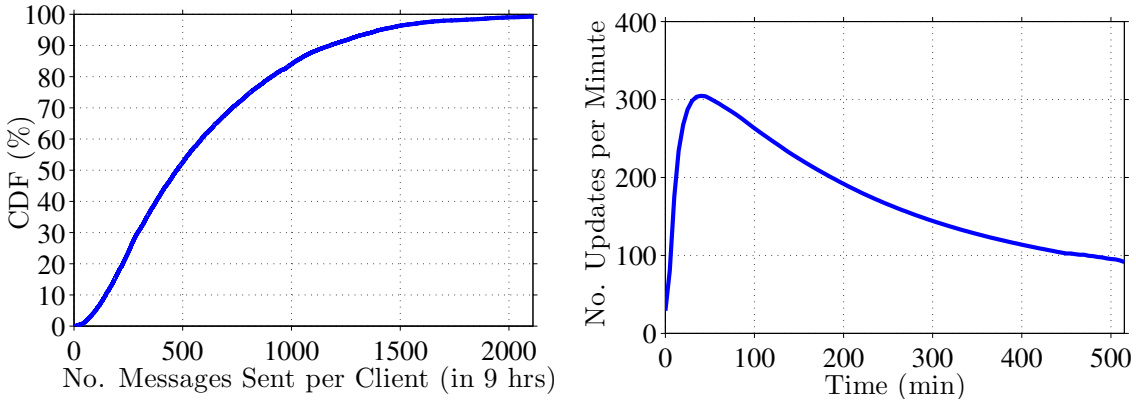
(a) (b)

Figure 5.1: RTTs achieved by the Current and the IRS systems.

(cumulative distribution function) of RTT reduction across all game sessions. The resulting CDF indicates that, with the IRS system, nearly all game sessions observe some RTT reduction, while more than 60% of them achieve 100 msec or higher RTT reduction. This is a significant improvement considering that the minimum RTT required by first-person avatar games is only 100 msec [16].

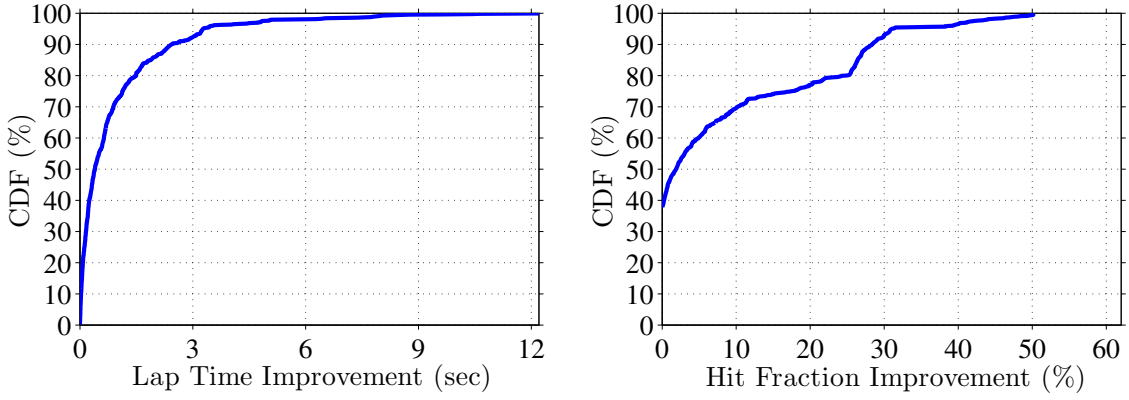
Imposed Overhead. The IRS system incurs some overhead, including probing messages among clients and update messages between clients and the server. The accumulated number of probing messages sent by each IRS client throughout the 9-hr experiment are counted and shown in Fig. 5.2(a). This figure shows that almost all clients imposed less than 2,000 messages in a 9-hr time period, which is about one packet every 16 sec on average. Next, we compute the number of update packets received by the IRS server. The update packets carry either the latest coordinates or RTT measurements. Fig. 5.2(b) shows the average number of update packets per minute received by the IRS server. This figure shows that the number of update messages is fairly small: up to 300 per minute are observed. Given that there are more than 500 PlanetLab nodes in the experiment, each IRS client sends less than one update message per minute to the IRS server. This illustrates that the load on the IRS server is low, and it can serve a large number of clients. In addition, this figure shows a decreasing trend on the IRS server load. This is because once the client coordinates are stabilized, they send fewer update packets to the server.

Player Performance. The expected player performance improvement due to the RTT



(a) (b)

Figure 5.2: Overhead incurred by the IRS system: (a) number of messages sent by each client in a 9-hr experiment, and (b) updates per minute received by the server.



(a) (b)

Figure 5.3: Player performance: (a) lap time in a racing game, and (b) hit fraction in a shooter game.

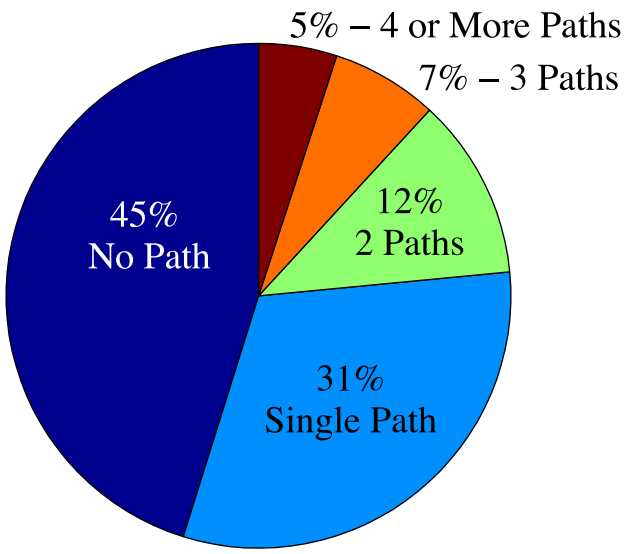


Figure 5.4: Number of detour paths found by the IRS system.

reduction achieved by the IRS system was computed. The two player performance metrics are *lap time* which is the average time a player finishes a lap in a racing game, and *hit fraction*, which is the ratio of the hit shots over the total fired shots using high-precision weapons such as rifles in a shooter game. Fig. 5.3(a) shows the lap time improvement and the hit fraction improvement is shown in Fig. 5.3(b). Fig. 5.3 shows that 40% of players can reduce their lap times by more than 1 sec and 30% of players can increase their hit fractions by more than 10%. The improvements on player performance are because of more responsive systems and smoother rendering, which are due to smaller RTTs achieved by the IRS system. Fig. 5.3 indicates that employing the IRS system leads to higher gaming quality, and thus better player performance. This in turn will stimulate players to play more online games, and thus increase the revenues of the online gaming companies.

Existence of Backup Detour Paths. As mentioned in Chapter 4.2, the IRS system may be affected by network dynamics, the IRS system copes with this by switching over to backup detour paths. Thus, the IRS system's ability to cope with network dynamics is dependent on the number of discovered detour paths between clients and their hosts. Fig. 5.4 illustrates the number of detour paths for individual clients. This figure shows that 55% of the clients have at least one detour path, and 24% of the clients have two or more. This illustrates that the IRS system finds backup detour paths even in small scale networks

Table 5.2: IRS server parameters for residential deployment.

Parameter	**Value**
K	8
Δd	64 msec
Δx	64 msec
T	60 sec
N	8
Session Time	3 to 10 min
Session Spawn Rate	30 to 60 sec

with only 500 clients and N neighbours restricted to 32, whereas a typical popular online gaming network may have millions of players [52]. The results show that the IRS is capable of coping with network dynamics.

5.2 Residential Deployment

Although the PlanetLab experiments illustrate that the IRS system results in performance improvement, it is worth noting that PlanetLab nodes may have characteristics different from those of residential machines. To show that the IRS system also works in residential environments, IRS clients were deployed on 17 home computers with DSL and cable modem access links. Due to the smaller number of participating nodes the IRS server parameters needed minor adjustments. Table. 5.2 summarizes these adjustments. The geographic locations of the 17 players are illustrated in Fig 5.5 [2]. The GSM module was used to randomly initiate new game sessions between players, but users may launch and close their IRS clients at any time.

Despite a small number of participants, the IRS system identified more than 8 detour paths among them. Fig. 5.5 shows a representative detour path found among residential computers. In the data collected, an IRS client in Vancouver, Canada had an average direct RTT of 199.37 msec to another client in Linköping, Sweden. The IRS system found a shorter detour path using a node in Los Altos, CA. The detour path resulted in an average RTT of only 101.27 msec. During the one week long experiment, consistency was observed in relay

[2]Thank you to all the volunteers who participated in the experiments.

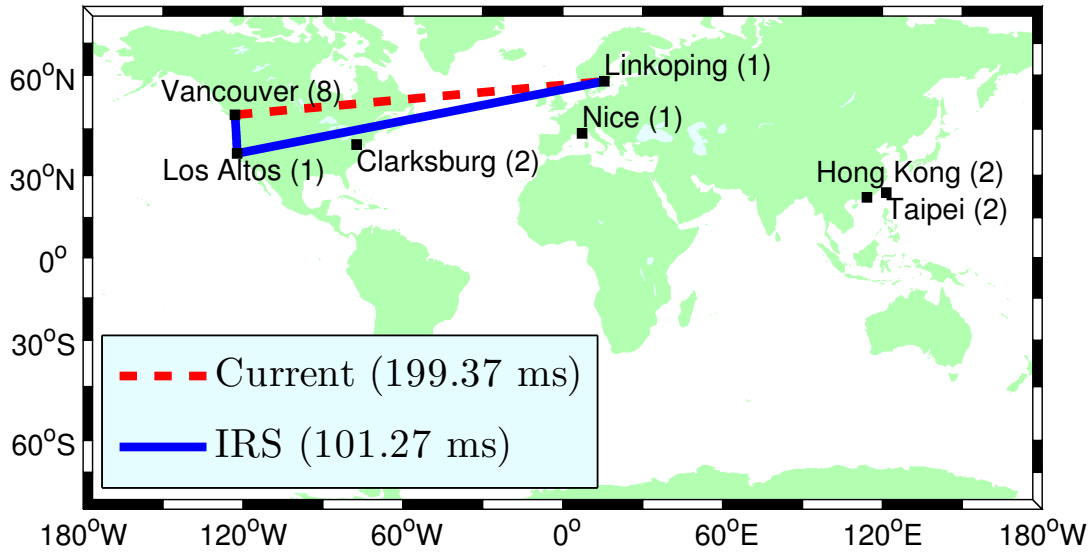


Figure 5.5: Sites in the residential measurement experiments.

node selections. For example, the Vancouver IRS client consistently picked the relay node in Los Altos. The only time it selected another node was when the Los Altos node was offline. The backup detour path, using another node in Vancouver, Canada resulted in an RTT of 162.61 msec. The residential deployment shows that the IRS implementation works even among home computers with residential access links, which may have high last-mile delay.

The number of results from the residential deployments is too small to conclusively quantify the impact of the IRS system on gaming quality using subjective user studies [1–3, 16,42,43]. However, one way around this problem is to build an emulator based on an open source first-person avatar game called BZFlag [53]. BZFlag is a multiplayer tank game, in which several players drive tanks in a battlefield and shoot each other for as many kills as possible. BZFlag is a representative of a typical online multiplayer game in which players play in their own home. BZFlag implements modern latency compensation techniques including dead reckoning [25,26] for movement predictions and smoothing algorithms [54] for correcting inconsistency due to inaccurate predictions. In order to eliminate any bias due to human factors the emulator uses only computer players or artificial intelligent players (AI). The emulator is built on top of the GLS (Game Latency Simulator) system implemented

Table 5.3: Results from BZFlag emulator and actual RTT traces of home computers.

Client 1	**Client 2**	**Hit Fraction (%)**		**Deviation (m)**	
		Current	IRS	Current	IRS
AS6678 (Cable, FR)	AS3462 (DSL, TW)	4.9958	19.6524	7.1354	2.0132
AS6327 (Cable, CA)	AS8473 (Cable, SE)	6.3249	28.5963	5.6272	1.2313
AS33657 (Cable, US)	AS3462 (DSL, TW)	5.8718	14.1991	6.0996	3.9364

in [55]. The GLS system closely emulates several BZFlag's computer players competing in a battlefield, and stores detailed statistics such as tank position, number of shots, and number of hits in log files for offline analysis. The GLS system, however, does not emulate network latency: a fixed RTT is used throughout each simulation for all players. Modifications were made to the GLS system to take RTT trace files as input and *faithfully* emulate real BZFlag clients running on home computers.

The results from the residential deployment are used to drive the emulator. First, the trace file of RTT measurements on the direct path is used to drive a one-hour game between two computer players. Next the trace file of RTT measurements over the active detour path is used to repeat the another game. The gaming quality is compared between these two games. Two performance metrics were considered: hit fraction and position deviation [55]. Hit fraction refers to the ratio of hit shots over the total shots, while the position deviation refers to the distance between the displayed tank position and the actual tank position. Low hit fraction and long position deviation indicate that the latency compensation algorithms implemented in BZFlag cannot accommodate the excessive network latency, and result in degraded gaming quality. Table 5.3 illustrates the average hit fraction and position deviation for three sample gaming sessions. This table clearly shows that residential users with cable modem and DSL access links can benefit from the IRS system with significant performance improvement: average hit fraction is improved by up to 4.5 times and the average position deviation can be reduced from about 5 meters to 1 meter. The emulation results illustrate that the IRS system works: (i) in residential networks and (ii) on modern online games that have implemented latency compensation algorithms.

5.3 Summary

In summary, the PlanetLab and residential experimental results clearly show that the IRS system improves the online gaming performance from several aspects: (i) it significantly reduces RTTs in game sessions, (ii) it imposes negligible network and processing overheads, (iii) it increases the gaming quality and player performance, and (iv) it allows many clients to have backup detour paths to cope with system dynamics.

It is worth mentioning that the results presented above are very *conservative* because the assumption is that each pair of clients forms a single gaming session. In reality, many more clients join a session, and the gaming quality is determined by the client with the *highest* RTTs to the session host. Hence, clients with smaller RTTs may *suffer* from excessive RTTs of other clients, and the potential of the IRS system should be even more significant than what has been reported in this chapter.

Chapter 6

Conclusions and Future Work

In this chapter, first we summarize this thesis. Then, we briefly describe the possible extensions of this work.

6.1 Conclusions

Detour routing is an application-level routing mechanism, which allows online games to locate better routing paths (in terms of latency) that are not possible in IP routing because of routing policies. To study the potential of detour routing, RTT measurements among players of popular online games are needed.

To achieve this, we developed scripts to collect clients' IP addresses from a popular gaming server and measured RTT values between individual client pairs. We quantified the potential gain of detour routing in terms of network performance metrics using the collected traces. We analyzed the pairwise RTT reduction, reachability (additional number of players that can be reached), and session RTT reductions of 0-hop, 1-hop, 2-hop, 3-hop, and k^*-hop shortest distances, where k^* is the optimal number of hops. The results showed that significant RTT reduction can be achieved by detour routing. For example, with k^*-hop detour routing, more than 50% of the client pairs can achieve more than 100 msec pairwise RTT reduction, and more than 90% of the game sessions achieve more than 100 msec session RTT reduction. We also observed that larger numbers of intermediate nodes (k) result in higher RTT reductions in k-hop detour routing. However, the improvement diminishes when $k \geq 4$. Furthermore, the results illustrated that simple 1-hop detour routing suffices for most practical cases, as it achieves up to 80% of the RTT reduction achieved by the

optimal k^*-hop detour routing.

We also analyzed the benefits of detour routing on the application performance metrics. We showed that detour routing can lead to significant improvement in several player performance metrics in various avatar games, such as: (i) hit fraction, kill number, death number in first person shooter games, (ii) lap completion time and frequency of leaving the track in car racing games, (iii) movement speed, combat speed, health point residue, and mana point residue in MMORPG (massively multiplayer online role-playing game) games, and (iv) attempt gain in football games. For example, our results showed that 67% of players in first person shooter games gain 18% hit ratio improvements, and some of them can improve their hit ratio by up to 40%. Better player performance indicates that detour routing improves gaming quality in terms of higher frame rates, quicker responsiveness, and smoother movements.

In addition, we presented the Indirect Relay System (IRS) that allows online game clients to find and utilize detour paths in order to reduce end-to-end RTTs. IRS supports three operations. First, the server employs a network coordinate system and RTT measurements to identify potential detour paths between any two clients. Second, the source client conducts end-to-end RTT measurements to destination via each relay client, and selects the detour path with the smallest RTT as the active detour path. Third, IRS monitors the lateness of game-state updates and switches to the best backup detour path whenever network lags occur. We implemented IRS and deployed it on more than 500 PlanetLab nodes and on several home computers with residential access links. We evaluated IRS using real experiments and trace-driven simulations. Our experimental and simulation results indicate that IRS reduces RTTs among game clients, while imposing negligible network and processing overheads. Smaller RTTs result in better gaming quality and higher player matchability, which are two major quality-of-service metrics in online games.

6.2 Future Work

The work in this thesis can be extended in several directions. Some of them are summarized in the following.

- The proposed SRTT algorithm requires multiple on-demand RTT measurements. In the worst case scenario the algorithm may encounter bad links, which could significantly increase the actual running time of the algorithm. A possible solution to this

problem is to add timestamps to the RTT measurements reported by IRS clients. The SRTT algorithm would then only perform on-demand measurements if it encounters stale RTT measurements.

- The SRTT algorithm does not check if the returned detour paths have available bandwidth to spare. It relies on the IRS client to manage the network dynamics and switches over to backup detour paths if the optimal path is congested. One possible improvement is to implement a load-balancing table to maintain the conditions of known paths. IRS clients periodically report bandwidth availability. The reporting mechanism can be piggybacked with the existing coordinates and RTT measurements updates. The SRTT algorithm then uses the load-balancing table on the IRS server to check for loads on the path.

- Detour paths discovery can be further optimized by improving the neighbor distribution process. As mentioned in Section 4.5, when a new client joins IRS, its neighbor update module (NUM) connects to the IRS server and requests a list of neighbors. Currently, the IRS server randomly chooses N nodes from the IRS server nodes list and sends it to the IRS client. This process can be optimized by choosing the N nodes based on neighbor proximity and other factors such as bandwidth availability. This optimization will improve the SRTT algorithm's ability to identify optimal detour paths. Furthermore, it will also mitigate the node overloading problem.

- The IRS overhead can be reduced by leveraging the existing proxy-ping mechanism, with which IRS clients have the ability to perform proxy-ping. It acts as the relay for the target client and responds to the ping request. The proxy-ping request can be used as a means to discover new neighbors. As mentioned, each IRS client maintains a list of N neighbors, and when the number of neighbors drops below N the IRS client requests additional neighbors from the IRS server. The number of times an IRS client requests additional neighbors can be reduced by inserting new neighbors discovered via proxy-ping. That is, whenever an IRS client receives a proxy-ping request to a target client that is currently not in the neighbor list, it will add the target client to its neighbor list. Since the IRS server issues proxy-ping requests during the ranking of the detour paths process, it can be assumed that the SRTT algorithm has already identified the optimal target clients for the relay client.

Bibliography

[1] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunket, E. Agu, and M. Claypool. The effects of loss and latency on user performance in Unreal Tournament 2003. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'04)*, pages 144–151, Portland, OR, August 2004.

[2] L. Pantel and L. Wolf. On the impact of delay on real-time multiplayer games. In *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'02)*, pages 23–29, Miami, FL, May 2002.

[3] T. Fritsch, H. Ritter, and J. Schiller. The effect of latency and network limitations on MMORPGs: a field study of Everquest 2. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'05)*, pages 1–9, Hawthorne, NY, October 2005.

[4] The value gamer: Play and purchase behavior in a recession, July 2009. `http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/07/valuegamer_final1.pdf`.

[5] S. Shirmohammadi and M. Claypool. Guest editorial for special issue on massively multiplayer online gaming systems and applications. *Multimedia Tools and Applications*, pages 1–5, June 2009.

[6] Blizzard entertainment, World of Warcraft reaches new milestone: 11.5 million subscribers. `http://www.blizzard.com/us/press/080122.html`.

[7] Cisco visual networking index: Forecast and methodology, June 2009. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`.

[8] C. Chambers, W. Feng, S. Sahu, and D. Saha. Measurement-based characterization of a collection of on-line games. In *Proceedings of the ACM SIGCOMM conference on Internet Measurement (IMC '05)*, pages 1–1, Berkeley, CA, USA, October 2005.

[9] G. Armitage, M. Claypool, and P. Branch. *Networking and Online Games*. John Wiley and Sons, 1st edition, 2006.

[10] S. Agarwal and J. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *Proc. of ACM SIGCOMM'09*, Barcelona, Spain, August 2009.

[11] M. Claypool. Network characteristics for server selection in online games. In *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN'08)*, San Jose, CA, January 2008.

[12] Valve corporation, Team Fortress 2. `http://www.teamfortress.com/`.

[13] Rockstar games, Grand Theft Auto IV. `http://www.rockstargames.com/IV/`.

[14] Blizzard entertainment, Starcraft 2: Wings of Liberty. `http://us.starcraft2.com/`.

[15] Electronic arts, Simcity 4. `http://simcitysocieties.ea.com`.

[16] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, November 2006.

[17] G. Amir and R. Axelrod. *Massively Multiplayer Game Development 2: Architecture and Techniques for an MMORTS*. Charles River Media, 1st edition, 2005.

[18] W. Feng and W. Feng. On the geographic distribution of on-line game servers and players. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '03)*, pages 173–179, May 2003.

[19] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee. Triangle inequality and routing policy violations in the Internet. In *Proc. of Conference on Passive and Active Network Measurement (PAM'09)*, pages 45–54, Seoul, Korea, April 2009.

[20] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyunand K. Claffy, and G. Riley. AS relationships: Inference and validation. *ACM SIGCOMM Computer Communication Review (CCR'06)*, 37(1):29–40, January 2007.

[21] L. Gao and F. Wang. The extent of AS path inflation by routing policies. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM'02)*, pages 2180–2184, Taipei, Taiwan, November 2002.

[22] N.E Baughman and B.N. Levine. Cheat-proof playout for centralized and distributed online games. In *Proc. of IEEE INFOCOM'01*, pages 22–26, Anchorage, AL, April 2001.

[23] F.E. Cecin, C.F.R. Geyer, S. Rabello, and J.L.V. Barbosa. A peer-to-peer simulation technique for instanced massively multiplayer games. In *Proc. of IEEE Symposium on Distributed Simulation and Real-Time Applications (DS-RT'06)*, pages 43–50, Washington, DC, October 2006.

[24] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'04)*, pages 144–151, Portland, OR, August 2004.

[25] Y. Bernierr. Latency compensating methods in client/server in-game protocol design and optimization. In *Proc. of Game Developers Conference (GDC'01)*, San Jose, CA, March 2001.

[26] J. Vogel and M. Mauve. Consistency control for distributed interactive media. In *Proc. of ACM Multimedia'01*, pages 221–230, Ottawa, Canada, September 2001.

[27] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'04)*, pages 161–165, Portland, OR, August 2004.

[28] C. Chambers, W. Feng, W. Feng, and D. Saha. A geographic redirection service for online games. In *Proc. of ACM Multimedia'03*, pages 227–230, Berkeley, CA, November 2003.

[29] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye. Measurement and estimation of network QoS among peer Xbox 360 game players. In *Proc. of Conference on Passive and Active Network Measurement (PAM'08)*, pages 41–50, Cleveland, OH, April 2008.

[30] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *ACM SIGOPS Operating Systems Review*, 35(5):131–145, December 2001.

[31] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQos: An overlay based architecture for enhancing Internet QoS. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI'04)*, pages 71–84, San Francisco, CA, March 2004.

[32] S. Yang, Y. Kim, and B. Wang. Designing infrastructure-based overlay networks for delay-sensitive group communications. In *Proc. of IEEE Global Communications Conference (GLOBECOM'07)*, pages 565–570, Washington, DC, November 2007.

[33] K. Vik, C. Griwodz, and P. Halvorsen. Constructing low-latency overlay networks: Tree vs. mesh algorithms. In *Proc. of IEEE Conference on Local Computer Networks (LCN'08)*, pages 36–43, Montreal, Canada, October 2008.

[34] S. Ren, L. Guo, and X. Zhang. ASAP: an AS-aware peer-relay protocol for high quality VoIP. In *Proc. of IEEE Conference on Distributed Computing Systems (ICDCS'06)*, pages 70–80, Lisboa, Portugal, July 2006.

[35] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic relationships in Internet routing overlays. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*, pages 469–480, Boston, MA, April 2009.

[36] C. Lumezanu, D. Levin, and N. Spring. Peerwise discovery and negotiation of faster paths. In *Proc. of Workshop on Hot Topics in Networks (HotNets'07)*, Atlanta, GA, November 2006.

[37] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, January/February 1999.

[38] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. In search of the elusive ground truth: The Internet's AS-level connectivity structure. In *Proc. ACM SIGMETRICS'08*, pages 217–228, Annapolis, MD, June 2008.

[39] Qstat web page, July 2009. `http://www.qstat.org`.

[40] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW'02)*, pages 5–18, Marseille, France, November 2002.

[41] Boost C++ libraries, August 2009. `http://www.boost.org/`.

[42] J. Nichols and M. Claypool. The effects of latency on online Madden NFL Football. In *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'04)*, pages 146–151, Kinsale, Ireland, June 2004.

[43] M. Claypool. The effect of latency on user performance in real-time strategy games. *Journal of Computer Networks*, 49(1):52–70, September 2005.

[44] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The effect of latency on user performance in Warcraft III. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '03)*, pages 173–179, May 2003.

[45] G. Wang, B. Zhang, and T.S.E. Ng. Towards network triangle inequality violation aware distributed systems. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 175–188, New York, NY, USA, 2007. ACM.

[46] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *Proc. of ACM SIGCOMM'04*, pages 15–26, Portland, OR, September 2004.

[47] W. Feng, F. Chang, W. Feng, and J. Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, 13(3):488–500, June 2005.

[48] J. Yan and B. Randell. A systematic classification of cheating in online games. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'05)*, pages 1–9, Hawthorne, NY, October 2005.

[49] C. Monch, G. Grimen, and R. Midstraum. Protecting online games against cheating. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'06)*, pages 1–11, Singapore, October 2006.

[50] J. Ledlie, P. Pietzuch, M. Mitzenmacher, and M. Seltzer. Network coordinates in the wild. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI'07)*, pages 299–312, Cambridge, MA, April 2007.

[51] Planetlab - an open platform for global research network. `http://www.planet-lab.org/`.

[52] Halo 2 hits 5 million players!, September 2007. `http://www.bungie.net/News/content.aspx?type=news&cid=12425`.

[53] BZFlag web page, April 2010. `http://bzflag.org/`.

[54] K.C. Lin, M. Wang, J. Wang, and D.E. Schab. The smoothing of dead reckoning image in distributed interactive simulation. In *Proc. of the AIAA Flight Simulation Technologies Conference*, pages 83–87, Baltimore, MD, August 1995.

[55] W. Palant, C. Griwodz, and P. Halvorsen. Evaluating dead reckoning variations with a multi-player game simulator. In *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'06)*, pages 20–25, Newport, RI, May 2006.