# IRS: A Detour Routing System to Improve Quality of Online Games

Cong Ly, Cheng-Hsin Hsu, *Member, IEEE*, and Mohamed Hefeeda, *Senior Member, IEEE*

***Abstract*—Long network latency negatively impacts the performance of online games, and thus mechanisms are needed to mitigate its effects in order to provide a high-quality gaming experience. In this paper, 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. We first collect extensive traces for RTTs among actual players in online games. We then analyze these traces to quantify the potential performance gain of the detour routing. Our analysis reveals that substantial reduction in the RTTs is possible. For example, our results indicate that more than 40% of players can observe at least 100 ms of RTT reduction by routing game-state updates through 1-hop detour paths. Because of the reduction in RTTs, players can join more gaming sessions that were not available to them due to long RTTs of the direct paths. Most importantly, we design and implement a complete IRS system for online games. 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 long RTT 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. In particular, we observe that, with the proposed IRS system, more than 80% of game sessions achieve 100 ms or higher RTT reduction.**

***Index Terms*—Gaming experience, latency reduction, performance optimization, quality-of-service (QoS).**

## I. Introduction

ONLINE games involve real-time interactions among players. Thus, high network latency becomes one of the main challenges for providing a high-quality gaming experience. For example, in first-person shooter games, higher network latency leads to irregular moves and slow responsiveness that affect players' shooting accuracy. Hence, game developers must carefully handle network latency to provide high-quality gaming experience to players. To cope with this, we aim at reducing the network latency in online games in which players form sessions.

Manuscript received September 08, 2010; revised December 03, 2010; accepted January 30, 2011. Date of publication February 14, 2011; date of current version July 20, 2011. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the British Columbia Innovation Council. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Gene Cheung.

C. Ly and M. Hefeeda are with the School of Computing Science, Simon Fraser University, Surrey, BC V3T 0A3, Canada.

C. Hsu is with Deutsche Telekom R&D Lab USA, Los Altos, CA 94022 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2011.2114645

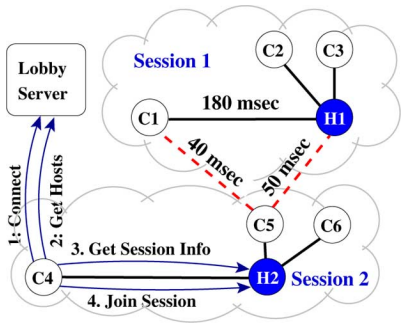


Fig. 1. Game sessions in online games.

We define a *session* as a set of players that frequently exchange game-state updates among themselves. The concept of session is fairly general and can be concretely defined in various types of online games. For example, in first-person shooter games, players start a session once they mutually agree on the same game settings such as map and rules. In large-scale simulation games, players that are close by in the virtual world form a session, because they are likely to interact with each other and share the same area of interest. We strive to minimize the pairwise network latency among players in each session, in order to maximize the overall gaming quality. In extreme cases, such as games similar to `World of Warcraft` and `Everquest`, where game clients only communicate with centralized game servers, each client form a game session with the server (i.e., the server is in all game sessions).

At a first glance, directly sending update messages from a player to another player (or server) 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 [1], and sending update messages through *relay* players may lead to shorter network latency. For illustration, Fig. 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 [2], report that TIVs are not due to measurement errors, and detour paths can often be found in the Internet [3].

In this paper, we rigorously analyze the potential of using detour paths in online games to reduce RTTs among players

and the impact of this RTT reduction on the actual player performance in different types of online games. Furthermore, we present a system called the *Indirect Relay System* (IRS) to efficiently leverage detour paths in online games. The IRS system *directly* reduces RTTs in online games, while earlier works mitigate the negative impacts of latency by either applying latency compensation techniques [4, Ch. 6] or matching players exclusively with nearby players in terms of RTTs [5]–[7].

In particular, the contributions of this paper can be summarized as follows.

- We conduct RTT measurements among players of a popular online game. Our trace files contain more than 18.8 million pairwise RTT measurements collected from online game players distributed over almost 8000 subnets.
- Using our traces, we quantify the potential of the detour routing in online games. We show that more than 40% of players can observe 100 ms 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 improvements in various player performance metrics, such as lap completion time in car racing games and hit ratio in first-person shooter games. Our results indicate that game developers can employ detour routing to improve the gaming quality, attract more players, and increase their revenues.
- We design and implement the complete IRS system, which provides three services: 1) it employs network coordinate systems to efficiently identify potential detour paths; 2) it sends a few end-to-end probing messages to rank these detour paths; and 3) 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.
- We extensively evaluate the proposed system using experiments in PlanetLab and on residential computers with DSL and cable modem access links. The experimental results show that the proposed system: 1) significantly reduces RTTs among players; 2) increases number of peers a player can connect to and maintain good gaming quality; 3) imposes negligible network and processing overheads; and 4) improves gaming quality and player performance.

We emphasize that although network latency is arguably the most important network quality-of-service (QoS) metric determining the gaming quality of online games, other QoS metrics including packet loss rate and delay jitter also affect gaming quality. For concrete discussion, we concentrate on network latency in this paper, but the proposed system can employ other QoS criteria to identify detour paths, rank detour paths, and switch active detour paths. Since multiple QoS criteria can be jointly considered in too many different ways, we believe that how to incorporate multiple QoS criteria in the IRS system is rather a game-specific *design decision* and thus is beyond of the scope of this paper.

The remainder of this paper is organized as follows. In Section II, we survey related work in the literature. We give an overview of online games and the proposed detour routing in Section III. We quantify the potential of detour routing in Section IV. Then, we present the complete system design in Section V. We implement the proposed system and evaluate it on more than 500 PlanetLab nodes and on several home computers in Section VI. Section VII concludes the paper.

## II. Related Work

### A. Latency Compensation

Coping with network latency is critical to the quality of online games [8]. Current latency compensation techniques can be roughly categorized into two groups: time manipulation and matchmaking. Time manipulation techniques compensate for latency by *adjusting* the timestamp of game-state updates. These techniques can further be classified into two approaches: time delay, such as lockstep and event-locking, and time warp, such as dead reckoning.

Lockstep and event-locking are two popular techniques used in practice. The lockstep algorithm [9] controls the consistency among players with unpredictable and varying latency. With this algorithm, players send out game-state updates every fixed interval, and a player is blocked until receiving updates from all other players. In event-locking [10], a player sends game-state updates to a gaming server, and the server relays them to all other players in the same session. While lockstep and event-locking are suitable for local area networks, they perform poorly in the Internet [11]. In dead reckoning [12], [13], players extrapolate the behavior and states of gaming objects and thus can continue rendering frames even if game-state updates are late. To maintain consistency, players agree upon thresholds of prediction errors on individual game-states, and game-state corrections are sent by the server if these thresholds are exceeded. Under this principle, several improvements have been proposed for dead reckoning, e.g., the authors of [14] propose to augment it with synchronized clocks to improve the consistency of gaming objects. While time manipulation techniques attempt to *hide* network latency from players, they may cause negative side effects. More precisely, time delay leads to poor responsiveness and time warp results in inconsistency and irregular moves due to game-state corrections. Therefore, time manipulation techniques may not work in networks with long and varying latency.

Matchmaking-based latency compensation techniques *prevent* a new player from starting a session with other players that have high network latency to that new player. Such techniques can be implemented at a centralized server, such as in Htrae [5], [15], or at individual game clients, such as in `Counter Strike: Source` [16]. For example, Htrae uses 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. Matchmaking-based latency compensation effectively *reduces* the number of players each player can play with and may not support *specific matches*. Specific matches refer to those sessions formed beforehand

among a predefined group of players. In contrast, the proposed IRS system directly *reduces* network latency, which allows each player to match with more players.

### B. Detour Routing for Latency Reduction

Several overlay networks have been proposed to improve the Internet routing performance from various aspects. For example, resilient overlay network (RON) [17] 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 [18] uses overlay routing to provide QoS enhancements in the application layer. Detour paths have also been used to reduce network latency. For example, Savage *et al.* [1] 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 [19], [20]. The authors of [19] propose a system to use inferred autonomous system (AS) maps and ping probes to find detour paths. Their system uses breadth-first search to find detour paths. The authors of [20] 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 toward some Internet servers. Similar to [20], the IRS system is built on the distributed detour path discovery method proposed in [21].

Game developers can borrow ideas from the aforementioned systems to realize detour routing. However, to the best of our knowledge, detour routing is not used in most existing games. In this work, we demonstrate how to leverage detour routing in online games, which has not been explored before in the literature. Different from our preliminary work [22], the present paper rigorously analyzes the potential of detour routing. We achieve this by conducting a large scale RTT measurement study using IP addresses collected from actual online games. We then perform extensive trace-driven simulations to quantify the performance improvement achieved by detour paths in both the network layer (such as RTT reduction) and the application layer (such as lap time in a racing game). We also present new empirical results from actual deployment on residential computers.

## III. Background and Overview

### A. Online Games

Online games are roughly classified into two types: avatar games and omnipresent games [8]. In avatar games, a player controls a single character, which exists at a precise location in the virtual space, and can only interact with nearby 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.

Players of different types of games can tolerate different amount of quality degradation caused by network latency, which can be quantified by the gaming performance of players [8], [23], [24]. This is because games have various degrees of *tightness* on the delivery deadline of game-state updates. For example, high network latency leads to irregular moves, which has a negative impact on players' lap completion time in racing games. In contrast, players of real-time strategy games continuously monitor gaming objects on a large map, and may be less sensitive to irregular moves of a single object. In fact, several experimental studies show that while players of omnipresent games can tolerate up to 1 s RTT, players of avatar games have more stringent latency requirements: 100 and 500 ms RTT for first-person and third-person avatar games, respectively (see [8] for a survey and the references therein for more details). Network latency higher than these thresholds would lead to drops in gaming performance, and could turn players away from the online games.

While our online game model (see Fig. 1) is applicable to various online games, we will only consider online 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*. A game session has a set of game settings including number of players and gaming rules. Clients in the same session exchange game-state updates. In each gaming session, one of the clients is chosen as the *host*, which runs the main gaming logic, validates the legitimacy of game-state updates, and forwards valid updates to all clients in the same session.

In some online games, a distributed session discovery method is used, where a client connects to a *master* server for a list of gaming hosts. Upon getting the list of active hosts, the client probes individual hosts and decides which host to connect to. In contrast to lobby severs, master servers maintain minimum amount of state information: most of the time, only the IPs of game hosts are provided to a new client. The difference between lobby and master servers does not affect the IRS system, and in the rest of this paper, we use lobby servers to refer to both lobby and master servers unless otherwise specified.

### B. Detour Routing for Online Games

We study the performance improvements of online games by reducing the RTT values among players in the same session. Reducing the RTT is accomplished by finding *detour paths* with one or more intermediate clients through which the gaming traffic is routed. These intermediate clients are called *relays*. Each relay client on the path adds an extra one-way delay $\alpha$ for processing and forwarding the traffic. We define a detour path as an application-level routing path that results in end-to-end delay (including the overhead) shorter than that of the direct Internet path. 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 TIVs in the Internet. For example, clients C1, C5, and H1 in Fig. 2 form a TIV. With TIVs in the Internet, we can minimize the pairwise network latency among clients in the same session 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

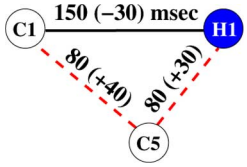


Fig. 2. Locating TIV using network coordinate systems.

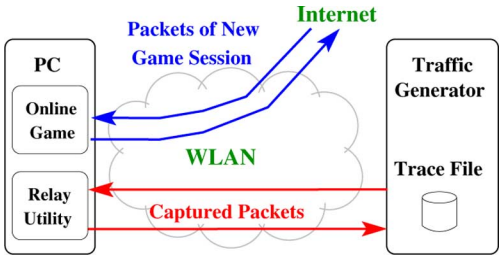


Fig. 3. Setup for measuring relay latency and traffic overheads.

a client has entries for other clients with which this client exchanges messages. For example, assume that client C1 in Fig. 2 sends messages only to the host H1. In this case, the detour routing table of C1 will have an entry $\langle H1, C5\rangle$, which means that data sent to H1 should be sent through the relay client C5.

Our first task in this paper is to explore whether this traffic relaying process yields a better gaming experience. We will study the magnitude of the expected performance improvement in different types of online 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.

### C. Quantifying Latency and Traffic Overheads

Sending game-state updates through a relay client leads to additional latency and traffic overheads. We chose two popular online games: `Starcraft 2` and `Counter Strike: Source`, and use them to quantify actual 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 [25]. The setup of our experiments is illustrated in Fig. 3. 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, which is a UDP proxy, to send game-state updates back to the traffic generator. This relay 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.[1]

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 ms in `Starcraft 2` and 6.5 ms 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-kb/s traffic overhead in `Counter Strike: Source` and 42 kb/s in `Starcraft 2`, which are insignificant to broadband access links. Our traffic overhead measurements are in line with those reported in the literature [27].

[1]We implement the relay utility in operating systems for higher accuracy. Similar utilities can also be implemented in the application level [26].

## IV. Potential of Detour Routing in Online Games

### A. Trace Collection

We need a pairwise RTT dataset of online games to quantify the potential benefits of detour routing. Unfortunately, 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 [7], or not publicly available due to business reasons, such as the Xbox dataset used in [5]. Therefore, we *had* to collect our own RTT dataset with pairwise RTT measurements among gaming clients and we are willing to share them with the research community. We compile the RTT dataset in three steps, which are detailed in the following.

*1) Collecting IPs of Game Clients:* We first identify IPs used by game clients. We use the `Qstat` utility[2] to achieve this. `Qstat` is an open-source command-line utility that allows users to browse the information of individual gaming 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 active hosts. `Qstat` then iteratively connects to each host, requests session information, and displays the information to users. We note that `Qstat` does not give IPs of *all* clients in gaming sessions, instead only the host IPs are returned. This is because modern online gaming protocols prevent hosts from disclosing client IPs to other clients, in order to avoid possible denial-of-service (DoS) attacks. We use `Qstat` to collect IPs of all hosts, and we use them to represent all gaming clients.

To collect IPs of game clients around the globe, we developed scripts to run `Qstat` on more than 550 PlanetLab nodes, and each PlanetLab node ran `Qstat` 60 times. After combining all collected IPs together, we had 28 924 distinct game client IPs. We use this set of IPs in our experiments.

*2) Measuring RTTs Among Clients:* We next describe how we measure the RTT between any two client IPs. 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 fairly accurate [28]. `King` uses DNS servers that support recursive queries for RTT estimation, and 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 gaming client IPs in these two subnets. We can do this without degrading the accuracy because `king` measures RTT between two clients using their authoritative name servers, and clients on the same /24 subnet most likely share the same authoritative name server. After the clustering, we have 8063 subnets. For each subnet, we randomly pick a client IP in that

[2][Online]. Available: http://www.qstat.org

subnet as a representative. We then conduct pairwise RTT measurements only among these 8,063 representative client IPs. We did not measure RTTs between client IPs in the same subnet. 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.

To conduct the RTT measurements, we developed scripts to run `king` on the 550+ PlanetLab nodes mentioned above for two weeks. On every PlanetLab node, we ran 12 measurement processes. Each process repeatedly measured the RTT of two randomly chosen representative client IPs, and wrote the successful measurement results into a data file. The measurement processes worked independently, and we merged all data files before analyzing them. For each IP pair, we measured their RTTs for ten times, and we report the medium RTT. Over this two-week experiment, we collected 18 884 321 RTT measurements, equivalent to about 230 GB of raw data.

We notice that, like other utilities, `king` may lead to biased measurement results. We filtered out *unreliable* measurements as follows. First, we dropped all measurements with RTTs $<$1 ms, as they are probably due to content-tracking firewalls/proxies [20]. Second, we sort all of the RTT measurements, and for each subnet pair with more than one RTT result, we used its median RTT as the final RTT. Using median RTT to filter out transient congestion and packet loss was suggested by the authors of [29], who also use the `king` utility to measure RTTs. After filtering out unreliable measurements, we have 12 930 645 distinct pairwise RTTs. These pairwise RTTs are stored in a matrix, which we call the *RTT matrix*.

*3) Limitations of the Traces:* We acknowledge that our traces are generated with some practical *inference heuristics*, which may not always be accurate. For example, some game developers may deploy dedicated game servers to host online game sessions, and their IPs will be included in our traces. The measurement errors of `King` are also well known and may lead to under estimated RTTs because authoritative name servers are likely to be better connected than game clients. Part of these measurement errors are unavoidable, unless we gain access to the source code of game clients, which is less possible to academic groups. Nonetheless, the way we quantify the potential of the IRS system can be applied to more accurate RTT dataset if they become publicly available.

### B. Notations and Performance Metrics

We study the potential of detour routing using the aforementioned RTT matrix. For a pair of clients $s$ and $t$, we let $d(s,t)$ be the RTT value between them. In Section III-C, through experiments, we found that the one-way relay overhead is about 6 ms. However, we acknowledge that the relay overhead would increase as more client pairs use the same relay client, and thus we set $\alpha = 12$ ms to be conservative. 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

$$T_k(s,t) = \begin{cases} d(s,t), & k = 0 \\ \min\limits_{\forall \text{client } r} \{d(s,r)+2\alpha+T_{k-1}(r,t)\}, & k \geq 1. \end{cases} \quad (1)$$

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, we write the $k$-hop shortest distance as

$$T_k^*(s,t) = \min_{0 \leq k' \leq k} \{T_{k'}(s,t)\} \quad (2)$$

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

$$T^*(s,t) = \min_{k \geq 0} \{T_k^*(s,t)\}. \quad (3)$$

We use the RTT matrix and (2) to recursively derive the $k$-hop shortest distances. To compute the $k^*$-hop shortest distance, we employ a modified Dijkstra's algorithm that takes relay overhead $2\alpha$ into consideration.

We define performance metrics to evaluate the potential of detour routing in the following.

1) *Pairwise RTT Reduction:* We define the $k$-hop RTT reduction between clients $s$ and $t$ as the RTT difference between their $k$-hop shortest distance and their direct distance. Similarly, we define their $k^*$-hop RTT reduction as the difference between their $k^*$-hop shortest distance and their direct distance. To derive the overall $k$-hop (or $k^*$-hop) RTT reduction, we can iterate through *all* pairs of known clients, and compute the shortest distance for each of them. This, however, may take prohibitively long time. To cope with this computational complexity, we only consider sample client pairs. That is, we randomly pick 500,000 pairs of clients and compute 0-hop, 1-hop, 2-hop, 3-hop, and $k^*$-hop shortest distances for each pair. We do not report $k$-hop shortest distances with $k \geq 4$, because we did not observe clear improvement when increasing $k$ from 3 to 4. This is because of the nontrivial overhead incurred by 4-hop, $4 \times 24 = 96$ ms. We mention that when choosing the random client pairs, we only consider those pairs with valid direct RTT values. This is because we cannot fairly compute the RTT reduction of client pairs with no direct distance.
2) *Reachability*: We define reachability as the number of players a game client can connect to and maintain good gaming quality. Different game types have different thresholds on RTTs [8], and these thresholds in turn define the reachability of a client. We compute the reachability of various RTT thresholds for different game types.
3) *Relay Load*: For each game client, we define its relay load as the number of active detour paths going over this client. The relay load indicates how much overhead the detour routing system imposes on each game client, which has a direct impact on gaming quality.

### C. Results for Potential Improvements

*1) RTT Reduction:* We compute the average RTT among all pairs for 0-hop (direct), 1-hop, 2-hop, 3-hop, and $k^*$-hop. Fig. 4(a) shows the results, where we also plot the 95% confidence interval. The figure shows that a clear reduction in the average RTT is possible using detour routing. For example, with only one relay node, the average RTT drops from about 160 ms to less than 60 ms. The figure also shows that most of the gain

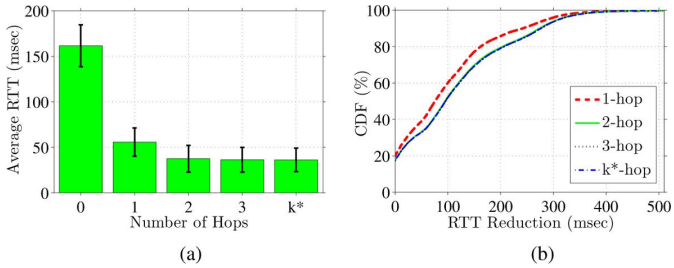


Fig. 4. Distribution of RTT reduction due to detour routing. (a) Average RTT reduction. (b) RTT reduction.

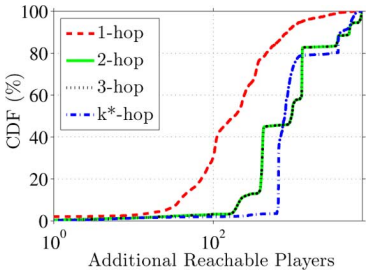


Fig. 6. Additional players allowed by detour routing.

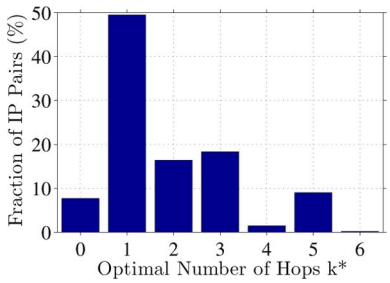


Fig. 5. Distribution of the optimal number of hops in detour paths.

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. 4(b) for 0-hop, 1-hop, 2-hop, 3-hop, and $k$*-hop detour routing. The results in Fig. 4(b) show 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-ms RTT reduction. In summary, Fig. 4 shows that using detour routing significantly reduces RTTs among gaming clients.

*2) Optimal Number of Hops ($k$*):* We define the number of optimal hops as the number of relay nodes traversed by the optimal detour path. We compute the number of hops using the Dijkstra's shortest path algorithm implemented in the Boost Graph Library (BGL).[3] We plot the distribution of the number of hops in the optimal detour paths in Fig. 5. This figure shows that almost 50% of optimal paths can be found within 1-hop.

*3) Reachability:* As mentioned in Section I, different types of games have different thresholds on RTTs, and these thresholds in turn define the reachability of clients. We compute the reachability of various RTT thresholds: 50, 100, and 200 ms. For each threshold, we iterate through every client IP and identify all other client IPs that have RTT values shorter than the threshold to the subject client. We call these clients *reachable* clients, and we count them. We repeat this process for 0-hop, 1-hop, 2-hop, 3-hop, and $k$*-hop. Then, we compute the difference between the results achieved by $k$-hop $(k > 0)$ detour routing and direct paths (0-hop). In Fig. 6, we plot the CDF curve of the results for 200-ms threshold; other results are similar. The figure shows, for example, 95% of clients can reach at least 100 additional players with 2-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 [5], [7], which *constrain* players' reachability to maintain gaming quality.

*4) Relay Load:* We report the relay loads of $k$*-hop simulations as they represent the worse-case scenario on relay overhead. Fig. 7(a) plots the mean relay load of different number of random IP pairs requesting detour paths. This figure shows that the mean relay load increases when more IP pairs need relay clients. In Fig. 7(b), we plot the CDF curve for the relay load of individual clients when 8000 random IP pairs concurrently request for detour paths. This figure illustrates that more than 90% of clients have relay load less than three, but a small number of them (about 2%) have relay load larger than 10. Fig. 7 reveals that although most clients will not be overloaded by detour routing, very few of well-connected clients may be overloaded. While well-connected clients are likely to have higher network capacity, in some rare cases, high relay load may lead to degraded gaming quality. Therefore, when designing the detour routing system (see Section V-B), we provide an *admission* mechanism for the potential relay clients to avoid quality degradation due to high relay load.

### D. Impact on Player Performance

Player performance is an indication of gaming quality: low frame rates, sluggish responsiveness, and irregular moves affect players' ability to interact with other players, and thus often lead to poor 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., [8], [11], [24], [30]–[35], define performance metrics for various games. Here, we analyze the impact of detour routing on several player performance metrics defined in previous works. We do this by leveraging on the extensive results already available in the literature. More specifically, we extract sample points from figures in [11], [30], [32], which map network latency to player performance metrics. For network latencies that do not appear in the figures, we compute the expected player performance using interpolation and extrapolation.

Due to space limitations, we only present some sample results.

*1) First-Person Shooter Games:* Beigbeder *et al.* [11] 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

[3][Online]. Available: http://www.boost.org/

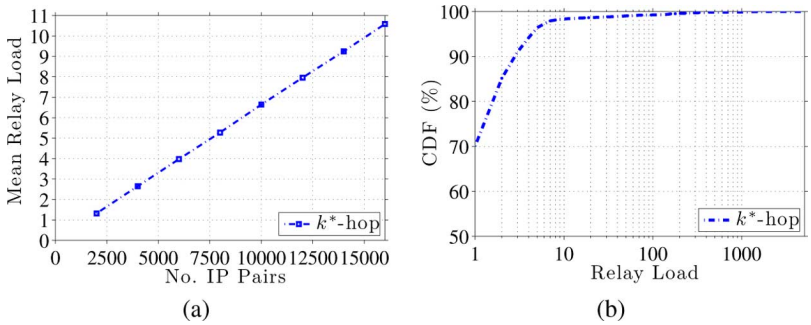


Fig. 7. (a) Mean relay load of all clients and (b) per-client relay load with 8000 IP pairs.

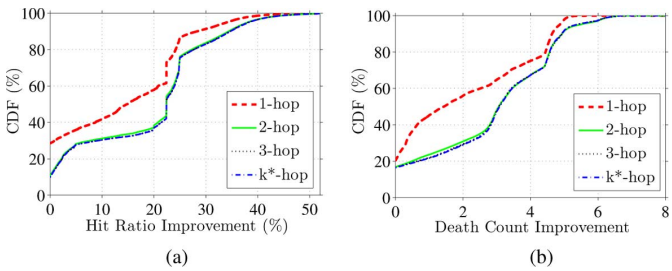


Fig. 8. Improvements in a first-person shooter game. (a) Hit ratio. (b) Death count.

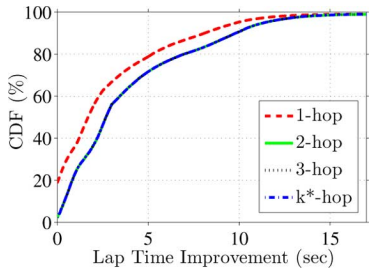


Fig. 9. Improvements in a first-person racing game.

avatar, and sees through its eyes a virtual world. Players move in the virtual world, and try to kill other players and/or bots controlled by computer algorithms. There are several modes for Unreal Tournament games. 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 number of times. 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 lead to miss shots.

Based on [11], we chose two player performance metrics that are most affected by the network latency, and we describe them in the sequel. We first 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. We then consider *death count*, which is the number of times a player dies in a 5-minute deathmatch game. In Fig. 8, we plot the impact of latency reduction on the player performance in first-person shooter games. In particular, Fig. 8(a) depicts the improvements in the hit ratio. This figure shows that 60% of players gain at least 20% hit ratio improvements with 2-hop detour routing, and some of them can improve their hit ratio by as much as 40%. Fig. 8(b) shows a similar improvement in death count.

*2) First-Person Car Racing Games:* Pantel and Wolf [30] 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. Players need to follow the racing track as close 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 chances for players to leave the track, and thus have a longer finish time.

Using a subjective study, Pantel and Wolf [30] summarize the players' gaming experience as follows. With RTT at 50 ms, players are not aware of any imposed latency. At 100 ms, players can feel the unresponsiveness when steering, but do not observe rendering issues. At 200 ms, players clearly see the frame rate is dropping, and the cars are harder to control. Finally, at 500 ms, 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, we adopt two player performance metrics from the study in [30]. We first 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 players in the study. We next 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 track. As a sample result, we present the lap time improvement in Fig. 9. This figure illustrates that 60% of players achieve 2 s or more lap time reduction with 2-hop detour routing. This is a nontrivial reduction as typical lap time in racing games is less than 14 s and the winning lap time is often within 1 ms of the second place [8].

*3) NFL Football Games:* Nichols and Claypool [32] study the impacts 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

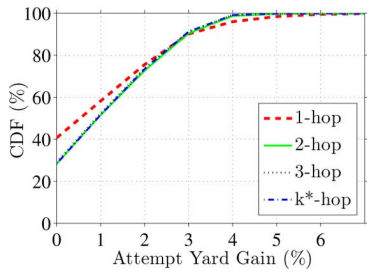


Fig. 10. Improvements in a football game.

is extracted from the figures in [32]. The extracted data 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. We plot the relative improvement on the number of yards gained per attempt in Fig. 10. The figure shows that 70% of players observe improvement in their average yard gains with 2-hop detour routing.

## V. Design of the IRS

### A. System Overview

We propose the 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. We say that client $r$ leads to a detour path from $s$ to $t$ if and only if $d(s,r)+2\alpha+d(r,t)<d(s,t)$. 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 performs 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 [29]. 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 $T$ s) 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 Section VI. We control the overhead of the algorithm by heuristically setting thresholds on the changes in the coordinates $(\Delta x)$ and RTT measurements $(\Delta d)$ below which the updates are not sent.

### B. Identifying Potential Detour Paths

To identify potential detour paths, we employ network coordinate systems, which enable us to derive pairwise RTT measurements without imposing high 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 a gossip protocol. 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 [29], 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 [20]. 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

$$\hat{e}(r) = \begin{cases} e(s,r) = d'(\mathbf{x}_s, \mathbf{x}_r) - d(s,r), & \text{if } d(s,r) \text{ is known} \\ e(r,t) = d'(\mathbf{x}_r, \mathbf{x}_t) - d(r,t), & \text{if } d(r,t) \text{ is known} \\ \frac{e(s,r)+e(r,t)}{2}, & \text{if } d(s,r) \text{ and } d(r,t) \text{ are both known} \end{cases} \tag{4}$$

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.

### C. 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 $2\alpha$, 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 $\alpha$ 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. Client $s$ then sends a *relay request* message, which includes its bandwidth requirement, to the active relay client $r$. Client $r$ checks whether it still has residual bandwidth to serve as the relay client of $s$, and sends a *relay reply* message accordingly. Through the request/reply messages, each IRS client enforces a maximum contribution bandwidth, which can be either a system parameter or derived by bandwidth measurement tools. An IRS client can also reject relay requests if its CPU load is too high to forward more packets.

### D. 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 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*. The definition of lag is different from game to game, but in general it depends on network latency and delay jitter, as well as the tolerant levels of the subject game. The precise definition of lag is therefore determined by game developers and is beyond the scope of this paper. When a lag occurs, the IRS system switches over to the next backup detour path for fast recovery. The detour path suffering from network lags becomes a backup detour path, and may be reused at a later time. 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.

### E. Handling Security Concerns

The IRS system carefully handles two types of attacks: DoS and man-in-the-middle. DoS refers to the attack where an attacker client floods many packets to an opponent in order to inflate the RTT between the victim client and its host. The victim client in turn suffers from sluggish responsiveness and may even be dropped from the game session [36], 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 network lags 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.

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 the one in [37] and prevent attacker clients from inspecting and modifying game-state updates. Since a malicious relay player does not know the actual contents of game-state updates, the relay player cannot selectively delay critical updates, such as firing a weapon. Thus, an attacher can only delay all (or random) updates, and the IRS client on the victim client would notice network lags and switch to backup detour paths. Hence, our IRS system can handle man-in-the-middle attacks.

### F. The Proposed Algorithm

Fig. 11 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

**SRTT: Shortest RTT Algorithm**

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

1. // **Client**, input: dst $t$, $\mathbf{r}$
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 $\mathbf{r}$ for detour path
8. **fall back** to the next detour path when lag happens

Fig. 11. Proposed algorithm.

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.

### G. 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 $O((K+1)\log(K+1))$ 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 (4). 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.* [29] 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. Because $T$ is in the order of seconds, the network overhead between clients and the server is negligible. Moreover, the network overhead among clients is also small. First, a relay client contributes a small bandwidth (about 40 kb/s as reported in Section III) toward every client using it as the relay client. Second, in typical network coordinate systems, a client sends

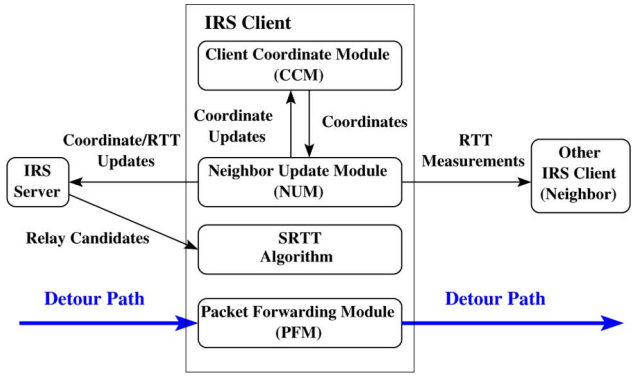


Fig. 12. IRS client architecture.

control messages to its neighbors infrequently. For example, as presented in Section VI, our experimental results using Pyxida [38] show that each client sends a probing message every 16 s on average. Hence, the network overhead among clients is also negligible.

## VI. Implementation and Experimental Results

Here, we first describe a real implementation of the IRS system. Then, we present performance results obtained from deploying the system on a wide-area testbed (PlanetLab) and on several home computers with DSL and cable modem access links.

### A. Implementation

The IRS system is implemented in Java code and it 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.

*1) IRS Client:* The IRS client consists of three modules: 1) neighbor update module (NUM); 2) client coordinate module (CCM); and 3) packet forwarding module (PFM). Fig. 12 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. 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 implemented the CCM based on the open source Pyxida project [38], which implements the Vivaldi [29]

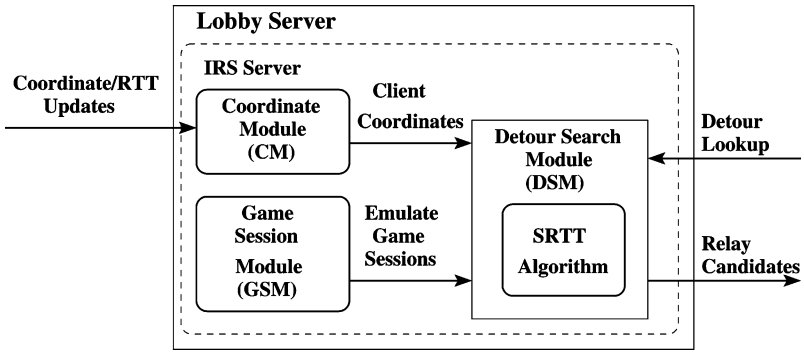


Fig. 13. IRS server architecture.

algorithm. We modified the Pyxida implementation to make it suitable for our IRS system. For example, the original Pyxida employs a gossip protocol for bootstrapping a new node, while in the modified version, a server-aided bootstrapping mechanism is realized to leverage on the coordinates stored at the IRS server. The default Pyxida parameters are used throughout the experiments if not otherwise specified.

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 re-transmitted to the destination client. We generate random synthetic packets 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 synthetic packets have random packet sizes between 25 and 100 bytes, while a random number of packets are grouped into bursts with an inter-burst time of 64 ms.[4] The PFM switches over to backup detour paths whenever network lags occur.

*2) IRS Server:* The IRS server consists of three modules: 1) coordinate module (CM); 2) detour search module (DSM); and 3) game session module (GSM). Fig. 13 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. The 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 join and leave game sessions. To emulate typical game matches, the GSM module periodically creates a new random game session every 15–60 s, and each game session lasts between 3–10 min. To be conservative, we chose rather short game sessions because they lead to more client dynamics, which in turn 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 Section IV-A and derived an empirical probability mass function (PMF) for the number of players per session, which is plotted in Fig. 14. We then followed this probability distribution to find a random number of players to form a game session. The GSM randomly chooses that many IRS clients from all active clients, and it selects a random host from them. 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.

[4]We tried a few other inter-burst times in our experiments, but did not see noticeable difference. We chose 64 ms to align it with Pyxida's update frequency to simplify the implementation.

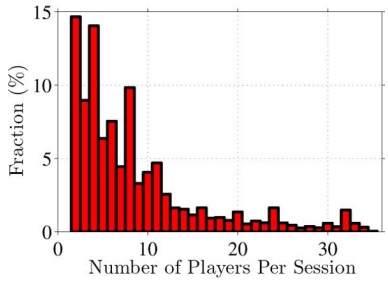


Fig. 14. Number of players in each game session.

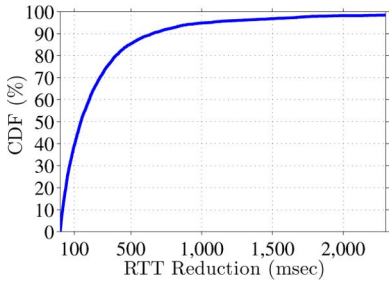


Fig. 15. RTT reduction achieved by the IRS system.

## B. 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 $K = 32$, $\Delta d = \Delta x = 64$ ms, $T = 60$ s, 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 3000 game sessions with length 3–10 min were randomly created with the number of players per session following an empirically-driven probability distribution (see Fig. 11). For each game session, we collected real RTTs (including relay overhead) of the detour paths computed by the DSM and saved

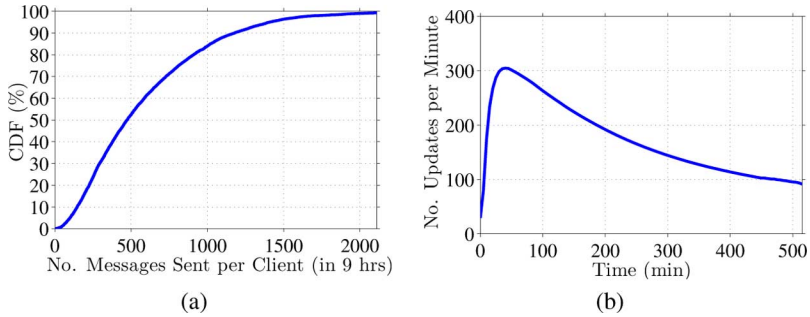


Fig. 16. Overhead incurred by the IRS system: (a) number of messages sent by each client in a 9-h experiment and (b) updates per minute received by the server.

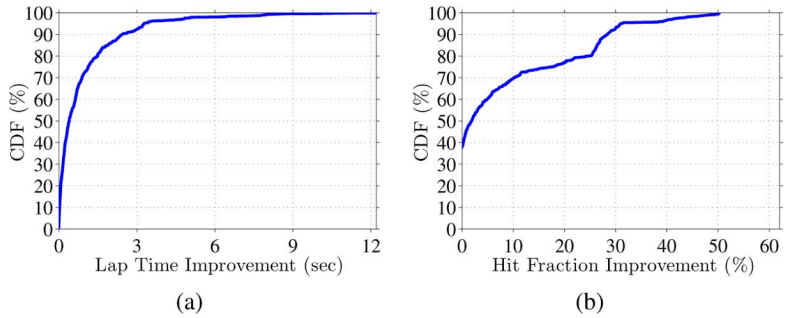


Fig. 17. Player performance: (a) lap time in a racing game and (b) hit fraction in a shooter game.

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.

### C. Experimental Results From PlanetLab

*1) RTT Reduction:* We report the RTT reduction achieved by our IRS implementation. We compute the RTT reduction of all game sessions, and plot the CDF in Fig. 15. 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 ms or higher RTT reduction.

*2) 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-h experiment, and we plot the CDF in Fig. 16(a). This figure shows that almost all clients imposed less than 2000 messages in a 9-h time period, which is about one packet every 16 s 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. 16(b). This figure shows that the number of update messages is fairly small: up to 300 per minute are observed. We mention that this experiment mimics the worst-case scenario where a large number $(500+)$ of clients simultaneously log in to the game server. In reality, the login times of clients are scattered over a longer period. Under this worse case scenario, the total peak load on the server is only five messages per second, which is equivalent to 4.32 kb/s (each update message contains two coordinates and up to $N = 64$ RTTs, and thus is 108 byte long including all headers). Hence, Fig. 16(b) 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.

*3) Player Performance:* We compute the expected player performance improvement due to the RTT reduction achieved by the IRS system using the same calculation as detailed in Section IV-D. We present CDFs of two sample player performance metrics: lap time and hit fraction. We plot the lap time improvement in Fig. 17(a) and the hit fraction improvement in Fig. 17(b). Fig. 17 shows that 30% of players can reduce their lap times by more than 1 s 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. 17 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.

*4) 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. 18. 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 $N$ neighbors restricted to 32.

TABLE I
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 |

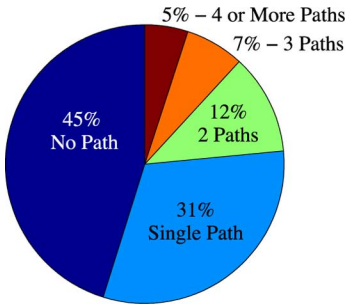


Fig. 18. Number of detour paths found by the IRS system.

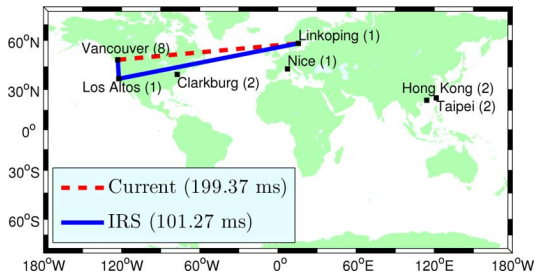


Fig. 19. Sites in the residential measurement experiments.

*5) Summary:* Our experimental results clearly show that the IRS system improves the online gaming performance from several aspects: 1) it significantly reduces RTTs in game sessions; 2) it imposes negligible network and processing overheads; 3) it increases the gaming quality and player performance; and 4) it allows many clients to have backup detour paths to cope with system dynamics.

### D. Residential Deployment

Although our PlanetLab experiments illustrate 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 $K = 8$, $\Delta d = \Delta x = 64$ ms, $T = 60$ s, and $N = 8$. The geographic locations of the 17 players are illustrated in Fig. 19. 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 eight detour paths among them. Fig. 19 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 ms 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 ms. 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 ms. The residential deployment shows that our IRS implementation works even among home computers with residential access links, which may have high last-mile delays.

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.[5] 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 [12], [13] for movement predictions and smoothing algorithms [39] 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 game latency simulator (GLS) system implemented in [40]. 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 1-h 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 [40]. 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 I. This table clearly shows that residential users with cable modem

[5][Online]. Available: http://bzflag.org/

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 m to 1 m. Our experimental results illustrate that the IRS system works: 1) in residential networks and 2) on online games that have implemented modern latency compensation algorithms.

## VII. Conclusion

We conducted a large-scale measurement study to quantify the potential gain of using detour routing in online games. We studied the pairwise RTT reduction, reachability (additional number of players that can be reached), and RTT reduction of 0-hop, 1-hop, 2-hop, 3-hop, and $k^*$-hop detour routing, where $k^*$ is the optimal number of hops. Our results showed that significant RTT reduction can be achieved by detour routing, and simple 1-hop detour routing is sufficient for most practical cases, as it achieves up to 80% of the RTT reduction achieved by the optimal $k^*$-hop detour routing. We analyzed the benefits of detour routing on the application performance metrics. We showed that detour routing leads to significant improvement in several player performance metrics, such as: 1) hit fraction and death count in first-person shooter games; 2) lap completion time and frequency of leaving the track in car racing games; and 3) attempt gain in football games. For example, we observed that 60% of players in first-person shooter games gain 20% hit ratio improvements, and some of them can improve their hit ratio by up to 40%.

We then presented a complete IRS system, which 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 measurements to identify potential detour paths between any two clients. Second, the source client conducts end-to-end RTT measurements to the 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 implemented the IRS system and deployed it on more than 500 PlanetLab nodes and on several home computers with residential access links. Our experimental results indicate that the IRS system efficiently reduces RTTs among game clients, yet imposes negligible network and processing overheads. Smaller RTTs result in better gaming quality and higher player matchability, which are two major QoS metrics in online games. We observed that more than 60% of game sessions achieve 100 ms or more RTT reduction compared with the current gaming networks.

## References


[1] 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*, vol. 19, no. 1, pp. 50–59, Jan./Feb. 1999.

[2] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality and routing policy violations in the Internet," in *Proc. Conf. Passive Active Netw. Meas. (PAM'09)*, Seoul, Korea, Apr. 2009, pp. 45–54.

[3] G. Wang, B. Zhang, and T. Ng, "Towards network triangle inequality violation aware distributed systems," in *Proc. ACM Conf. Internet Meas. (IMC'07)*, San Diego, CA, Oct. 2007, pp. 175–188.

[4] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games*, 1st ed. New York: Wiley, 2006.

[5] S. Agarwal and J. Lorch, "Matchmaking for online games and other latency-sensitive P2P systems," in *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009, pp. 315–326.

[6] C. Chambers, W. Feng, W. Feng, and D. Saha, "A geographic redirection service for on-line games," in *Proc. ACM Multimedia*, Berkeley, CA, Nov. 2003, pp. 227–230.

[7] M. Claypool, "Network characteristics for server selection in online games," in *Proc. SPIE/ACM Multimedia Computing and Networking (MMCN'08)*, San Jose, CA, Jan. 2008, pp. 681808:1–681808:12.

[8] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, pp. 40–45, Nov. 2006.

[9] N. Baughman and B. Levine, "Cheat-proof playout for centralized and distributed online games," in *Proc. IEEE INFOCOM*, Anchorage, AL, Apr. 2001, pp. 22–26.

[10] F. Cecin, C. Geyer, S. Rabello, and J. Barbosa, "A peer-to-peer simulation technique for instanced massively multiplayer games," in *Proc. IEEE Symp. Distrib. Simul. Real-Time Applic. (DS-RT'06)*, Washington, DC, Oct. 2006, pp. 43–50.

[11] 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. ACM SIGCOMM Workshop Netw. Syst. Support for Games (NetGames'04)*, Portland, OR, Aug. 2004, pp. 144–151.

[12] Y. Bernierr, "Latency compensating methods in client/server in-game protocol design and optimization," in *Proc. Game Developers Conf.*, San Jose, CA, Mar. 2001.

[13] J. Vogel and M. Mauve, "Consistency control for distributed interactive media," in *Proc. ACM Multimedia*, Ottawa, ON, Canada, Sep. 2001, pp. 221–230.

[14] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Proc. ACM SIGCOMM Workshop Netw. Syst. Support for Games*, Portland, OR, Aug. 2004, pp. 161–165.

[15] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, "Measurement and estimation of network QoS among peer Xbox 360 game players," in *Proc. Conf. Passive Active Netw. Meas.*, Cleveland, OH, Apr. 2008, pp. 41–50.

[16] G. Armitage, "Optimising online FPS game server discovery through clustering servers by origin autonomous system," in *Proc. ACM Workshop Netw. Oper. Syst. Support for Digital Audio and Video*, Braunschweig, Germany, May 2008, pp. 3–8.

[17] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 131–145, Dec. 2001.

[18] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz, "OverQos: An overlay based architecture for enhancing Internet QoS," in *Proc. USENIX Symp. Netw. Syst. Design and Implementation*, San Francisco, CA, Mar. 2004, pp. 71–84.

[19] S. Ren, L. Guo, and X. Zhang, "ASAP: An AS-aware peer-relay protocol for high quality VoIP," in *Proc. IEEE Conf. Distrib. Computing Syst.*, Lisboa, Portugal, Jul. 2006, pp. 70–80.

[20] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee, "Symbiotic relationships in Internet routing overlays," in *Proc. USENIX Symp. Networked Syst. Design and Implementation*, Boston, MA, Apr. 2009, pp. 469–480.

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

[22] C. Ly, C. Hsu, and M. Hefeeda, "Improving online gaming quality using detour paths," in *Proc. ACM Multimedia*, Florence, Italy, Oct. 2010, pp. 55–64.

[23] T. Henderson, "The effects of relative delay in networked games," Ph.D. dissertation, Dept. Comput. Sci., Univ. of London, London, U.K., 2003.

[24] S. Zander, I. Leeder, and G. Armitage, "Achieving fairness in multiplayer network games through automated latency balancing," in *Proc. ACM SIGCHI Int. Conf. Adv. Comput. Entertainment Technol. (ACE'05)*, Valencia, Spain, Jun. 2005, pp. 117–124.

[25] C. Ly, "Latency reduction in online multiplayer games using detour routing," M.S. thesis, Sch. Computing Sci., Simon Fraser Univ., Surrey, BC, Canada, 2010.

[26] K. Chen and J. Lou, "Toward an understanding of the processing delay of peer-to-peer relay nodes," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. (DSN'08)*, Anchorage, AK, Jun. 2008, pp. 410–419.

[27] W. Feng, F. Chang, W. Feng, and J. Walpole, "A traffic characterization of popular on-line games," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 488–500, Jun. 2005.
[28] K. Gummadi, S. Saroiu, and S. Gribble, "King: Estimating latency between arbitrary Internet end hosts," in *Proc. ACM SIGCOMM Internet Meas. Workshop*, Marseille, France, Nov. 2002, pp. 5–18.
[29] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. ACM SIGCOMM*, Portland, OR, Sep. 2004, pp. 15–26.
[30] L. Pantel and L. Wolf, "On the impact of delay on real-time multiplayer games," in *Proc. ACM Workshop Netw. Oper. Syst. Support Digital Audio Video*, Miami, FL, May 2002, pp. 23–29.
[31] T. Fritsch, H. Ritter, and J. Schiller, "The effect of latency and network limitations on MMORPGs: A field study of Everquest2," in *Proc. ACM SIGCOMM Workshop Netw. Syst. Support for Games*, Hawthorne, NY, Oct. 2005, pp. 1–9.
[32] J. Nichols and M. Claypool, "The effects of latency on online Madden NFL Football," in *Proc. ACM Workshop Netw. Oper. Syst. Support Digital Audio Video*, Kinsale, Ireland, Jun. 2004, pp. 146–151.
[33] M. Claypool, "The effect of latency on user performance in real-time strategy games," *J. Comput. Netw.*, vol. 49, no. 1, pp. 52–70, Sep. 2005.
[34] G. Armitage, "An experimental estimation of latency sensitivity in multiplayer Quake 3," in *Proc. IEEE Int. Conf. Netw.*, Sydney, Australia, Sep. 2003, pp. 137–141.
[35] T. Henderson, "Observations on game server discovery mechanisms," in *Proc. ACM SIGCOMM Workshop Netw. Syst. Support Games*, Portland, OR, Aug. 2002, pp. 144–151.
[36] J. Yan and B. Randell, "A systematic classification of cheating in online games," in *Proc. ACM SIGCOMM Workshop Netw. Syst. Support Games*, Hawthorne, NY, Oct. 2005, pp. 1–9.
[37] C. Monch, G. Grimen, and R. Midstraum, "Protecting online games against cheating," in *Proc. ACM SIGCOMM Workshop Netw. Syst. Support Games*, Singapore, Oct. 2006, pp. 1–11.
[38] J. Ledlie, P. Pietzuch, M. Mitzenmacher, and M. Seltzer, "Network coordinates in the wild," in *Proc. USENIX Symp. Netw. Syst. Design Implementation*, Cambridge, MA, Apr. 2007, pp. 299–312.
[39] K. Lin, M. Wang, J. Wang, and D. Schab, "The smoothing of dead reckoning image in distributed interactive simulation," in *Proc. AIAA Flight Simulation Technol. Conf.*, Baltimore, MD, Aug. 1995, pp. 83–87.
[40] W. Palant, C. Griwodz, and P. Halvorsen, "Evaluating dead reckoning variations with a multi-player game simulator," in *Proc. ACM Workshop Netw. Oper. Syst. Support Digital Audio Video*, Newport, RI, May 2006, pp. 20–25.

**Cong Ly** received the B.Sc. and M.Sc. degrees from Simon Fraser University, Surrey, BC, Canada, in 2002 and 2010, respectively.

His research interests include online gaming, wireless networks, and multimedia applications. In addition to publications, he served as a Game and Network Programmer for Crash Tag Team Racing, Scarface: The World is Yours, and Prototype. He is currently a Technical Director with Two79 Inc., Vancouver, BC, Canada, an innovative company that specializes in creating unique online experiences.

**Cheng-Hsin Hsu** (S'09–M'10) received the B.Sc. and M.Sc. degrees from National Chung-Cheng University, Taiwan, in 1996 and 2000, respectively, the M.Eng. degree from the University of Maryland, College Park, in 2003, and the Ph.D. degree from Simon Fraser University, Surrey, BC, Canada, in 2009.

He is a Senior Research Scientist with Deutsche Telekom R&D Lab USA, Los Altos, CA. His research interests are in the area of multimedia networking and distributed systems.

**Mohamed Hefeeda** (S'01–M'04–SM'09) received the B.Sc. and M.Sc. degrees from Mansoura University, Egypt, in 1994 and 1997, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2004.

He is an Associate Professor with the School of Computing Science, Simon Fraser University, Surrey, BC, Canada, where he leads the Network Systems Lab. His research interests include multimedia networking over wired and wireless networks, peer-to-peer systems, mobile multimedia, and Internet protocols. In addition to publications, he and his students have developed actual systems, such as pCache, svcAuth, pCDN, and mobile TV testbed. He serves as the Preservation Editor of the ACM Special Interest Group on Multimedia (SIGMM) web magazine. He served as the program chair of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2010) and as a program cochair of the International Conference on Multimedia and Expo (ICME 2011). In addition, he has served on many technical program committees of major conferences in his research areas, including ACM Multimedia, ACM Multimedia Systems, and the IEEE Conference on Network Protocols (ICNP). He is on the editorial boards of the *ACM Transactions on Multimedia Computing, Communications and Applications*, the *Journal of Multimedia*, and the *International Journal of Advanced Media and Communication.*

Dr. Hefeeda was the recipient of the Best Paper Award at the IEEE Innovations 2008 Conference for his paper on the hardness of optimally broadcasting multiple video streams with different bitrates. The mobile TV testbed software developed by his group won the Best Technical Demo Award at the ACM Multimedia 2008 Conference.