Matchmaking for Online Games and Other Latency-Sensitive P2P Systems

Sharad Agarwal Jacob R. Lorch
Microsoft Research

ABSTRACT– The latency between machines on the Internet can dramatically affect users' experience for many distributed applications. Particularly, in multiplayer online games, players seek to cluster themselves so that those in the same session have low latency to each other. A system that predicts latencies between machine pairs allows such matchmaking to consider many more machine pairs than can be probed in a scalable fashion while users are waiting. Using a far-reaching trace of latencies between players on over 3.5 million game consoles, we designed *Htrae*, a latency prediction system for game matchmaking scenarios. One novel feature of Htrae is its synthesis of geolocation with a network coordinate system. It uses geolocation to select reasonable initial network coordinates for new machines joining the system, allowing it to converge more quickly than standard network coordinate systems and produce substantially lower prediction error than state-of-the-art latency prediction systems. For instance, it produces 90th percentile errors less than half those of iPlane and Pyxida. Our design is general enough to make it a good fit for other latency-sensitive peer-to-peer applications besides game matchmaking.

Categories and Subject Descriptors – C.2.4 [Computer Systems Organization]: Computer Communication Networks – Distributed Systems; K.8.0 [Personal Computing]: General – Games
General Terms – Algorithms, design, experimentation, measurement, performance
Keywords – latency estimation, matchmaking, network coordinates, online gaming

1. INTRODUCTION

Online gaming is a rapidly growing industry with revenues exceeding that of the entire movie business [29]. Appealing to increasingly discriminating game players requires careful attention to their chief concerns, particularly *lag*, the perceived time between an action and its effect. For the majority of games, the primary contributor to lag is direct communication between participants' game machines over the Internet. To reduce lag and hence improve the game experience, it is critical to perform accurate *matchmaking*—selecting groups of players with low latency to each other.

In typical matchmaking, each player sends network probes to potential game hosts and selects one host based on latency and bandwidth. Since probing consumes time and bandwidth, and players have limited patience for matchmaking, only a small fraction of potential hosts can be probed. Furthermore, in games where player machines communicate directly with each other rather than only with the host, it is prohibitive to probe all the potential paths traffic will take, which can be quadratic in the number of online players. For these reasons, game matchmaking can benefit from *latency prediction*—determining the latency between machines without any probing. In this paper, we develop a new latency prediction system, called *Htrae*, suited to game matchmaking. Using a trace of 50 million latency measurements between 3.5 million players of the popular game Halo 3, we show that our system works better than state-of-the-art latency prediction systems.

A novel feature of Htrae is its merging of two disparate approaches to latency prediction, network coordinate systems (NCS) and geolocation. An NCS assigns each machine a coordinate in a virtual metric space, such that the distance between two machines' coordinates approximates the round-trip time (RTT) between them. Because initial conditions are random, NCSes take time to converge to a reasonable state, and usually converges to a sub-optimal local minimum. Geolocation, on the other hand, determines each machine's location on the planet and uses physical distance as a predictor of network delay. This approach does not model the impact of different routing efficiencies on different paths and disadvantages machines whose location is imprecisely known. Htrae combines these approaches by assigning each machine a coordinate on a virtual Earth based on its physical location, but allowing this coordinate to shift based on observations of actual latency. In so doing, we gain the benefits of both approaches: the realistic initial conditions of geolocation, combined with the ability of an NCS to converge to a state with greater predictive power.

Htrae also includes enhancements that are beneficial in game matchmaking. Triangle inequality violations and different network access latencies for different machines are scenarios that traditionally pose problems for an NCS. They occur often enough in our game traces that we use simple heuristics to help combat their effects. We also observe that since reliably measuring RTT for game purposes can consume a fair amount of bandwidth, the overhead of one additional message to notify the probee of the result is low, and quite worthwhile considering its usefulness to the recipient.

Our evaluation for game matchmaking shows that Htrae has substantially better predictive power than other latency prediction systems. For instance, its 90th percentile of prediction error is 71 ms, compared to 176 ms for Pyxida [13] and 162 ms for iPlane [18], even when restricting consideration only to the 23% of pairs iPlane can make a prediction for. Also, when searching for the best peer game server, Htrae picks the best one 70% of the time, compared to only 35% for Pyxida and 55% for iPlane. Even though it does not always find the best one, 95% of the time it picks one at most 47 ms worse than optimal, compared to 184 ms for Pyxida and 131 ms for iPlane. In scenarios allowing a limited amount of probing to select the best server, Htrae finds one with under 75 ms of latency 68% of the time, compared to 44% for Pyxida and 41% when no latency prediction is used. Our results show that Htrae will have tremendous impact on the playability of games, and may also be useful for other latency-sensitive peer-to-peer applications.

The contributions of our work are as follows:

- We propose a novel latency prediction system, Htrae, a hybrid of geolocation and network coordinate systems that achieves the benefits of both.
- Based on extensive traces of latencies between game consoles, we thoroughly evaluate the accuracy, convergence and drift of Htrae.

- We also evaluate in detail the effectiveness of various implementation details of Htrae, including a component that corrects for triangle-inequality violations in an NCS.

The rest of this paper is structured as follows. §2 details the design of our system, Htrae, including how we merge geolocation with an NCS. §3 shows how we implemented Htrae. §4 describes the traces of game matchmaking probes we collected and presents the methodology we use to evaluate our system. §5 gives the results of that evaluation. §6 discusses these results as well as avenues for future work. §7 describes related work, and, finally, §8 concludes.

2. DESIGN

Htrae is a hybrid between two approaches to latency prediction: geolocation and network coordinate systems.

Geolocation predicts latency based on the real-world distance between two physical machines, which many researchers have found is a strong predictor of RTT [10, 14, 24]. However, this correlation is weak, especially for the home machines typically found in games; for instance, we found a correlation coefficient of only +0.56 among Halo 3 players. Furthermore, if geolocation inaccurately judges the location of some player's machine, it will consistently give that player poor performance.

An NCS gives each machine a coordinate in a virtual space, such that the distance between two coordinates is an estimate of their RTT. Coordinates are dynamically adjusted based on observations of RTT so as to make estimation more accurate. Unfortunately, accurately embedding the Internet graph in a virtual coordinate space is difficult. One reason is that the Internet has many routing inefficiencies, some that lead to *triangle-inequality violations* (TIVs), where two nodes have a greater RTT than the sum of their respective RTTs to some other node. Coordinate spaces cannot have such violations [16, 36]. Furthermore, embeddings are often sensitive to initial conditions [12], since they can fall into one of many imperfect local minima in the space of possible coordinate assignments [28].

Our insight is that these two approaches, NCS and geolocation, are complementary, i.e., we can combine them in a way that mitigates disadvantages of both. We do this by *geographic bootstrapping*, i.e., initializing NCS coordinates to correspond to the locations of the nodes in actual space. Our approach improves on an NCS because it provides better initial conditions, and improves on geolocation because its dynamic coordinate refinement can correct inaccurate or missing information. Essentially, our coordinate system is a rough representation of Earth, modified to better predict Internet latencies; the name Htrae comes from a warped version of Earth in a certain comic-book universe.

2.1 Geographic bootstrapping

Geographic bootstrapping requires a virtual space with a known relationship to the real world. Therefore, in Htrae, we use latitude and longitude on a virtual Earth. Also, as in Vivaldi [6], we use a virtual *height*, which represents the component of latency a machine experiences on all its paths, e.g., due to its Internet access link. The predicted RTT between two machines is the great-circle distance between their virtual locations, times 0.0269 ms/mile, plus the sum of the two machines' heights. We obtained the factor 0.0269 from Figure 1, which shows the relationship between distance and median RTT. It shows that this relationship is strongly linear, with $R^2 = 0.976$, slope 0.0269 ms/mile, and y-intercept 63.32 ms. This factor of 0.0269 ms/mile is about five times greater than the inverse speed of light. A factor of two is expected since we use round-trip latency, and the remaining factor of 2.5 suggests other causes besides the speed of light.

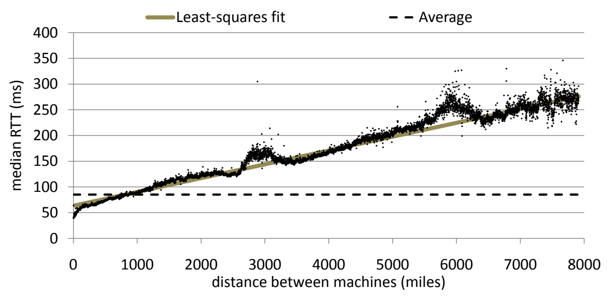

Figure 1. Correlation between distance and RTT. Each point is the median RTT among machines with the same distance, rounded to the nearest mile. RTT data is from Halo 3 players from March 1–31, 2008, and distance data is from MaxMind's IP-to-geo database. The least-squares fit weights each point by its number of contributing machine pairs.

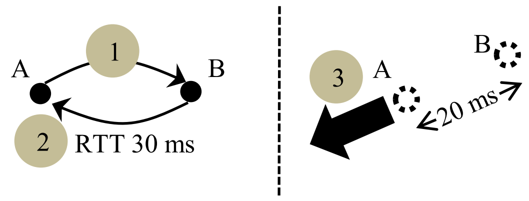

Figure 2. Overview of updating coordinates. (1) Node A sends a message to B. (2) B responds, and A measures the RTT. (3) If the distance between their virtual coordinates is too low/high, A applies a virtual force to its coordinate, moving it away from/toward B's.

When a machine joins the system, it determines its latitude and longitude in one of various ways, e.g., by looking up its IP address in a database. It then uses this as its initial location, along with height equal to half the least-squares y-intercept, or 31.66 ms. If it cannot determine its location, it uses latitude and longitude 0. From then on, whenever it determines its RTT to another machine, it applies a virtual force to its coordinates, either toward the other machine if the RTT was unexpectedly small or away if it was unexpectedly large, as shown in Figure 2. The magnitude of this force is calculated as in Vivaldi [6], but adapted to using spherical coordinates instead. See Figures 3 and 4 for details. Note that the authors of Vivaldi tried using spherical coordinates in their system, but found them not to work well; we will see in §5.2 that they work in Htrae because of geographic bootstrapping.

As in Vivaldi, each node also maintains an estimate of the uncertainty of its coordinates, a weighted moving average of the error observed between expected RTTs and observed RTTs. When coordinates improve such that distances better predict observed RTTs, uncertainty will decrease. This uncertainty is used to decide how strong a force to apply when updating coordinates: the greater the moving node's uncertainty, and the lower the other node's uncertainty, the stronger the force will be, as shown in Figure 3. However, unlike Vivaldi, we do not always initialize uncertainty to 100%. If a machine initializes its coordinates based on geographic location, it uses initial uncertainty of 29.2%, the average relative error obtained from using geolocation alone.

RTT measurements are prone to errors, which can harm the correctness and stability of network coordinates [13]. Therefore, as in Pyxida [13], we do not use a single RTT measurement when adjusting coordinates but rather an aggregate of multiple measurements. Specifically, we use the median of five back-to-back RTT measurements; our traces of Xbox LIVE probes report only these medians.

$$\begin{aligned}
w_s &\leftarrow w_A/(w_A+w_B)\\
\varepsilon &\leftarrow (\|\vec{x}_A-\vec{x}_B\|-l_{AB})/l_{AB}\\
w_A &\leftarrow c_e w_s \varepsilon + (1-c_e w_s) w_A\\
\vec{x}_A &\leftarrow \vec{x}_A + c_c w_s (\|\vec{x}_A-\vec{x}_B\|-l_{AB})\, u(\vec{x}_B-\vec{x}_A)
\end{aligned}$$

Figure 3. Vivaldi update algorithm, run after node A learns the RTT l_{AB} to node B. Here, $\vec{x}_N$ is the virtual location of node N, w_N is the uncertainty of node N's coordinates, $u(\vec{y})$ is the unit vector in the direction of $\vec{y}$, and c_c and c_e are algorithmic constants.

2.2 TIV avoidance and history

Triangle inequality violations in Internet delay are typically caused by inefficient routing between two nodes, resulting in more delay between them than the sum of delays via some other intermediate nodes. Game consoles on singly-homed residential connections can be particularly susceptible because they are restricted by their ISPs' routing policies, as opposed to a server in a datacenter that has multiple upstream ISPs to choose from. For example, in our dataset, a game console in Vancouver, BC measured a 378 ms median RTT to a console in Tukwila, WA. This is an unusually high delay for a distance of only 141 miles. Indeed, around the same time, the Vancouver console measured a median RTT of only 47 ms to a console in Pleasanton, CA (distance of 959 miles) and the Tukwila console had a RTT of 75 ms to Los Angeles (1,100 miles away).[1] Such violations are relatively infrequent—in this example, the Vancouver and Tukwila consoles were involved in 78 RTT measurements to other nodes, none of which appear to be TIVs.

TIVs are problematic for both NCSes and geolocation-based latency estimation. Geolocation will under-estimate the latency between two nodes, as evident from the above example. Similarly, an NCS will underestimate the latency because the coordinates for the two nodes may have converged to stable positions based on the vast majority of non-TIV latency measurements. Further, when a TIV measurement is used to update an NCS, the coordinates of the nodes involved will be pushed further apart by the resulting force, thereby worsening their positions with respect to other nodes.

Some prior solutions to this problem have been shown to either worsen prediction accuracy or not scale in distributed settings [33]. We instead use a heuristic similar to TIV Alert proposed in [33]. When updating a node's coordinate, if the measured latency exceeds the predicted latency by δ, we skip this coordinate update. We apply this heuristic only when both nodes in question have uncertainty lower than the geographic bootstrapping default of 29.2%, and update their uncertainties based on this measurement. This guards against slowing down convergence for nodes with poor initial coordinates. We have empirically determined that a δ of 100 ms works best for our client population.

While this heuristic reduces the impact of TIVs on the quality of node coordinates, it does not help improve latency prediction of TIV edges. To address that problem, we rely on *history prioritization*. Every time a node learns the RTT to another, it saves this RTT in its *history*. When a prediction is needed, if there is history for the destination node, we use the most recently measured RTT instead of the RTT predicted by the coordinates. Note that we use the most recent one because we assume robust RTT measurements, as discussed earlier. The past RTT can be a good estimate of future RTT because RTTs on the Internet can be very stable. In our data, for about 95% of nodes that measured RTT multiple times between themselves for as long as 50 days, the coefficient of variation in RTT was under 0.2. History prioritization is particularly useful for node pairs that cause TIVs, because latency estimates from their coordinates will be inaccurate.

[1]We found the physical locations of these consoles by using a geolocation database (§ 4.1), and confirming by looking up the DNS names of nearby routers using traceroute.

$$\begin{aligned}
r &\leftarrow 1-c_c w_s (\|\vec{x}_A-\vec{x}_B\|-l_{AB})/\|\vec{x}_A-\vec{x}_B\|\\
d &\leftarrow \cos^{-1}[\cos(\phi_A)\cos(\phi_B)+\\
&\qquad \sin(\phi_A)\sin(\phi_B)\cos(\lambda_B-\lambda_A)]\\
\gamma &\leftarrow \tan^{-1}\left[\frac{\sin(\phi_B)\sin(\phi_A)\sin(\lambda_B-\lambda_A)}{\cos(\phi_A)-\cos(d)\cos(\phi_B)}\right]\\
\phi_A &\leftarrow \cos^{-1}[\cos(rd)\cos(\phi_B)+\sin(rd)\sin(\phi_B)\cos(\gamma)]\\
\beta &\leftarrow \tan^{-1}\left[\frac{\sin(\phi_B)\sin(rd)\sin(\gamma)}{\cos(rd)-\cos(\phi_A)\cos(\phi_B)}\right]\\
\lambda_A &\leftarrow \lambda_B-\beta
\end{aligned}$$

Figure 4. Adaptation of Vivaldi update algorithm to spherical coordinates. Here, $\vec{x}_N$, the coordinates for node N, consist of ϕ_N, the latitudinal distance in radians between node N and the North Pole, and λ_N, the longitude in radians of node N. We compute r, the ratio between the desired and current distance between nodes A and B; d, the current distance in radians between them; γ, the angle from the North Pole to B to A (which stays the same as A moves); and β, the angle from B to the North Pole to A's new location. Note that some special cases are not shown.

It might seem unlikely that among millions of players, a player will ever probe the same player twice, but it does happen. This is presumably because the factors that make two players good candidates for matchmaking, such as liking the same type of game, having similar skill level, and liking to play at the same time of day or week, change infrequently.

2.3 AS correction

There are many autonomous systems (ASes) serving game players, so most pairs of players will belong to different ASes. Thus, we can expect our coordinate system will automatically tune itself to predicting latencies assuming endpoints are in different ASes. Unfortunately, this means that when endpoints are in the same AS, the faster resulting routing will not be reflected in the coordinate system, leading to systematic overestimation of the RTT of such paths. This can be particularly frustrating for game players, since it may lead to unnecessary exclusion of some of the closest players.

To understand our solution, recall that Htrae, like other NCSes, augments coordinates with a virtual height reflecting the latency a machine incurs on essentially all of its paths [6]. For home machines, we can loosely consider this latency to have two main components. First, a packet incurs latency in the so called "last-mile." In the case of a machine with a broadband DSL connection, this is the latency between the machine and the next IP-level device, such as the broadband remote access server (BRAS). The second component is the remaining latency through the high-speed network core for the AS the machine resides in. Typically, this would be a set of core routers where that AS peers with others.

As shown in Figure 5, for a node A to reach node B, packets would traverse the last-mile, reach the core, traverse inter-AS links to the second core, and then traverse the last-mile. However, a packet from A to C will traverse a shorter path that skips some of the second component of the height. This can occur when two nodes are part of the same AS and are "close-by" in the network.

Using BGP routing tables, we are able to determine if two nodes belong to the same AS, and using geolocation we are able to deter-

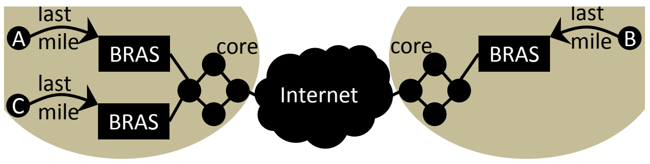

Figure 5. Simple model of two ASes with broadband machines on the Internet

mine their physical distance. Given that it is difficult to break a height down into its two components based solely on end-to-end RTT measurements, we rely on a heuristic: we ignore a portion of the sum of heights when predicting RTT or updating coordinates for such nodes. We have empirically determined that ignoring 20% of the heights for nodes in the same AS and under 225 miles apart produces the best results.

2.4 Symmetric updates

As we have observed earlier, RTT measurement between game machines requires multiple round trips and typically is coupled with bandwidth measurements. Therefore, it does not add substantially to traffic to send one additional message, at the conclusion of the measurement, to notify the other machine of the RTT. Since the RTT is the same in both directions, the recipient can use the reported RTT to update its own coordinates, saving it the time of performing its own RTT measurement. Note that the overhead is even less when the matchmaking service is centralized. In that case, the measuring machine must expend network traffic to notify the central service of the RTT anyway, at which point the service can update the coordinates of both endpoints. For these reasons, Htrae adjusts both machines' coordinates after an RTT measurement. It adjusts them effectively simultaneously, i.e., it uses the pre-update coordinates for one in the computation of the other.

3. IMPLEMENTATION

Our Htrae implementation is based on the publicly-available code for Pyxida, a practical implementation of the Vivaldi algorithm for the Azureus BitTorrent client [13]. We ported this code to C# and added our algorithmic modifications as well as support for experimentation on home machines. Our system is approximately 3,600 lines of code, about a third of which is the direct port of Pyxida.

To enable geographic bootstrapping, Htrae relies on a location service. For simplicity, we built a centralized server that loads the GeoIP City Database from MaxMind [20] in memory and responds to IP address lookups with the corresponding latitude and longitude. The 01 January 2009 version of the database that we use has 4,100,436 entries, each with an IP address range, latitude, and longitude. §5 discusses how well this database covers the IP addresses in our traces and deployment.

To enable AS correction, Htrae uses a routing table service. For simplicity, we built a centralized server that loads a routing table into a PATRICIA trie [22] in memory and responds to IP address lookups with the origin AS. The routing table we use is from the Route Views Project [27], in particular from the route-views.oregon-ix.net router which peers with 43 different routers across 37 ASes on the Internet. We use a snapshot from 04:00 on 01 January 2009, with entries for 277,383 prefixes.

Home networks typically use network address translation (NAT), preventing straightforward direct communication between them [11]. To enable experiments in which home machines communicate with each other, we implemented a simple approach similar to STUN [26].

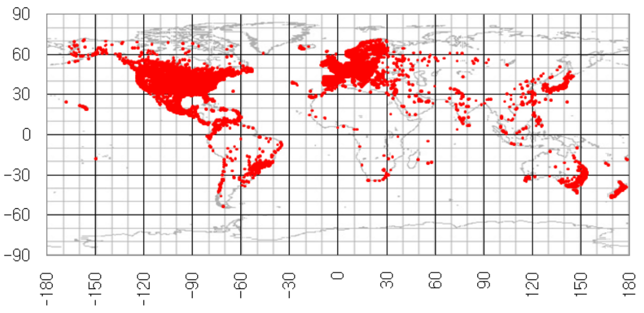

Figure 6. Geographic spread of nodes in trace A

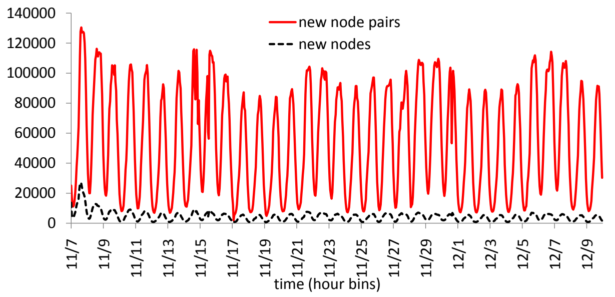

Figure 7. Arrival rate of nodes and node pairs in trace A

4. METHODOLOGY

In this section, we describe our methodology for the experiments in §5. We describe our data set, how we use it to evaluate Htrae, and how we use it to compare to other systems. We also describe how we deployed Htrae to conduct tests in a real-world environment.

4.1 Traces

To evaluate the techniques behind Htrae in the context of game matchmaking, we use traces of Xbox 360 consoles participating in matchmaking for the popular game Halo 3. For Halo 3, matchmaking involves choosing one console to be a server, then choosing up to 16 consoles to be clients. When a player starts matchmaking, his console chooses whether to be a client or server and notifies the Xbox LIVE matchmaking service. Xbox LIVE replies with a set of potential servers, and the client probes them all. Each probe measures RTT by taking the median value of multiple ping-like measurements. Based on the probe results, the client chooses a server.

For each *session*, i.e., instance of one client probing one or more servers to find a game, we log: UTC time, client's and servers' public IP addresses, and RTT from the client to each server. Due to the enormous volume of online game play, the logging system cannot record every probe, so for each session it chooses randomly with probability 10% whether to log all measurements in that session.

In this paper, we focus on the two time periods described in Table 1. Our results from several other time periods are similar and we focus on these two for conciseness. With almost 50 million RTT measurements across over 3.5 million unique IPs in trace A, our evaluation data set is still among the largest ever used.

Figure 6 shows the geographic locations of the nodes in trace A. While there is extensive coverage across North America and Europe, several other major regions such as Japan and eastern Australia are also well represented. We believe such geographic spread is common for large distributed systems.

Two important characteristics of our traces are the rate at which new nodes are seen, and the rate at which any node probes another it has not probed before. Figure 7 shows that after the first three days

trace	start time	training end	end time	distinct IPs	session count	total probes	post-training probes
A	11/07/2008 12am	11/10/2008 12am	12/10/2008 12am	3,534,120	15,630,101	49,946,991	44,227,511
B	01/14/2009 12am	01/17/2009 12am	01/24/2009 12am	1,700,547	5,859,214	20,300,141	14,810,694

Table 1. Halo 3 matchmaking traces

trace	num. of sessions with choice of			
	1 server	2 servers	3 servers	> 3 servers
A	9,351,950	2,096,555	1,138,157	3,043,439
B	3,888,694	620,834	344,420	1,005,266

Table 2. Server counts per session

of trace A, there is an almost constant diurnal arrival of new nodes through the end of the month. The high rate of new probe pairs indicates the need for efficient latency estimation between nodes that have not previously directly communicated.

4.2 Trace replay

To evaluate a latency predictor, we replay each session from a trace in timestamp order. Nodes are initialized using geographic bootstrapping at the time that they first appear in the trace. Subsequently, if a node is absent and then later returns, it rejoins using the coordinates it had the last time it was in the trace. Thus our evaluation reflects the churn that is present in online gaming.

During the training period, we simply feed each probe's source IP address, destination IP address, and RTT to the predictor. After training ends, evaluation begins. For each session, for each of its probes, we ask the predictor to predict the RTT from the source IP address to the destination IP address. We evaluate these predictions, then feed the predictor the session's RTT measurements.

We use two main metrics to quantify the quality of a prediction. The first, *prediction error*, is the absolute difference between the actual median RTT and the prediction. This is relevant to applications, like games, that care about absolute RTT magnitude, e.g., to decide if a pairing meets some QoS requirement. The second, *best-server error*, reflects the need to choose among a set of others the one with the lowest latency, as is the approach of current matchmaking systems. It is computed on a per-session basis, as the additional RTT a client would experience if it sought the lowest-RTT server based on the predictor's output instead of a perfect oracle. In best-server experiments, we ignore sessions with only one probe; Table 2 shows that this still leaves a large number of sessions for evaluation.

Given the enormous size of trace A, we did not have time to evaluate every aspect of Htrae on it. We instead focus on trace B for most of our evaluation, and use trace A for overall results and evaluations of long-term properties such as convergence and drift.

4.3 Comparison to other systems

We compare Htrae to various other latency prediction systems, including Pyxida [13], Hyperbolic Vivaldi [17], iPlane [18], and OASIS [10]. Pyxida is the implementation of the Vivaldi algorithm, with improvements, in the Azureus BitTorrent client. We turned off the recent neighbor set after finding it unhelpful and after discussions with an author indicating it was chiefly meant to deal with an artifact of the Azureus deployment. Hyperbolic Vivaldi is an adaptation of Pyxida that uses hyperbolic coordinates, as described by Lumezanu *et al.* [17]. iPlane uses multiple vantage points on the Internet to create an *atlas* with information about every link. It uses the atlas to predict the path and its RTT between a given pair of nodes. OASIS's main purpose is to select, for a given client, the lowest-RTT server providing a given service. It uses infrastructure nodes to periodically probe address prefixes and learn their geographic locations, which it then uses to estimate RTTs.

For Pyxida and Hyperbolic Vivaldi, the code and algorithms are published, so we are able to evaluate them with trace replay. However, iPlane and OASIS are online services, so we use a different methodology to compare to them. Since both are designed to reflect the current state of the Internet, we do not use historical data. We use the last day of trace B, and query the systems on January 25, 2009, shortly after that trace was captured. To avoid overwhelming the services, we randomly sample 0.5% of the sessions from the last day and use only that subsample for evaluation. This subsample contains 10,869 probes in 2,648 sessions.

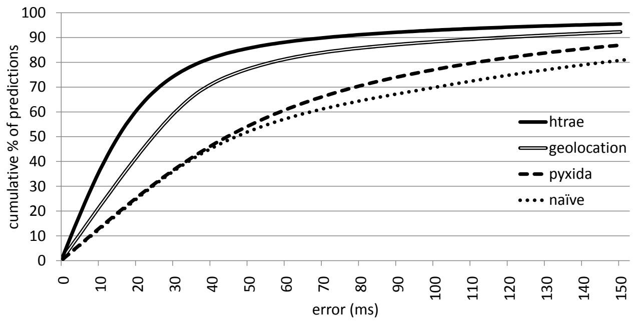

Figure 8. CDF of prediction error for Htrae, Geolocation, Pyxida, and Naive on trace A. Horizontal axis limited to 150 ms.

For illustration, we sometimes also compare to *Oracle* and *Naive*. Oracle always predicts perfectly, i.e., it always returns the actual RTT. Naive always guesses 85 ms, the average seen in Figure 1.

4.4 Deployment

To demonstrate the effectiveness of our Htrae implementation, we deployed it on several friends' home computers running Windows at 23:00 PST on October 7, 2008. This deployment used an older version of Htrae with a 3-dimensional virtual coordinate system rather than the spherical coordinate system we later found to be more accurate. We also used earlier versions of the MaxMind database and Route Views BGP table. Eleven people participated with locations in CA, IL, MA, NY, WA, and the U.K. We ran several experiments serially, with each lasting approximately 30 minutes. Everyone's state reset at the start of each experiment. In each experiment, every three seconds, each node calculated its RTT to another and compared the result to what would have been predicted. It then incorporated the RTT measurement into its predictor. For measurement stability, it calculated RTT as the minimum from ten probes sent 100 ms apart. The first five minutes of each experiment are considered training; we report results for only the remaining time.

5. EVALUATION

We begin our evaluation by comparing Htrae to other latency prediction systems. We then examine in detail the impact of some of Htrae's components, followed by analyses of convergence, drift, and a hybrid of probing and prediction. Finally, we summarize results from our modest deployment.

5.1 Htrae

Figure 8 shows the prediction error of Htrae on the month-long trace A, comparing it to Pyxida, Geolocation, and Naive. We see that Htrae substantially outperforms all other three. For 50% of predictions, Htrae is off by under 15 ms, compared to 24 ms, 44 ms, and 47 ms for Geolocation, Pyxida, and Naive respectively. At the 95th percentile, Htrae is at 138 ms, while Geolocation is at 208 ms, Pyxida at 244 ms, and Naive at 285 ms. Figure 9 shows the results from the week-long trace B, and as expected, they are similar.

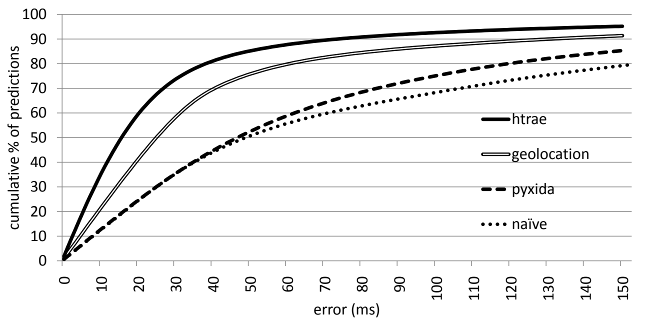

Figure 9. CDF of prediction error for Htrae, Geolocation, Pyxida, and Naive on trace B

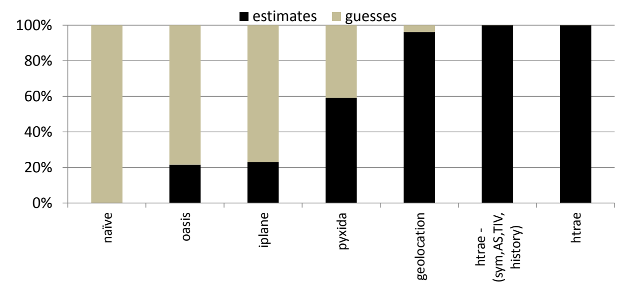

Figure 10. Estimates versus guesses in trace B. To avoid overloading the iPlane and OASIS services, a random subset of the trace was considered.

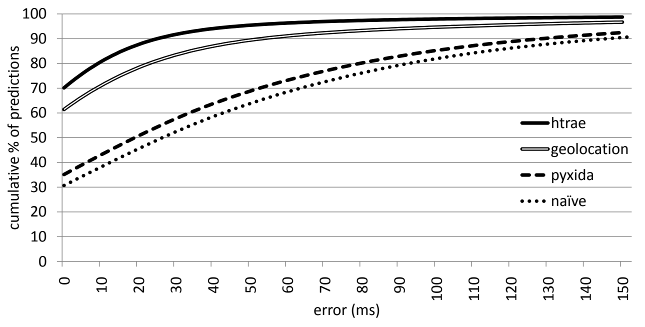

Figure 11. CDF of best-server error for Htrae, Geolocation, Pyxida and Naive in trace A

We also considered how often the errors were underestimates or overestimates, but for space reasons we only present the following summary of the results. For all predictors, the significant error is due to underestimation rather than overestimation: none of the predictors overestimates by more than 85 ms more than 1% of the time. Also, Htrae and Geolocation overestimate about as often as they underestimate, while Pyxida and Naive usually underestimate: 80% and 76% of the time, respectively.

One explanation for a predictor's good or bad performance is the frequency with which it has no knowledge or 100% uncertainty about one or both endpoints. In these cases, its best option is to simply guess the average of 85 ms. Figure 10 shows how often each predictor must guess. Pyxida guesses for 41% of the 14,810,694 post-training probes in trace B. In contrast, Geolocation guesses only 4% of the time while Htrae guesses less than 1% of the time.

In Figures 11 and 12, we show the best-server results for traces A and B. Htrae picks the best server over 70% of the time, while Geolocation, Pyxida, and Naive do so only 61%, 35%, and 31% of

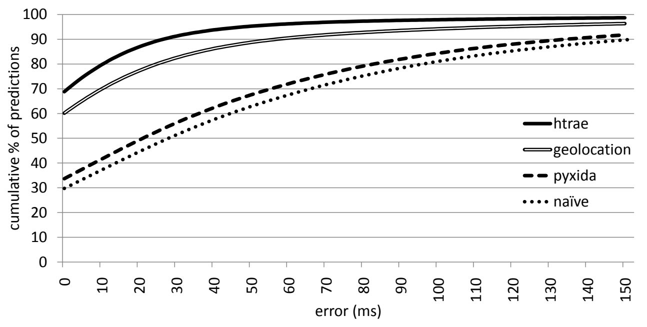

Figure 12. CDF of best-server error for Htrae, Geolocation, Pyxida, and Naive in trace B

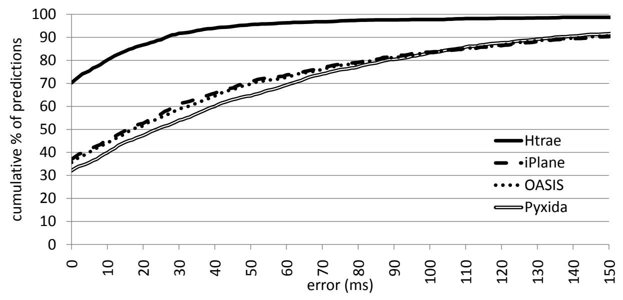

Figure 13. CDF of best-server error for various systems, where a system lacking coverage of a node pair is forced to guess.

the time, respectively. The rest of the time, when Htrae does not pick the best server, it picks one that is still very close: the 95th percentile of additional latency beyond optimal is only 46 ms. Geolocation's 95th percentile is much higher, at 107 ms, and Pyxida and Naive are at 183 ms and 204 ms. Performance at the 95th percentile is important since players often judge games based on the most memorably bad experience they have, or have heard about. A penalty of up to 107 ms due to poor estimation is likely unacceptable for many online games. We now compare Htrae's performance to two online latency prediction systems, OASIS and iPlane, which we query directly.

First, we find that the coverage of Halo 3 players by these services is quite low. OASIS produces estimates for only 22% of our requests, while iPlane does 23%. For the remaining requests to iPlane or OASIS, we use a guess, which is the median latency it produced for all other probes: 9 ms for OASIS and 81 ms for iPlane. In Figure 13, we see that these systems do only slightly better than Pyxida, but much worse than Htrae. The main reason OASIS and iPlane do poorly here is their lack of information for most pairs.

To go beyond the coverage issue, we now consider a different approach. When comparing to OASIS, we discard any probe where OASIS has no prediction, and similarly for iPlane. This gives them a sizable advantage in that each gets to restrict the experiment to the portion of the Internet it has modeled. Figure 14 presents the results of the best-server comparison for OASIS. Despite only considering node pairs when OASIS has a prediction, Htrae still does a better job. OASIS targets services that are in hosted environments unlike home machines, and thus may not need to model last-mile latencies. Our trace that Htrae trains on has end-to-end RTTs, and hence it is able to model end nodes more accurately, which OASIS was not designed to do. Finally, one of OASIS's authors has indicated to us that it uses stale geography information, since active collection stopped a while ago due to various difficulties. The difficulty of keeping such geolocation information up-to-date further

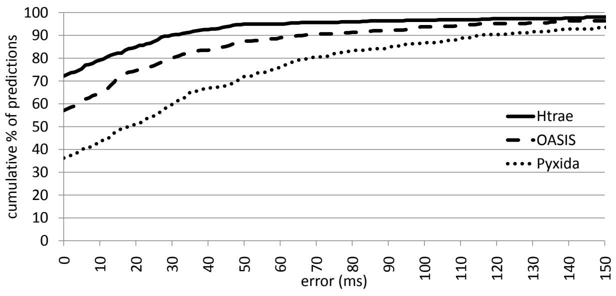

Figure 14. CDF of best-server error for various systems, considering only client/server pairs OASIS makes a prediction for.

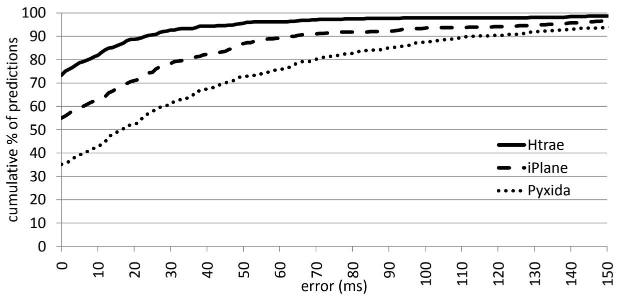

Figure 15. CDF of best-server error for various systems, considering only client/server pairs iPlane makes a prediction for.

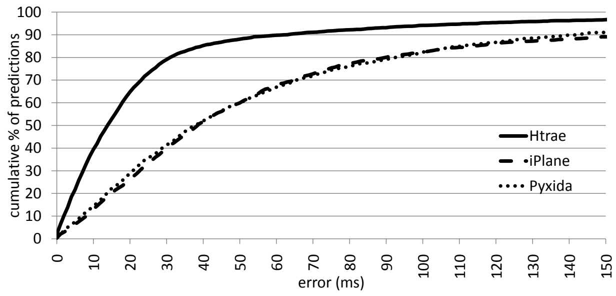

Figure 16. CDF of prediction error for various systems, considering only RTTs iPlane makes a prediction for.

demonstrates the need for the dynamic component of Htrae, which can deal with geolocation inaccuracy.

In Figure 15, Htrae performs better than iPlane in picking the best server, even when restricted to those that iPlane can predict. Figure 16 has the results for a prediction-error comparison to iPlane, and Htrae does better here as well. We note that iPlane probes only one representative out of a cluster of nodes, while our traces have end-to-end measurements. This additional fidelity in Htrae's training likely accounts for some of its advantage.

5.2 Components of Htrae

To understand where the improvements in Htrae come from, we consider Figure 17, where we show how prediction error changes as various components of Htrae are removed. Comparing Htrae to Htrae without symmetric updates, we see that symmetric updates provide a 3 ms prediction advantage at the 50th percentile that grows to 37 ms at the 95th percentile. Considering the effect of removing TIV avoidance, AS corrections, and history we find that those three elements combined provide a small additional improvement. This is as expected since TIVs, intra-AS probes, and repeat probing are all relatively rare; when we look at only intra-AS probes (not shown for space reasons), AS correction improves latency prediction by several milliseconds. Finally, removing geographic bootstrapping to yield the basic Pyxida algorithm, we see a large drop, showing that the majority of Htrae's performance advantage comes from this bootstrapping. In particular, it is clearly not due to the use of spherical coordinates, since we find Vivaldi does badly when modified to use them. We conclude that spherical coordinates are only helpful due to the use of geographic bootstrapping. Additionally, Vivaldi with hyperbolic coordinates performs only marginally better than Vivaldi with spherical coordinates.

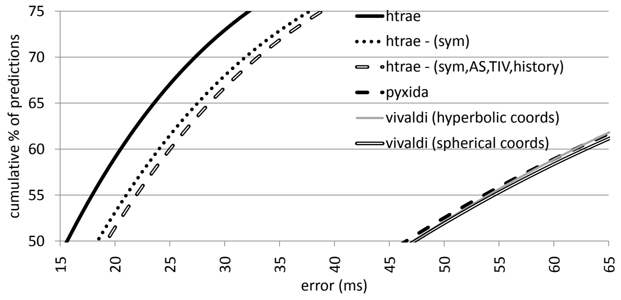

Figure 17. CDF of prediction error on trace B for Htrae; Htrae without symmetric updates; Htrae without symmetric updates, AS corrections, TIV avoidance, or history; Pyxida; Vivaldi with hyperbolic coordinates; and Vivaldi with spherical coordinates. Note the zoomed-in axes.

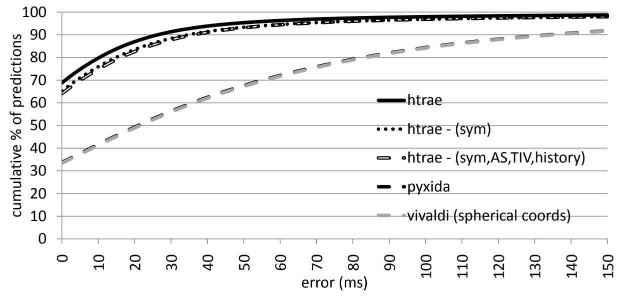

Figure 18. CDF of best-server error for Htrae; Htrae without symmetric updates; Htrae without symmetric updates, AS corrections, TIV avoidance, or history; Pyxida; and Vivaldi with spherical coordinates. The latter two lines overlap.

Figure 18 shows the error for best-server prediction. We again see that geographic bootstrapping provides a major advantage, followed by symmetric updates.

In an earlier trace from March 2008, we found that history prioritization provided near-perfect prediction for 10% of predictions, producing substantial improvement in both prediction error and best-server error. However, in trace B, the occurrence of repeated probes between pairs of nodes is extremely rare. We suspect this is due to changes in player behavior as the game has matured.

5.3 Convergence

An important issue for a latency prediction system is how long it takes to *converge*, i.e., reach its optimal operation point. An NCS, in particular, can take a long time to converge as it adapts from its initial positions to positions that reflect latency observations. Note that in some contexts, NCS convergence indicates how long it takes to reach an embedding with zero error, but since TIVs make a per-

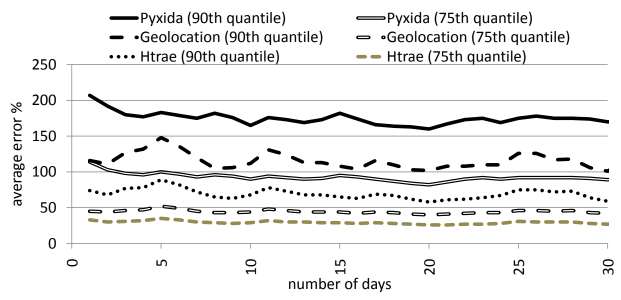

Figure 19. 75th/90th quantile of prediction error seen within a day as a function of number of days running.

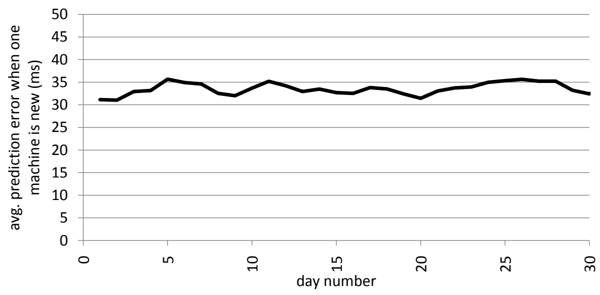

Figure 21. Average prediction error per day for Htrae when one of the involved machines is a new arrival.

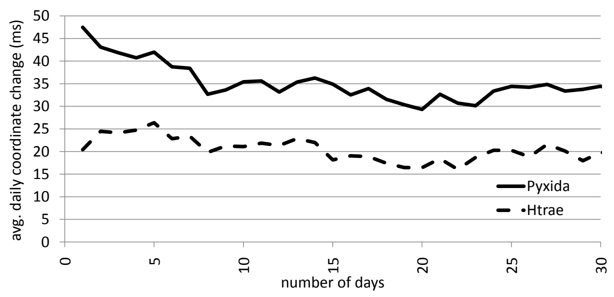

Figure 20. Average per-machine coordinate movement, among those whose coordinates changed, as a function of number of days running.

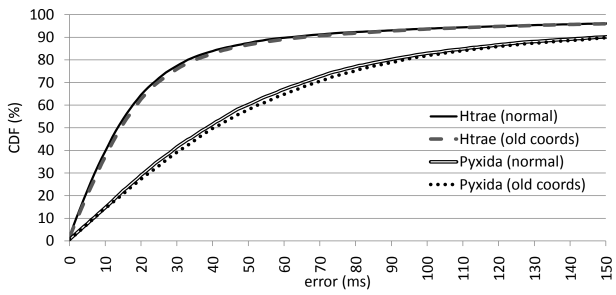

Figure 22. CDF of prediction error using either up-to-date or three-week-old coordinates.

fect embedding impossible this target is only possible in artificially constructed networks. In this subsection, we compare the convergence of Htrae on our game-machine trace to that of Pyxida.

One way to measure convergence is as the time when the error rate reaches its steady state [6]. Thus, we measure, on each day, the 75th and 90th quantiles among all prediction errors occurring that day. Note that all our evaluations include realistic churn as seen in the trace. We turn off history prioritization in Htrae since we are interested in the convergence of the coordinate system. Figure 19 shows the results. Note that there is some variation by day just because of the nature of the data set, which one can see from the variation in stateless geolocation. Pyxida takes about 2–3 days to converge to its steady-state error levels, while Htrae converges essentially immediately. Since Htrae, unlike Pyxida, initializes virtual coordinates using reasonable albeit incomplete information, we get much faster convergence across the full set of machines.

Another way to measure convergence is as the time when the rate of coordinate shifting reaches its steady state. Thus, now we measure, on each day, the average per-machine coordinate movement among machines whose coordinates change that day. Note that in measuring the distance between two coordinates, we use the absolute difference in their height rather than the sum of their heights. Figure 20 shows the results. We observe that Pyxida takes about 7 days to converge to its steady-state coordinate movement levels, while Htrae converges essentially immediately.

5.4 Drift

A potential downside of using an NCS is *drift*, i.e., the tendency for coordinates to move systematically in some direction over time [13]. Drift is a problem because if a machine does not participate for an extended period of time, then when it resumes its coordinates will be in the wrong position relative to the remaining population, which has drifted. Drift is also problematic for Htrae because if coordinates drift from true Earth, e.g., by rotating 10 degrees westward in longitude, then newly arriving machines that obtain initial coordinates based on their location on Earth will be in the wrong position with respect to other machines' coordinates. Pyxida uses *gravity*, a weak force that pulls all coordinates gently and slowly toward the origin, to counter drift. However, this only works on translational drift; it cannot help with rotational drift, which is the only kind that can occur in Htrae's spherical coordinate system. Fortunately, as we will now show, drift does not pose a significant problem for Htrae.

The standard way to measure drift is to measure the movement of the centroid of the population. However, this would only reflect translational drift, not rotational drift, and it is the latter we are most concerned about since our coordinates are spherical coordinates. Hence, rather than measure drift directly, we instead measure the *effect* of drift. We measure the extent to which newly arriving nodes see greater error as time progresses, and the extent to which machines that are absent for an extended period suffer from returning to a shifted coordinate system. Again, we now turn off history prioritization in Htrae since we are interested in coordinate drift.

First, we measure, for each day of a one-month run, the average error seen by newly arriving machines on that day. Figure 21 shows the result. We see that there is no noticeable trend upward, suggesting that this effect of drift is not present.

Next, we simulate the effect of absence. On the last day of a one-month trace, we compare the effect of using the latest coordinates for the probing machine to using its coordinates from three weeks earlier. This simulates what would happen if the machine had been absent for those three weeks and needed to use its coordinates from just before its absence. We only consider machines whose uncertainty has changed by less than 10% during those three weeks, since we are only interested in machines whose coordinates have converged before their departure. Figure 22 shows the results. We see that the effect of using old coordinates is small, generally producing less than 2 ms of additional error. Indeed, it is roughly

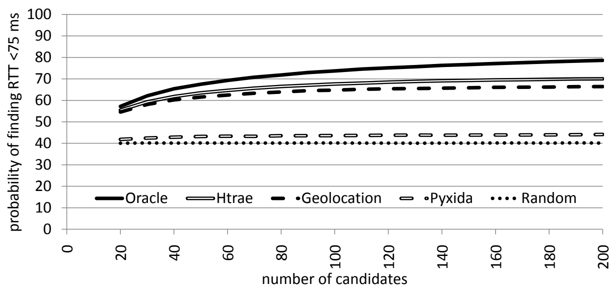

Figure 23. Probability of finding a server with RTT under 75 ms, as a function of number of candidates. Each prediction algorithm predicts which six of those candidates have the lowest RTT, and probes only them. "Oracle" always finds the candidate with lowest RTT, and "random" chooses an arbitrary six to probe.

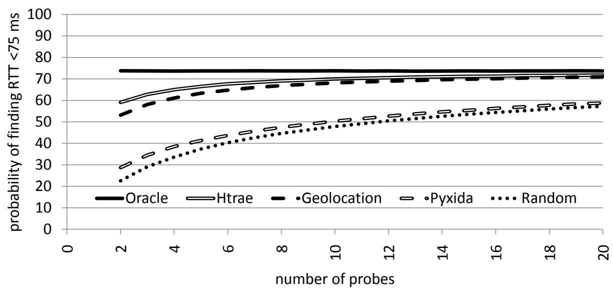

Figure 24. Probability of finding a server with RTT under 75 ms, as a function of number of probes. Each prediction algorithm predicts which of the 100 candidates have the lowest RTT, and probes only them. "Oracle" always finds the candidate with lowest RTT, and "random" chooses an arbitrary set to probe.

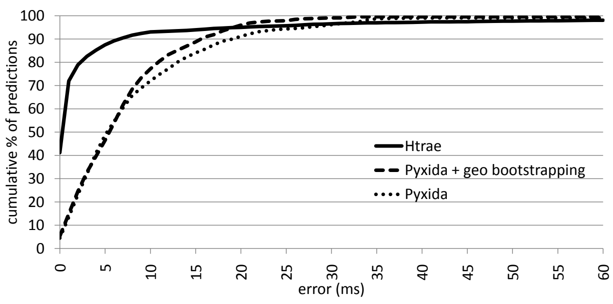

Figure 25. CDF of prediction error in various experiments during Htrae deployment.

the same amount seen in Pyxida, which uses gravity to counter drift. Thus we believe that drift is not a problem for Htrae. We suspect that the reason old coordinates work slightly worse is due to other factors such as Internet topology changes and/or improvement to coordinates for other nodes over the course of three weeks.

In conclusion, it appears that drift does not pose a significant problem for Htrae. We believe this is because the steady arrival of new machines, each of which is initialized with reasonable confidence in its correct location on true Earth, prevents the coordinate system from drifting away from true Earth. Since churn is common in other peer-to-peer applications besides online games, we expect drift will not be a problem in other applications as well.

5.5 Limited probing

So far, our analyses have assumed that matchmaking relies solely on prediction for latency estimation, i.e., that there is no time to perform network probing to determine latency. We are primarily motivated by that scenario for game matchmaking where users are very averse to waiting, or for games where it is prohibitive to probe all potential traffic paths because they have an all-to-all rather than client/server communication pattern. Nonetheless, there may be other application scenarios where some network probing can be used to supplement latency prediction.

We now evaluate the performance of latency prediction systems when given n servers and asked to reduce it to a subset of m servers. In this scenario, the application will then probe the m servers, and pick one with an acceptable latency. A recent study showed that clients prefer games better as RTT goes down to 75 ms, but below this they do not care [2]. Hence we evaluate how often a latency prediction system selects a subset containing a server with RTT under 75 ms. Note that in such an evaluation, latency prediction systems are allowed a much larger tolerance of error, not only because they need not place the lowest-latency server among the m, but also because they can make up to $m-1$ "mistakes".

During our trace, many clients probed as many as 200 servers, so we know the actual RTT to all of them. We randomly select n of the servers a client probed and query the latency prediction system for all n. We then choose the m that were predicted to have the lowest RTT, and determine if any one of these has a latency under 75 ms.

In Figure 23, we present the $m = 6$ scenario while varying n on the horizontal axis. We pick 6 because that is the average number of probes during a multi-server session in our traces. We see that Htrae performs the closest to Oracle, and significantly above Pyxida. In this scenario, geolocation also performs extremely well because the number of nodes for which it has no or inaccurate location information is relatively small. Since it gets six tries to pick servers within 75 ms, these errors have little impact. However, it is quite possible for nodes with poor location information to get starved and never be picked by geolocation. Furthermore, we found that some clients experience consistently bad performance with geolocation, due presumably to inaccurate placement.

In Figure 24, we fix n at 100 and vary m. This shows how the benefit of latency prediction varies as more probing is done. We see that Htrae once again does best of all the non-oracle prediction systems. However, as expected, the benefit of Htrae, and latency prediction in general, declines as the number of probes allowed is increased. This is natural, because the point of latency prediction is to avoid probing to reduce user wait time and network overhead.

We conclude that Htrae is most useful in scenarios where there is time to probe only a modest fraction of the candidate game traffic paths available. This includes scenarios where the matchmaking service wants to make a quick decision without relying on any probes, where probes are costly in terms of game delay, or where game traffic will traverse so many paths that there is not time to probe them all. However, even when there is time to probe a large number of candidates, using Htrae or geolocation is still far preferable to using random selection as is commonly done today.

5.6 Deployment

We now present results from our live deployment of Htrae on the home machines of eleven volunteers. Because of the limited size of this deployment, and hence lack of multiple nodes within the same small AS, we do not expect AS correction to be beneficial and in fact it is not. Figure 25 presents results of our experiments.

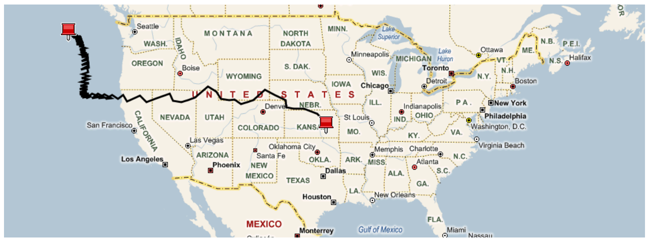

Figure 26. Virtual path taken by the coordinates of a machine in Redmond, WA. Its IP address was initially incorrectly classified as being in the center of the U.S.

We find that even in this small deployment, geographic bootstrapping is quite helpful, reducing Pyxida's average error from 11 ms to 8 ms. The frequent all-to-all nature of probing ensures Pyxida is given ample time to converge, but we see that it converges to a local minimum worse than what we obtain with better initial conditions. Finally, the remaining improvements reduce average error to 6 ms; chiefly, this comes from history prioritization, which unsurprisingly is extremely useful in a deployment enabling all-to-all probing.

An interesting finding in our deployment is an illustration of the dynamic nature of geographic bootstrapping. One of our volunteers' machines was in Redmond, WA, but our database could not pinpoint its IP address in more detail than just its country. Figure 26 shows how over the course of the experiment, this machine adjusted its coordinates from the initial position in the center of the U.S. to a location much closer to where it actually is. Estimation systems relying solely on geographic information would not be able to adapt in this way to missing information, but this example illustrates that with Htrae it happens naturally.

6. DISCUSSION AND FUTURE WORK

There is substantial room for improvement in the way game matchmaking is done today [3]. Adding a latency prediction system to select good candidates for probing offers substantial improvement in the latency of the ultimate choice. In addition, incorporating latency prediction will allow the design of a much different type of matchmaking system, one with greater flexibility to explore a broader range of alternatives. For instance, in the case of matchmaking for a peer-to-peer game, the service will be able to select groups of players with mutually low latency, without requiring all candidates to probe all others.

Furthermore, it is clear that a network coordinate system by itself is not sufficient as a latency prediction system for games; geolocation is an important tool in making predictions accurate. For some applications, such as static content distribution, geolocation by itself may be sufficient. However, for online gaming, it can be costly to make certain customers unendingly suffer simply because of inaccurate geolocation. Htrae has the advantage of automatically correcting for such errors and adapting to changing network conditions. It also produces better worst-case outcomes.

We have evaluated Htrae over month-long traces, and certainly over such a long period there would have been many routing changes on the Internet. However, we have not examined how Htrae adapts to major changes, such as a trans-oceanic link being accidentally cut, or two major ISPs de-peering. It would be interesting to evaluate how prediction error increases during this time and how quickly Htrae converges to a stable representation of the new network.

Finally, there are likely other applications for a latency prediction system as accurate and scalable as Htrae, particularly where participants are likely to be numerous home machines. For instance, distributed hash tables need to select close machines for routing table entries and lookups [7]. Also, file sharing systems often find it desirable, for the sake of ISP friendliness and performance, to have peers select close peers for content exchange [4]. Some voice-over-IP applications use intermediate peer relays to handle firewall traversal, and the choice of peer can have a tremendous impact on call quality. Especially in this application, users will expect fast call-setup times and may not tolerate long delays from probing. We would like to examine how well Htrae works for these scenarios.

7. RELATED WORK

Our work is closely related to three general approaches to latency prediction: graph representation, NCSes, and geolocation.

7.1 Graph representation

All-to-all probing, as in RON [1], produces highly accurate latency prediction but will not scale to large systems, such as one with millions of game players. One way that researchers have found to achieve scalability is to use *graph representation*. In this approach, a cluster of nodes having similar latency properties is represented by a vertex in a graph, and edges within that graph represent links between clusters. The latency between two clusters is the total latency of the links along the path joining them. IDMaps [9] achieves such a graph embedding through the use of a select set of nodes they called *tracers*. IDMaps periodically measures the distance between each pair of tracers, and between each cluster and its closest tracer. Theilmann and Rothermel [31] proposed an extension of this approach, in which the tracers themselves are clustered so they do not have to perform all-to-all probing. The tracers form a tree hierarchy, allowing for a graph having more tracers.

iPlane [18] seeks to learn the *actual* graph structure of the Internet. In this graph, which it calls an *atlas*, the nodes are actual routers and endpoints and the edges are actual network links. iPlane uses multiple vantage points on the Internet and a variety of techniques to create its atlas. This atlas includes latency, loss, and bandwidth information about each link, enabling it to predict properties of arbitrary paths. iPlane Nano [19], a recent variant, uses sophisticated atlas compression to allow decentralized latency prediction even though atlas creation remains centralized.

It is unclear whether graph representation systems, which involve a great deal of centralized calculation on an enormous amount of data, will scale to game matchmaking systems with millions of participants. Our evaluation relative to iPlane suggests it may be difficult for a graph representation system to achieve the coverage necessary to deal with such a large population.

7.2 Network coordinate systems

To achieve even further scalability, many have proposed using a network coordinate system, which embeds the graph structure of the Internet into a virtual coordinate space. This technique of embedding a graph into a metric space has been used in other disciplines to reduce the size of the representation [32, 35, 37]. In the specific application of latency estimation, this reduction in representation size has particular advantages: it reduces the amount of background probing necessary to keep the data up to date, and it enables more decentralized implementations. It is thus well-suited for game matchmaking, and in turn for Htrae, since even when there is a centralized matchmaking service it must dramatically limit its per-machine state due to the large number of participants.

The idea of an NCS was first proposed for use in GNP [23]. This system makes use of a fixed set of nodes called *landmarks*. Once the system obtains the RTT between each pair of landmarks, the system computes the virtual coordinates for each landmark in a way that minimizes the sum of the squared error in RTT estimation over all landmark pairs. For a non-landmark node to compute its virtual coordinates, it finds its RTT to each landmark, and chooses virtual coordinates for itself that minimize the sum of the squared error in RTT estimation over all paths from itself to the landmark.

One downside of this approach is that the landmarks become bottlenecks. To allay this issue, Lighthouse [25] uses multiple sets of landmarks, each with its own distinct coordinate space. Lighthouse unifies these spaces into a global space so that the distance between coordinates in distinct spaces can be calculated. PIC [5] expands the set of landmarks further by allowing any node, once it has computed its coordinates, to become a landmark for new nodes. Another approach is to let nodes calculate coordinates using measurements to only a subset of the landmarks, as in ICS [15] or with virtual landmarks [30]. However, provisioning even a few landmarks to deal with millions of customers is an expensive proposition for a game matchmaking system.

Fortunately, some researchers have discovered that landmarks are not necessary. There exist graph embedding techniques that rely only on distance measurements between nodes, enabling nodes to learn reasonable coordinates in a global space by progressive refinement based on latency measurements. These techniques include spring embedding [8], used by Vivaldi [6], and force-field explosion simulation, used by Big-Bang Simulation [28]. Such embedding techniques are thus well suited to the application of game matchmaking, which is why we use one in Htrae.

Researchers have noted problems with NCSes, including the sensitivity of coordinate embeddings to initial conditions and the difficulty of embedding triangle-inequality violations [12]. Suggested fixes have included techniques for avoiding local minima and otherwise reducing the effect of initial conditions [28], rather than choosing initial conditions based on geography as we do. Wang *et al.*proposed solutions to the problem triangle-inequality violations pose [33]; we have adapted one of them for use in Htrae and evaluated its effectiveness in a realistic workload.

We are not the first to attempt using an Earth-like coordinate system, but we are the first to have done so successfully. In the paper describing GNP [23], the authors indicate they found a spherical coordinate system inferior to a Euclidean one. Vivaldi's creators found a similar result [6]. However, we got it to work by using a new approach not tried before: using geolocation for *all* machines' initial positions, not just for a small number of landmarks.

7.3 Geolocation

Htrae relies upon earlier work that has enabled and refined the estimation of the latitude and longitude of a machine. NetGeo [21] extracts textual location names from `whois` records and looks them up in a geographic database. IP2Geo [24] extracts location names from the DNS name of the node or nearby nodes. OASIS [10] uses Meridian [34] to find which infrastructure node with a known geographic location is closest to an address prefix, and assigns that geographic location to that prefix. Finally, some commercial databases [20] complement these techniques with others such as mining websites that ask users for their location.

7.4 Evaluation

One of our contributions is a thorough evaluation of latency prediction on a large distributed system. Generally, latency prediction system evaluations have used data sets such as PlanetLab ping times and King measurements, which are not representative of home machines and do not illustrate realistic churn. A notable exception is that of Ledlie *et al.* [13], who deployed the Vivaldi algorithm in the Azureus BitTorrent client, revealing much about the sources of error in a large P2P system containing mostly home machines. Pyxida, the resulting NCS implementation, incorporates solutions to many of these problems, and is generally considered the state-of-the-art in NCSes [13]. Our trace of game players' machines is even larger than Pyxida's corpus, and has enabled us to evaluate in even more detail the effectiveness of various implementation elements of an NCS. It also allowed us to examine properties of latency and churn specific to game-matchmaking scenarios, and to devise a system with significantly better predictive power than Pyxida.

8. CONCLUSION

Htrae is a latency prediction system designed for use in game matchmaking and other latency-sensitive applications that run primarily on home machines. Htrae is simple and scalable, as illustrated by the small size of our implementation and the ease of our deployment. Its primary innovation is that it uses a novel combination of two classic approaches to latency prediction, geolocation and a network coordinate system (NCS), fusing them in a way that mitigates the disadvantages of each. It achieves this fusion by assigning each machine a location on a virtual Earth; each location starts out roughly matching the machine's approximate real-world location, and is then dynamically adjusted to improve the correspondence between distances and latencies.

To guide our design of a system to deal with realistic game matchmaking scenarios, we used observations of 50 million matchmaking probes among 3.5 million game machines. This data has enabled a thorough evaluation of our system, showing how effective its various elements are, how well it solves classic problems in latency prediction, and how well it performs relative to other state-of-the-art latency prediction systems. We found that Htrae significantly out-predicts other latency predictors, including Pyxida, iPlane, and OASIS. With Htrae, 90th percentile prediction error is reduced by a factor of well over 2, as is 90th percentile best-server error. As a consequence, we expect our system to enable more efficient and effective matchmaking for games, as well as for a variety of other applications where users are sensitive to latency.

9. ACKNOWLEDGMENTS

We thank Chris Butcher, Roger Wolfson and Luke Timmins for all their work collecting Halo 3 traces and sharing them with us. We are also grateful to the creators of OASIS and iPlane for their public query services, and in particular to Harsha Madhyastha for providing code to query iPlane. We thank all our volunteers for running the Htrae system on their home machines.

10. References

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[2] 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 *In NetGames*, Aug./Sep. 2004.

[3] Bungie. Halo 3 Forum. http://www.bungie.net/Forums/posts.aspx?postID=8455638.

[4] D. R. Choffnes and F. Bustamante. Taming the Torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems. In *Proc. SIGCOMM Conference*, pages 363–374, Aug. 2008.

[5] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet coordinates for distance estimation. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pages 178–187, Mar. 2004.

[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. SIGCOMM Conference*, pages 426–437, Aug./Sep. 2004.

[7] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, pages 85–98, Mar. 2004.

[8] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[9] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transactions on Networking*, Oct. 2001.

[10] M. J. Freedman, K. Lakshminarayanan, and D. Mazières. OASIS: Anycast for any service. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, pages 129–142, May 2006.

[11] S. Guha and P. Francis. Characterization and measurement of TCP traversal through NATs and firewalls. In *Proc. Internet Measurement Conference (IMC)*, Oct. 2005.

[12] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43, 2000.

[13] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, pages 299–311, Apr. 2007.

[14] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye. Measurement and estimation of network QoS among peer Xbox 360 game players. In *Proc. Passive and Active Measurement Conference (PAM)*, pages 41–50, Apr. 2008.

[15] H. Lim, J. C. Hou, and C.-H. Choi. Constructing Internet coordinate system based on delay measurement. *IEEE Transactions on Networking*, 13(3):513–525, June 2005.

[16] C. Lumezanu, D. Levin, and N. Spring. PeerWise discovery and negotiation of faster paths. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, Nov. 2007.

[17] C. Lumezanu and N. Spring. Measurement manipulation and space selection in network coordinates. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pages 361–368, June 2008.

[18] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *Proc. Symposium on Operating Systems Design and Implementation (OSDI)*, pages 367–380, Nov. 2006.

[19] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path prediction for peer-to-peer applications. In *Proc. Networked Systems Design and Implementation (NSDI)*, Apr. 2009.

[20] MaxMind. Geolocation and online fraud prevention from MaxMind. http://www.maxmind.com/.

[21] D. Moore, R. Periakaruppan, and J. Donohoe. Where in the world is netgeo.caida.org? In *Proc. INET Conference*, July 2000.

[22] D. R. Morrison. PATRICIA—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, Oct. 1968.

[23] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. IEEE Computer and Communications Conference (INFOCOM)*, 2002.

[24] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *Proc. SIGCOMM Conference*, pages 173–185, Aug. 2001.

[25] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proc. International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.

[26] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN – simple traversal of user datagram protocol (UDP) through network address translators (NATs). Network Working Group RFC 3489, Mar. 2003.

[27] Route Views Project. University of Oregon. http://www.routeviews.org/.

[28] Y. Shavitt and T. Tankel. Big-Bang Simulation for embedding network distances in Euclidean space. *IEEE/ACM Transactions on Networking*, 12(6):993–1006, Dec. 2004.

[29] Y. Staff. Video game sales break records. http://us.i1.yimg.com/videogames.yahoo.com/feature/video-game-sales-break-records/1181404, Jan. 2008.

[30] L. Tang and M. Crovella. Virtual landmarks for the Internet. In *Proc. Internet Measurement Conference (IMC)*, pages 143–152, Oct. 2003.

[31] W. Theilmann and K. Rothermel. Dynamic distance maps of the Internet. In *Proc. Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Mar. 2000.

[32] W. S. Togerson. Multidimensional scaling of similarity. *Psychometrika*, 30(4):379–393, Dec. 1965.

[33] G. Wang, B. Zhang, and T. S. E. Ng. Towards network triangle inequality violation aware distributed systems. In *Proc. Internet Measurement Conference (IMC)*, pages 145–157, Oct. 2007.

[34] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proc. SIGCOMM Conference*, pages 85–96, Aug. 2005.

[35] G. Yona, N. Linial, and M. Linial. ProtoMap: automatic classification of protein sequences and hierarchy of protein families. *Nucleic Acids Research*, 28(1):49–55, 2000.

[36] H. Zheng, E. K. Lua, M. Pias, and T. G. Griffin. Internet routing policies and round-trip times. In *Proc. Passive and Active Measurement Conference (PAM)*, Mar./Apr. 2005.

[37] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multi-dimensional scaling. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):198–207, Apr. 2002.