Improving Online Gaming Quality using Detour Paths

Cong Ly
School of Computing Science
Simon Fraser University
Surrey, BC, Canada
cly@cs.sfu.ca

Cheng-Hsin Hsu
Deutsche Telekom R&D Lab
5050 El Camino Real 221
Los Altos, CA 94022
cheng-hsin.hsu@telekom.com

Mohamed Hefeeda
School of Computing Science
Simon Fraser University
Surrey, BC, Canada
mhefeeda@cs.sfu.ca

ABSTRACT

We study the problem of improving the user perceived quality of online games in which multiple players form a game session and exchange game-state updates over an overlay network. We propose an Indirect Relay System (IRS) to forward game-state updates over detour paths in order to reduce the round-trip time (RTT) among players. The IRS system efficiently identifies and ranks potential detour paths between any two players, and dynamically selects the most suitable one based on network and client conditions. To the best of our knowledge, this is the first system that directly reduces RTTs among players in online games, while previous works in the literature mitigate the network latency issue by either hiding it from players or preventing players with high RTTs from being in the same game session. We implement the proposed IRS system and deploy it on 500 PlanetLab nodes. The results from real experiments show that the IRS system improves the online gaming quality from several aspects, while incurring negligible network and processing overheads. We also deploy the IRS system on a number of residential computers with DSL and cable modem access links, and we successfully found several detour paths among them. To evaluate the IRS system with wider ranges of system parameters, we conduct extensive trace-driven simulations using a large number of real game client IPs. The experimental and simulation results show that the proposed IRS system: (i) significantly reduces 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.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

General Terms

Design

MM'10, October 25–29, 2010, Firenze, Italy.

1. INTRODUCTION

Multiplayer online games have become increasingly popular in the past few years, and several market studies predict that online games will continue to grow in terms of market revenues, number of users, and generated Internet traffic volume [28]. Online games require real-time interactions, which make high network latency one of the main challenges to provide high-quality gaming experience over the best-effort Internet. For example, in car racing games, high network latency leads to slow responsiveness, which increases the player's average lap time and results in player frustration. Hence, game developers must carefully handle network latency to provide high-quality gaming experience to players.

In this paper, 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 is referred to as *Indirect Relay System* (IRS). For any two players, the proposed IRS system provides three services: (i) it employs network coordinate systems to efficiently identify potential detour paths, (ii) it sends a few end-to-end probing messages to rank these detour paths, and (iii) it monitors the lateness of game-state updates on the *active* detour path and dynamically switches to other detour paths to cope with network dynamics.

The IRS system *directly* reduces RTTs among players in online games, while most earlier works mitigate the negative impacts of latency by either applying latency compensation techniques [4, Chap. 6], or matching players exclusively with nearby players in terms of RTTs [1, 10, 11]. Through an actual implementation and extensive trace-driven simulations, we show that, compared to direct links used in current online games, the IRS system significantly reduces RTTs among players, while imposing negligible network and processing overheads. For example, in our real experiments, the IRS system achieves more than 100 msec RTT reduction in 80% of game sessions, while each player on average sends only one small control message every 16 sec which is negligible. While a relay player carries game-state updates for other players, the additional traffic amount is insignificant to broadband access links: our experiments, reported in Sec. 3.1, show that each `Counter Strike: Source` and `Starcraft 2` player generates about 40 kbps game-state updates on average. In addition to RTT reduction, our simulation results show that the IRS system increases the players' matchability: 50% of the players can be matched with 20% or more additional players while maintaining good gaming quality. Last, the IRS system is designed to adapt to network dynamics, and thus is robust to network congestion, dropped players, network outage, and malicious players.

The rest of the paper is organized as follows. In Sec. 2, we survey the related work in the literature. We describe the considered online gaming network model and give an overview of our solution

in Sec. 3. Then, we present the proposed system in Sec. 4. We implement the proposed system and evaluate it on more than 500 PlanetLab nodes and on several home computers in Sec. 5. To exercise wider ranges of system parameters using a large number of realistic game client IPs, we also evaluate the proposed system using trace-driven simulations in Sec. 6. Sec. 7 concludes the paper.

2. RELATED WORK

Coping with network latency is critical to the quality of user experience in online games [12]. 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 [5] 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 [9], 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 [6].

In dead reckoning [7, 29], clients extrapolate the behavior and state of gaming objects and thus can continue rendering frames even if game-state updates are late. 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 if these thresholds are exceeded. Under this principle, several improvements have been proposed for the dead reckoning system, e.g., the authors of [2] 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, 1, 17] *prevents* a new player from starting a game with other players that have high expected network latency to that new player. Chambers et al. [10] propose to use IP-to-geolocation databases to filter out far-away players and redirect players to close-by ones. Htrae [1, 17] 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 may not support *specific matches*, which are the games formed before-hand, e.g., among friends.

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 system directly reduces the network latency.

The IRS system is not the first work that utilizes detour paths for shorter network latency. For example, Savage et al. [27] point out that direct path between two IPs may lead to longer network latency than a detour path through a third IP. More recently, detour paths have been used in peer-to-peer (P2P) overlays [26, 19]. The authors of [26] propose a system to use inferred AS (autonomous system) maps and ping probes to find the detour paths. Their system uses breadth-first search to find detour paths. The authors of [19] propose a symbiotic overlay network, which provides peering incentive by associating a peer with other peers only when they can mutually help each other to reduce network latency to some Internet servers. Similar to [19], the IRS system is built on the distributed detour path discovery method proposed in [21].

3. BACKGROUND AND OVERVIEW

3.1 Background

Online games are roughly classified into two types: avatar games and omnipresent games [12]. In avatar games, a player controls a single character that exists at a precise location in the 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, and third-person avatar games in which a player sees the character from a distance. 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 include real-time strategy games and simulation games.

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 equivalent concept as 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 propose to divide the virtual world into smaller segments through a process known as segmentation [3]. Segments are smaller regions of the virtual world. Players in the same segment exchange game-state updates, and thus each segment is equivalent to a game session in our model.

While our model is applicable to various online games, we will only consider avatar games in the rest of this paper for concrete discussion. In avatar games, each player runs a copy of the game software on his/her machine, and we refer to this machine 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. 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 to validate 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. Fig. 1 illustrates two game sessions, where each session has four players: a host and three clients.

Once connected to the host, clients in the same session start ex-

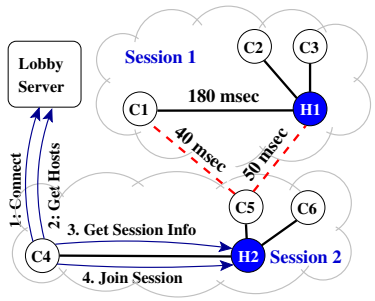


Figure 1: Game sessions in online game systems.

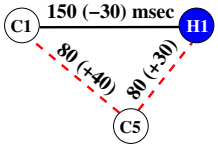


Figure 2: Locating TIV using network coordinate systems.

changing game-state updates via the host using Internet routing. Sending updates directly from a client to its host may lead to longer network latency than sending them over a *detour path* through a *relay* client. This is illustrated in Fig. 1, where the direct path between C1 and H1 (the solid line) has an RTT of 180 msec, while the detour path from C1 to C5 to H1 (the dashed lines) has a total RTT of 90 msec. Therefore, sending game-state updates over the detour path via client C5 reduces the RTT by half. In this example, the triangle of clients 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). Such triangles are called as *triangle inequality violations*, or TIVs. We call side (C1, H1) as the long side and sides (C1, C5) and (C5, H1) as the short sides. TIVs exist in the Internet because inter-domain routing is based on business policies rather than network latency [27], and a few groups have proposed employing overlay routing in distributed systems to reduce network latency [26, 21, 20]. In this paper, we apply similar idea to online games, which are very sensitive to network latency.

Sending game-state updates through a relay game client leads to additional latency and traffic overheads. To quantify actual relay overhead, we chose two popular online games: `Starcraft 2` and `Counter Strike: Source`, and use them to measure overheads. `Starcraft 2` is an omnipresent game and `Counter Strike: Source` is a first-person shooter game; readers interested in more details about them are referred to [22]. We first play a game on a commodity PC with 2.8 GHz Intel CPU for 30 min, and we capture the game-state updates and structure them into a trace file. We then use a traffic generator to replay the captured game-state updates toward our PC, and at the same time we play a new 30-min game session. We also run a relay utility that receives game-state updates from and sends them back to the traffic generator. This utility measures the latency overhead as the difference between the time an update arrives at our PC's network adapter and the time it goes onto the network adapter's outgoing queue. Concurrently running the online game and relay utility on the same PC allows us to measure realistic overheads. We observe an average latency overhead of 6.2 msec in `Starcraft 2` and 6.5 msec in `Counter Strike: Source`. This experiment reveals that, even with modern online games, the latency overhead is insignificant. Our 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. Our traffic overhead measurements are inline with those reported in the literature [14].

3.2 System Overview

We propose the Indirect Relay System (IRS), which 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 overhead 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 our 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 [13]. 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. We control the overhead of the algorithm by heuristically setting thresholds on the changes in the coordinates (Δx) and RTT measurements (Δd) below which the updates are not sent.

4. DESIGN OF THE INDIRECT RELAY SYSTEM: IRS

We first show the steps to identify potential detour paths between any two clients. We then explain the employed ranking procedure and we 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.

Identifying Potential Detour Paths. To identify potential detour paths, we 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. At a first glance, we may think that finding detour paths can be as simple as using network coordinates to find the relay client with the smallest end-to-end RTT among all possible relay clients. This approach, unfortunately, does *not* work, because most

coordinate spaces satisfy the triangle inequality [13], and thus RTT estimations computed using network coordinate systems form no TIVs.

We 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 [19]. 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. We give an example to illustrate the above observation. Fig. 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. 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 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, 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. 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 $\hat{E}(r) =$

$$\begin{cases} E(s,r) = D'(\mathbf{x}_s, \mathbf{x}_r) - D(s,r), \text{ if } D(s,r) \text{ is known;} \\ E(r,t) = D'(\mathbf{x}_r, \mathbf{x}_t) - D(r,t), \text{ if } D(r,t) \text{ is known;} \\ \frac{E(s,r)+E(r,t)}{2}, \text{ if } D(s,r) \text{ and } D(r,t) \text{ are both known,} \end{cases} \quad (1)$$

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.

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 most promising 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. Upon 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.

Managing Network Dynamics. The IRS system may be affected by network dynamics, such as network congestion as well as overloaded, and disconnected relay clients. Relay clients may also be under denial-of-service (DoS) attacks from malicious 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 application programming interface (API) 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 for fast recovery. 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 and departure. 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. Last, it increases the complexity of launching DoS attacks on relay clients, and thus demotivates malicious clients from attacking others.

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 [30], 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 under DoS attacks.

Man-in-the-middle refers to the attack where an attacker makes two connections to victims and modifies/delays game-state updates 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 API 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. [23], and prevents attacker clients from modifying game-state updates. If an attacker client delays game-state updates, the IRS client on the victim client would notice a network lag and switch to backup detour paths. Hence, our IRS system efficiently handles both types of man-in-the-middle attacks.

Pseudo Code. Fig. 3 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

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 3: The proposed algorithm.

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.

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. (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. [13] state that using $N = 32$ in Vivaldi leads to good performance. Since the average and maximal 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 Sec. 3.1) 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 Sec. 5, our experimental results using Pyxida [16] show that each client sends a probing message every 16 secs on average. Hence, the network overhead incurred by the IRS system is negligible.

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

This section first describes a real implementation of the IRS system. We then deploy the system on PlanetLab and present our experimental results from PlanetLab. We also deploy our system on several home computers with DSL and cable modem access links.

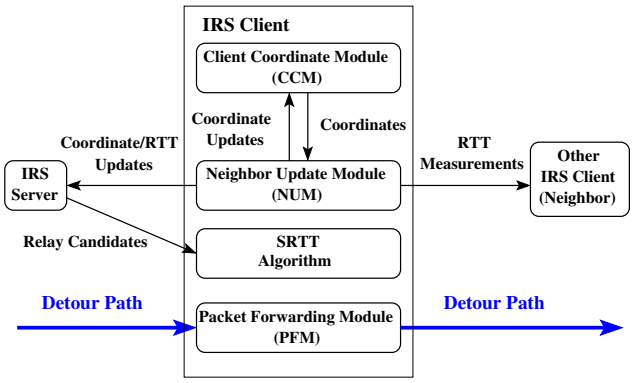


Figure 4: IRS Client architecture.

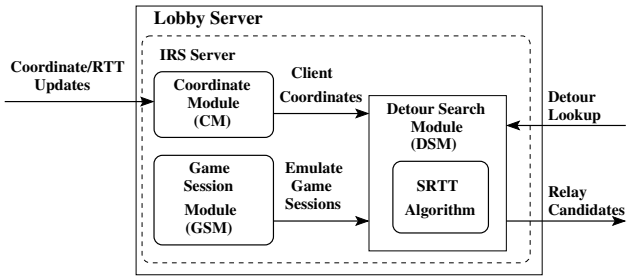


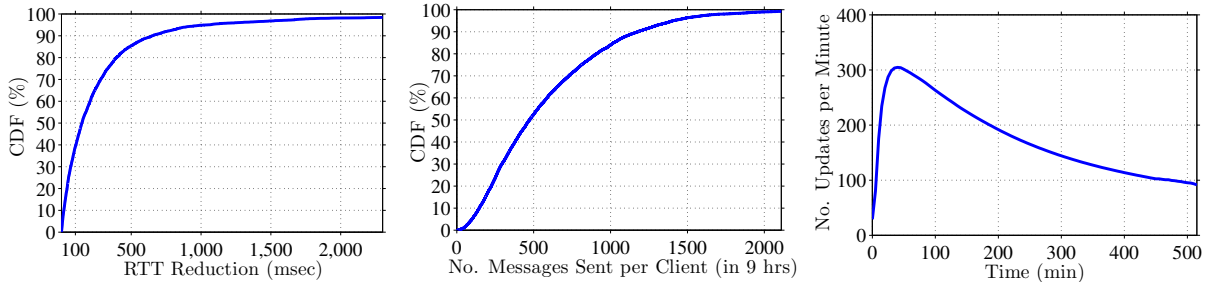
Figure 5: IRS Server architecture.

5.1 Implementation

We implemented the IRS system in 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. We present the IRS client and server below.

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 illustrates the IRS client architecture. The NUM 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 NUM connects to the IRS server and requests a list of neighbors. Upon getting the list of neighbors, the NUM 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 NUM is also responsible for sending the coordinates and RTT measurements to the IRS server.

We notice that while NUM schedules the RTT measurements, it does not implement the network coordinate system itself. Instead, the network coordinate system is implemented in the CCM module. The NUM gets the latest coordinates from the CCM whenever the NUM decides to send a coordinate update to the IRS server, or receives an RTT measurement request from a neighbor. We im-



(a) (b)

Figure 6: RTT reduction achieved by the IRS system.

Figure 7: 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.

plemented the CCM based on the open source Pyxida project [16], which implements Vivaldi [13] algorithm. We modified the Pyxida implementation to make it suitable for our IRS system.

While the NUM and CCM are in the control plane of the IRS client, the PFM is in the data plane and maintains the detour paths. That is, all game-state updates are sent to the PFM, and retransmitted to the destination client. Following the results from Feng et al. [14], we randomly set the packet size between 25 and 100 bytes to emulate real life game traffic. We use these *synthetic* game-state updates 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 our experimental results. The PFM switches over to backup detour paths whenever network lags occur.

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. 5 illustrates the IRS server architecture. The CM is essentially a database and manages the coordinates and RTT measurements sent by clients. The CM provides the coordinates and RTTs to the DSM. DSM 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 DSM invokes the SRTT algorithm to compute a set of potential detour paths, which are sent back to that client.

While the CM and DSM are sufficient to provide detour path lookup service, we implemented the GSM in our IRS server to emulate players, who may join game sessions. To emulate typical game matches, the GSM module periodically creates a new random game session every 15-60 sec, and each game session lasts between 3 to 10 minutes. To be conservative, we chose rather short game sessions because they lead to more client dynamics, which in turns impose more challenges to the proposed IRS system. We programmed the GSM to generate random game sessions as follows. We first analyzed the game session information collected in Sec. 6.1 and derived an empirical PMF (probability mass function) for the number of players per session. We then followed this probability distribution to find a random number of players for a game session. Let k be the resulting number of players. The GSM 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 GSM emulates this game session by finding detour paths from individual clients to the host. The GSM achieves this by sending multiple lookup requests to the DSM. The GSM collects statistics on the detour and direct paths, and saves them in a log file.

5.2 PlanetLab Deployment

We deployed the IRS client on more than 500 PlanetLab nodes. We ran the IRS server on a workstation in our Lab. We let $K = 32$, $\Delta d = \Delta x = 64$ msec, $T = 60$ sec, and $N = 32$. To rule out time-of-day variations on network conditions, we instructed the GSM module 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 were randomly 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, we collected real RTTs (including relay overhead) of the detour paths computed by the DSM and saved them in a log file, which was then post-processed to quantify the performance of the IRS system. For comparison, we also logged RTTs of the direct paths and compute the performance of the current gaming networks. In the figures, we use IRS to denote results achieved by our implementation, and Current to denote results without our implementation.

We consider the following performance metrics in our PlanetLab experiments. For each game session, we measure the end-to-end RTT between each client and the host. We then define *session* RTT as the highest RTT from any client to the session host. Given that game-state updates must be validated by the host, session RTT determines the gaming quality and we report session RTTs if not otherwise specified. We also keep 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. We consider two first-person avatar games: a shooter game and a racing game. We follow an empirical functions given in [12] to map the RTT of each session to the player performance in hit fraction and average lap time. We notice that due to the 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. We filtered out the results collected from these failed clients, and we successfully collected statistics for about 3,000 game sessions.

5.3 Experimental Results from PlanetLab

RTT Reduction. We report the RTT reduction achieved by our IRS implementation. We first sort the sessions on RTTs without the IRS system in descending order, and plot the RTTs with and without the IRS system (figure not shown due to the space limitation). We found 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 improve-

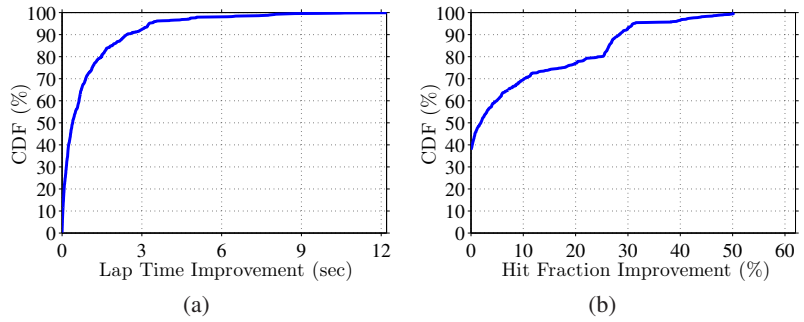


Figure 8: Player performance: (a) lap time in a racing game, and (b) hit fraction in a shooter game.

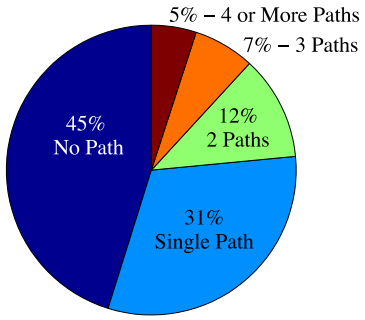


Figure 9: Number of detour paths found by the IRS system.

ments. The IRS system resorts to the direct path if no better detour path is found. Next, we compute the RTT reduction of all game sessions, and plot the CDF (cumulative distribution function) in Fig. 6. This figure shows that nearly all game sessions observe some RTT reduction due to the IRS system, while more than 60% of them achieve 100 msec or higher RTT reduction, which is significant.

Imposed Overhead. The IRS system incurs some overhead, including probing messages among clients and update messages between clients and the server. We count the accumulated number of probing messages sent by each IRS client throughout the 9-hr experiment, and we plot the CDF in Fig. 7(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. We also compute the number of update packets received by the IRS server. The update packets carry either the latest coordinates or RTT measurements. We compute the average number of update packets per minute at the server, and we plot it in Fig. 7(b). 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 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 number of update packets to the server.

Player Performance. We compute the expected player performance improvement due to the RTT reduction achieved by the IRS system. We present CDFs of two player performance metrics: *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. We plot the lap time improvement in Fig. 8(a) and the hit fraction improvement in Fig. 8(b). Fig. 8 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 15%. 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. 8 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. Last, we study the number of detour paths between clients and their hosts. For each client, we compute the number of detour paths from it to its host using the end-to-end RTT measurements. We compute the number of detour paths for individual clients, and we plot its PMF in Fig. 9. 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 with only 500 clients and N neighbours restricted to 32.

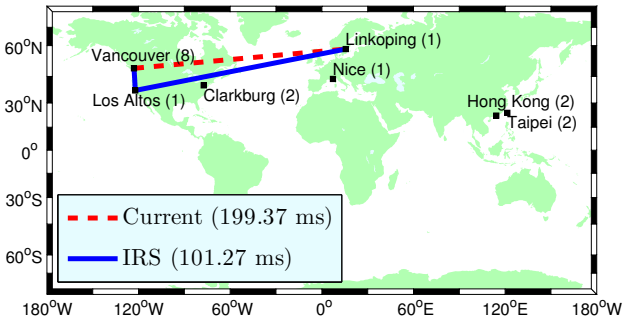


Figure 10: Sites in the residential measurement experiments.

Summary. Our 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.

5.4 Residential Deployment

Although our PlanetLab experiments show that our IRS system results in performance improvement, we acknowledge that PlanetLab nodes may have characteristics different from those of residential machines. To show that the IRS system also works in residential environments, we deployed IRS clients on 17 home computers with DSL and cable modem access links. We let $K = 8$, $\Delta d = \Delta x = 64$ msec, $T = 60$ sec, and $N = 8$. The geographic locations of the 17 players are illustrated in Fig 10. We use the GSM module to randomly initiate new game sessions between two players, but users may launch and close their IRS clients at any time.

Despite a small number of participants, we have identified more than 8 detour paths among them. Fig. 10 shows a representative

Table 1: Results from BZFlag emulator and actual RTT traces of home computers.

Client 1	Client 2	Hit Fraction (%)		Position 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

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, we observed consistency in relay 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 is offline. The backup detour path, using another node in Vancouver, Canada resulted in an RTT of 162.61 msec. The residential deployment shows that our IRS implementation works even among home computers with residential access links, which may have high last-mile delay.

Next, we quantify the impact of our IRS system on gaming quality using an online game and actual RTT measurements collected from residential computers. We achieve this by emulating an open source first-person avatar game called BZFlag [8]. 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 online game because it implements modern latency compensation techniques including dead reckoning [7, 29] for movement predictions and smoothing algorithms [18] for correcting inconsistency due to inaccurate predictions. We have decided to only use computer players in our emulations in order to eliminate any bias due to human factors. More specifically, we constructed our emulator on top of the GLS (Game Latency Simulator) system implemented in [24]. 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. We modified the GLS system to take RTT trace files as input and *faithfully* emulate real BZFlag clients running on home computers.

We consider several pairs of residential clients where the IRS system results in RTT reduction. We first take the trace file of RTT measurements on the direct path and use it to drive a one-hour game between two computer players. We next take the trace file of RTT measurements over the active detour path and repeat the game. Then, we compare the gaming quality in these two games. We consider two performance metrics: hit fraction and position deviation [24]. 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. We report the average hit fraction and position deviation for three sample gaming sessions in Table 1. 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. Our emulation results illustrate that the IRS system works: (i) in residential networks and (ii) on modern online games that have implemented latency compensation algorithms.

6. TRACE-DRIVEN SIMULATION

In this section, we conduct extensive trace-driven simulations to quantify the potential of the IRS system using a number of game client IPs.

6.1 Trace Collection

We need pairwise RTT trace of online games to conduct trace-driven simulations. However, we are not aware of any publicly available RTT traces among a large number of game clients. Existing RTT traces are either sparsely constructed without pairwise measurements, such as the trace used in [11], or not publicly available due to business reasons, such as the Xbox trace used in [1]. Therefore, we *had* to collect our own RTT trace with pairwise RTT measurements among game clients. We compile our trace files in three steps, which are described in the following.

Collecting IPs of Game Clients. We use the `Qstat` utility [25] to get game client IPs. `Qstat` is an open source command-line utility that allows users to browse the information of individual game sessions so that they can join interesting sessions. We have run `Qstat` on many machines with various arguments, and we made two observations. First, the lobby server `h2master.steampowered.com` of `Counter-Strike:Source` returned the largest number of IPs during our experiments. Therefore, we use this lobby server throughout our experiments. Second, we found that the lobby server returns different sets of IPs to `Qstat` running on different machines. We suspect that the lobby server implements a matchmaking algorithm that only returns close-by IPs. To collect IPs of game clients around the globe, we run `Qstat` on more than 550 PlanetLab nodes, and for 60 times on each. After combining all collected IPs together, we had 28,924 distinct game client IPs.

Measuring RTTs among Clients. Since we have no control over the game clients, we measure the pairwise RTT using the `king` utility. `King` supports measuring RTT between two arbitrary IPs, and has been shown to be reasonably accurate [15]. `King` uses DNS servers that support recursive queries for RTT estimation, and it returns errors if abnormal measurement results are observed.

Measuring all pairwise RTTs among the considered 28,924 client IPs would take prohibitively long time. To accelerate the measurements without losing accuracy, we cluster client IPs into /24 subnets, and we measure the RTT between each pair of subnets. We then use the RTT between two subnets as the RTT between any two client IPs in these two subnets. After the clustering, we have 8,063 subnets. For each subnet, we randomly pick a client IP in that subnet as a representative. We then conduct pairwise RTT measurements only among these 8,063 representative client IPs. 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. To overcome this, we developed scripts to run `king` on the 550+ PlanetLab nodes mentioned above. Over

our two-week experiment, we collected 18,884,321 RTT measurements, equivalent to about 230 GB of raw data.

Filtering Unreliable Measurements. 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 because not all `king` measurements were successful, since `king` relies on recursive DNS queries to estimate RTTs, and not all name servers support recursive queries. We notice that, like other measurement utilities, `king` may lead to biased measurement results. We filtered out *unreliable* measurements as follows. First, we dropped all measurements with RTTs < 1 msec, as they probably are due to content-tracking firewalls/proxies [19]. We needed to drop them to avoid over-estimating the potential of the IRS system. Second, we sorted all RTT measurements, and for each subnet pair with more than one RTT samples, we used its median value. Using median RTT to filter out transient congestion and packet loss was suggested by the authors of [13]. After filtering out unreliable measurements, we got 12,930,645 distinct pairwise RTTs. We prepared two trace files: (i) *ping trace* of timestamped ping results, and (ii) *RTT matrix* of median RTTs for individual IP pairs.

6.2 Simulation Setup

We have implemented a trace-driven simulator in Java, which uses Vivaldi [13] as the network coordinate system. We have designed the simulator to be flexible in the sense that many system parameters can be exercised. The simulator runs in two modes: *ping* and *matrix*. In ping mode, the simulator takes the ping trace as input, and replays the RTT measurements following their actual timestamps. Each ping mode simulation is two-week long, and the IRS simulator chooses a random pairs of IPs every minute to evaluate the performance of the IRS system. While the ping mode faithfully recreates pings in real networks, it does not allow changing some system parameters such as ping frequency, neighbor set size, and candidate set size. Therefore, we make our simulator also support the matrix mode, in which the simulator takes the RTT matrix as input and randomly generates pings following the specified system parameters. The median RTTs in the RTT matrix are used as the measurement results. More precisely, in the matrix mode, the IRS simulator assigns each client N neighbors. It then generates RTT measurements between each client and its random neighbors, and stops once P pings have been generated for this client, where P is a system parameter. Once all measurements and updates are completed, the simulator randomly chooses 10,000 pairs of clients for performance evaluation.

In both ping and matrix modes, for each pair of randomly chosen client IPs, we run the SRTT algorithm to find the detour paths. In addition to the performance metrics introduced in Sec. 5, we also consider matchability in simulations. The matchability is the number of clients each client can connect to and maintain good gaming quality. Different types of games have different thresholds on required RTTs for good gaming quality [12], and these thresholds in turn define the matchability of every client IP. We consider the matchability of three RTT thresholds: 50, 100, and 200 msec. In our simulations, we compute the matchability of both source and destination of the randomly chosen IP pairs. For each IP, we identify all other clients that have an RTT shorter than the threshold to the IP. We call these clients *matchable* clients, and we count the number of them. We define the matchability of each client as the number of its matchable clients over the total number of clients appear in the trace file. We compute matchability with and without the IRS system, and report the difference.

We let the overhead $O_r = 10$ msec. We run the simulations several times with wider ranges of system parameters, and we collect

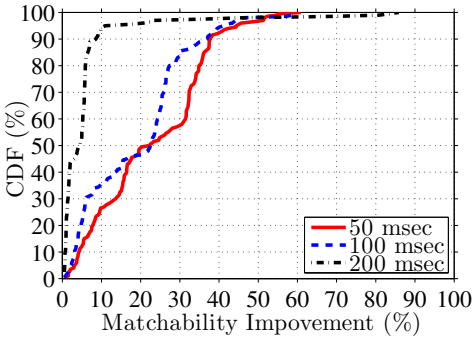


Figure 11: Improvement on matchability.

aforementioned performance metrics. In ping mode, we vary Δd and Δx from 8 to 512 msec, and T from 0.5 to 64 min. In matrix mode, we vary the size of candidate sets K from 2 to 256, the average pings per client P from 10 to 640, and the size of neighbor sets N from 16 to 512. To be conservative, we assume each session has two clients.

6.3 Simulation Results

Our simulation results confirm our findings in PlanetLab experiments reported in Sec. 5.3: the IRS system results in high RTT reduction, low overhead, and improved player performance. Details are not given due to the space limitations. We next report sample simulation results that cannot be easily derived from experiments.

Matchability. We present matchability improvement achieved by the IRS system. We run the IRS simulator in ping mode with $\Delta = 8$ msec and $T = 30$ sec, and we only consider the first 100 pair of client IPs in this simulation. We cannot include all 20,000 IP pairs because computing matchability requires finding detour paths to all client IPs in the trace, which takes a long time. For each IP, we compute the matchability for RTT thresholds of 50, 100, and 200 msec, with and without the IRS system. We compute the CDF for each threshold and plot the results in Fig. 11. This figure shows that employing the IRS system allows players to be matched with many more players, while maintaining good gaming quality. For example, about 50% of the players can be matched with 20% or more additional players when the threshold is 50 or 100 msec.

Impact of Δd, Δx and T. We run the IRS simulator in ping mode to evaluate the impact of these system parameters. Figures of the simulation results are not shown due to the space limitations. We first fix the $\Delta d = \Delta x = 8$ and vary T from 0.5 to 64 min. We then compute the mean RTT reduction of game clients. We observe a quite constant mean RTT reduction under different T values, which indicates the IRS system is not sensitive to T values. Moreover, for any T value, we observe each game client sends 4.5 messages every minute on average, which is equivalent to one packet every 13.33 sec and is insignificant. Next, we fix $T = 30$, and change Δd and Δx from 8 to 512. The simulation results indicate that smaller Δd and Δx values result in higher RTT reduction, while smaller Δd and Δx values also incur more update messages. In general, setting $\Delta d = \Delta x = 64$ msec achieves a good tradeoff between the optimality and overhead.

7. CONCLUSIONS

We proposed the Indirect Relay System (IRS) that allows online game clients to find and utilize detour paths in order to reduce end-to-end RTTs. The IRS system supports three operations. First, the server employs a network coordinate system and RTT measure-

ments 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, the IRS system monitors the lateness of game-state updates and switches to the best backup detour path whenever network lags occur. We evaluated the IRS system using real experiments and trace-driven simulations. We implemented the IRS system and deployed it on more than 500 PlanetLab nodes and on several home computers with residential access links. To exercise wider ranges of system parameters, we also implemented a trace-driven simulator, and conducted extensive simulations. Our experimental and simulation results indicate that the IRS system reduces RTTs among game clients, while imposes 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. More precisely, we observed that more than 80% of game sessions achieve 100 msec or more RTT reduction compared to the current gaming networks, while the number of update and probing packets is negligible. Furthermore, the IRS system increases the matchability: 50% of the clients can be matched with 20% or more clients while achieving good gaming quality.

This work can be extended in several directions. First, more complete RTT traces among game consoles, such as those from Xbox 360 [1], can be employed for larger-scale simulations to better quantify the potential of the proposed system in the wild. Second, despite critical control messages are almost always encrypted, players may come up with creative ways to cheat. We plan to study possible cheating (and anti-cheating) techniques in the IRS system.

8. REFERENCES

[1] 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.

[2] 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.

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

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

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

[6] 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.

[7] 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.

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

[9] F. Cecin, C. Geyer, S. Rabello, and J. 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.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] K. Lin, M. Wang, J. Wang, and D. 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.

[19] 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.

[20] 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.

[21] C. Lumezanu, D. Levin, and N. Spring. PeerWise discovery and negotiation of faster paths. In *Proc. of ACM Workshop on Hot Topics in Networks (HotNets'07)*, pages 1–6, Atlanta, GA, November 2007.

[22] C. Ly. Latency reduction in online multiplayer games using detour routing. Master's thesis, School of Computing Science, Simon Fraser University, Canada, May 2010.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.