Battling demons and vampires on your lunch break.

Switchboard: A Matchmaking System for Multiplayer Mobile Games

Justin Manweiler⋆ Sharad Agarwal† Ming Zhang† Romit Roy Choudhury⋆ Paramvir Bahl†

†Microsoft Research ⋆Duke University

{sagarwal,mzh,bahl}@microsoft.com {justin.manweiler,romit.rc}@duke.edu

Abstract— Supporting interactive, multiplayer games on mobile phones over cellular networks is a difficult problem. It is particularly relevant now with the explosion of mostly single-player or turn-based games on mobile phones. The challenges stem from the highly variable performance of cellular networks and the need for scalability (not burdening the cellular infrastructure, nor any server resources that a game developer deploys). We have built a service for matchmaking in mobile games – assigning players to games such that game settings are satisfied as well as latency requirements for an enjoyable game. This requires solving two problems. First, the service needs to know the cellular network latency between game players. Second, the service needs to quickly group players into viable game sessions. In this paper, we present the design of our service, results from our experiments on predicting cellular latency, and results from efficiently grouping players into games.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*; K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Design, Measurement, Performance

Keywords

Latency Estimation, Matchmaking, Mobile Gaming

1. INTRODUCTION

Games have become very popular on mobile phones. The iPhone app store has over 300,000 applications as of October 2010, roughly 20% of which are games [26], and yet 80% of application downloads are games [16]. On the Windows Phone 7 platform, the top applications are games, all the way down to number 29 (as of 10 December 2010). Despite this popularity, mobile gaming is still in its infancy. The vast majority of mobile games are either single player, turn-based (latency-insensitive games such as card games or strategy games), or multiplayer only over Bluetooth or Wi-Fi.

We believe that *interactive* multiplayer games, such as FPS (first-person shooter) or racing games, that work over cellular data are just around the corner. While a few are currently available, for them to become numerous and successful, there are many mobile systems challenges to overcome. In a very recent interview [27], John Carmack, co-founder of id Software, said (in the context of FPS games and 3G) : "*multiplayer in some form is where the breakthrough, platform-defining things are going to be in the mobile space*". One urgent challenge is managing the highly variable network performance that applications experience over 3G cellular [21]. Already a difficult problem for multiplayer games on home broadband connections [5], a player with poor network performance can destroy the experience of others in the same networked game.

The key to solving this problem is effective matchmaking. Players should be grouped into games where each player's network performance meets the needs of the game, and the size of the group is as large as possible within the limits of the game's architecture. For matchmaking to be effective, it must solve two problems. First, the network performance between (potentially many) game players needs to be estimated. This estimation should be done quickly so that impatient gamers are not left waiting, and in a scalable way so as not to overburden cellular network links nor expensive server bandwidth. Second, players need to be grouped together into games based on their network performance and desired game characteristics (*e.g.,* game topology or size). This can be difficult if there are many players.

A particularly challenging type of matchmaking is that for P2P games. In such games, the game developer is not burdened with the expensive task of maintaining high-powered and high-bandwidth servers in many locations across the planet. Instead, individual player devices are matchmaked into different game sessions and they exchange game state among themselves. This is a very popular architecture for multiplayer gaming – Xbox LIVE supports game sessions in this way that are measured in the 100s of millions each month [5].

In this work, we address the problem of matchmaking for P2P multiplayer games over cellular data networks. Today, a major US cellular carrier charges an additional $3 per month for a public IP address and unrestricted inbound traffic to a phone. With this option, we have been able to communicate directly between phones over 3G without going through a server on the Internet. We take the controversial stance that soon, most cellular data plans will include this feature by default and there will be many such P2P applications on phones. Even though we address matchmaking for P2P games, our system and contributions are also applicable to the traditional server-based game matchmaking problem.

As far as we know, this is the first paper to address the matchmaking problem for multiplayer mobile games over cellular networks. Specifically, our contributions include:

- We show that not only is phone-to-phone traffic feasible over cellular networks, it reduces latency compared to via an Internet server.
- Despite the difficulty that prior work [7, 21] implies, we show that it is actually possible to estimate or predict the latency that a phone will have. We do so based on the experience of other phones and information about the cellular connection that is available to the phone. Our goal is not to identify all such predictors – that is a moving target with rapidly evolving cellular networks. Rather, our goal is to show that such predictors do exist and can be easily determined automatically without requiring detailed and proprietary information from cellular networks.
- We show how, using such latency estimation, we can significantly reduce the burden on individual phones and cellular networks for effective matchmaking.
- We design and implement Switchboard, a matchmaking system for mobile games that is scalable not only in the measurement overhead but also in grouping players together quickly even when there are tens of thousands of players.

2. MOTIVATION AND PRIOR WORK

2.1 Latency in multiplayer games

Multiplayer gaming has gone past the confines of a single console or set of consoles on the same LAN to working across the Internet. This has come at the cost of additional latency. Studies have established that user behavior and performance in games can change significantly with 50ms-2000ms of additional latency, depending on the type of game [12]. Some games use high-precision objects (*e.g.,* rifles and machine guns) and first-person perspectives which tighten the latency bounds that a player can tolerate. As a result, researchers and game developers have built several techniques for hiding latency in networked games. Not surprisingly, these techniques rely on manipulating how time is perceived or handled by different components of the game.

Some games advance time in lockstep [8] or with event-locking. A player can advance to the next time quantum only when all other players (or all other players that this player is directly interacting with) are also ready to advance. So if any player experiences occasional high network delay, the lockstep protocol will ensure that everyone proceeds at the (slowest) pace so that there is no inconsistency in the distributed game state.

Some games use dead reckoning [9] to predict the future positions of players or objects (such as bullets). So if due to network delay, I do not receive a new position update from a remote player or object, I can use the last few updates to plot a trajectory and speed and guess the current position. If the packet arrives later and the position calculation does not match, the software will have to reconcile inconsistent game state [11], which often appears to the player as a "glitch in the matrix" – an object that suddenly jumps from one spot to another. Some games use simple linear trajectory calculations, while others calculate more complex angular velocities and use human movement models.

These techniques are effective and commonly used for hiding network jitter. That is, if the additional network delay is occasional, the player may not notice the side effects of these techniques. However, if the typical network latency of one (or more) player(s) is high, then the experience for all players suffers because games in lockstep will progress very slowly, or there will be many consistency corrections with dead reckoning.

2.2 Matchmaking in online games

To reduce the impact of players with persistently high latency, many online games use some form of matchmaking to setup each game session. When a player launches a game and selects online gameplay, the game will typically make the player wait in a virtual "matchmaking lobby". While in this lobby, game clients connect to a matchmaking service that maintains a current list of servers with game sessions that are waiting for players to join [15]. At any point in time, there may be many game servers available for hosting a game. Clients will estimate their latency to each of these servers, and join one that they have low latency to.

While there are many online games that have servers on the Internet, there are major costs associated with maintaining these servers. Servers across the planet are needed to provide low latency games to players in different geographic regions. Each such location may need many servers to host the large numbers of game sessions that popular games experience. The traffic consumed by such hosting can be enormous, especially considering that FPS games frequently exchange packets and can last for as long as 7-19 minutes [17].

A popular alternative is to leverage the compute power of large numbers of game consoles and PCs on the Internet. Some P2P games use a *star-hub* topology, where one host player serves as the central point of coordination and all the client players exchange game state updates through the host. Hosts can be selected based on their network connectivity and past success in hosting games. Such games have similar communication patterns as client-server games, except that a game player replaces an Internet server as the hub. Another commonly-used topology is the *clique*, where one player can directly communicate with any other player in the group. It avoids some shortcomings of the star-hub topology, namely the single point of failure and performance bottleneck. However, it is more challenging for the game developer to maintain consistency among players. Microsoft Xbox LIVE is a very popular platform for P2P games and matchmakes games using the star-hub topology – it has over 23 million users [29] and the number of online P2P game sessions for *individual* popular game titles are measured in the 100s of millions per month [5], with roughly 16 players in each game session.

2.3 P2P games over cellular networks

Inspired by the popularity of P2P online games, we believe that low latency, multiplayer games over cellular data networks are better enabled through P2P (peer-to-peer or phone-to-phone, take your pick). Mobile platforms have a large number of games today, written by a variety of developers. These platforms, such as Windows Phone 7, iPhone, Android, are used in many regions of the world. Not all game developers can afford to host servers everywhere, pay for traffic, and manage them.

In addition to the cost benefit, the latency benefit of P2P is significant. In Figures 1 and 2, we show the latency between two phones, either directly or by summing up the individual latencies from each phone to a mutual server. In this fashion, we can compare four strategies – P2P, using a single server, using two servers, and using many geo-distributed servers.

A hosting strategy with modest cost would be to have a single server in one location, for example at the University of Washington. This strategy is 139ms to 148ms worse than P2P at the 50th percentile. A more expensive strategy would be to host servers at both locations in which we conducted experiments and use the closer one, either University of Washington or Duke University – the penalty is 47ms to 148ms. The most expensive strategy is to host servers in many datacenters with direct peering to many ISPs and use third-party DNS redirection services that optimize for la-

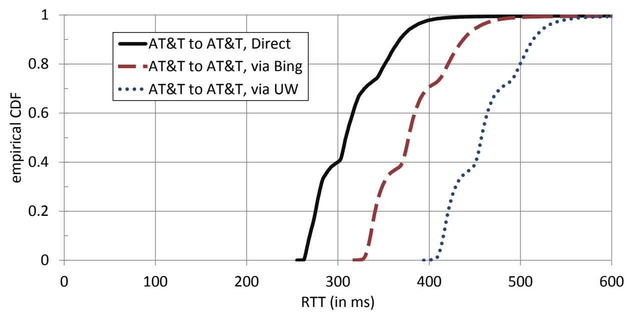

Figure 1: CDF of ping latency between two phones on 3G HSDPA connectivity in Redmond, WA, either direct, or via a nearby University of Washington server, or via the best server offered by geo-distributed Bing Search. Horizontal axis cropped at 600ms.

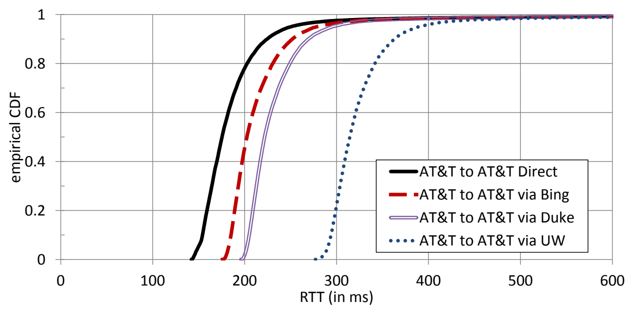

Figure 2: CDF of ping latency between two phones on 3G HSDPA connectivity in Durham, NC, either direct, or via a nearby Duke University server, or via a distant University of Washington server, or via the best server offered by geo-distributed Bing Search. Horizontal axis cropped at 600ms.

tency, such as a large search engine like Bing – this strategy is 27ms to 69ms worse than P2P. Depending on the type of game [12], these latency penalties can deteriorate the game experience. Hence, we believe P2P is an attractive model for mobile games as well.

2.4 Cellular network performance

As is apparent from Figures 1 and 2, phones in different parts of the same mobile network can experience very different latencies. One of the most important aspects of matchmaking is knowing the latency that each player will have to each potential peer. While this has been studied for consoles on the Internet [5], there are several open questions in the mobile context. Should each phone ping all the available peers to estimate this latency? For how long should it ping? How often do these latency estimates need to be updated? How will this scale to a popular game with many players? Predicting future latencies based on past latencies or other information about the network can be used to reduce the overhead of such measurements.

Recent work has characterized the performance of 3G [28], and the performance of TCP flows [14] and applications [24, 21] over 3G. This work has shed light on different applications experiencing different network performance and improvements to TCP throughput. CDNs select low-latency servers by typically geo-locating the client (or LDNS server) IP address. However, recent work [7] on 3G phones shows this will not work in the cellular domain. They note that different cities have different latency distributions, but with the caveat that the measurements were to a single server, and time variation was not factored out. That work motivated us to explore this problem in more depth and understand whether it is even possible to predict cellular network latency.

For P2P communication to work over cellular data networks, phones in a game session have to be able to receive inbound traffic connections. While on some mobile networks in the US this is not currently possible, AT&T Wireless provides a public IP address to a phone and unrestricted inbound traffic for an extra US$3 a month [6]. Sprint offers the same feature for free. We believe that once compelling applications such as fast, multiplayer games become popular, this will be the default behavior.

2.5 Grouping

Once the latencies between players are measured or predicted, the remaining challenge in matchmaking is to group them into viable game sessions. Each session should have only those players that have latencies to each other within the tolerance of the game. This tolerance may be specified, for instance, as a 90^{th} percentile latency that must be below a certain threshold. Even though 10% of the time higher latencies may be experienced, those might be corrected with software techniques such as dead reckoning. Each session should be packed with as many viable players as the game allows (just a single player in each session is an easy solution but rather boring for the player) [1].

Ideally, a single matchmaking system should accommodate different types of P2P topologies that game developers may use, such as clique and star-hub. Creating such groups under latency constraints while maximizing group sizes is related to the *minimum clique partition problem* in graph theory. If we treat each player as a node and connect two nodes with an edge if their latency is below the developer's constraint, we can cast the grouping problem as partitioning the graph into cliques with the objective of minimizing the number of cliques under the clique size constraint. Finding the minimum clique partition of a graph is NP hard [25]. Polynomial time approximation algorithms for this problem exist only for certain graph classes [22]. Gamers are rather impatient and would prefer not to spend much time in the matchmaking lobby waiting for a match to happen. Grouping should run fast and scale to a large number of players in case a game becomes popular.

This grouping problem for P2P games is markedly different from that for client-server games. In client-server games, each player typically picks a game server based on a combination of server load, client-server latency, and number of players on a server [15]. This selection does not take into account the latency of other players who have picked that server, and the player may still experience poor gameplay if other players have chosen poorly.

3. ESTIMATING CELLULAR LATENCY

Each multiplayer game will have its own tolerance for latency between players. For instance, a fast-paced, action-packed shooter game may require that for 90% of traffic, the latency should be under 150ms, while a different game may tolerate 250ms at the 90^{th} percentile because it uses bows-and-arrows or uses very sophisticated dead reckoning. If the matchmaking service knows in advance what latency each player will experience to each of the

[1]After a game session has been formed and gameplay has begun, some games still allow new users to join the session. This type of matchmaking is easier given that for each new player, the choice is among a (much smaller) number of ongoing game sessions. In this paper, we focus on the original, harder problem of grouping for new game sessions.

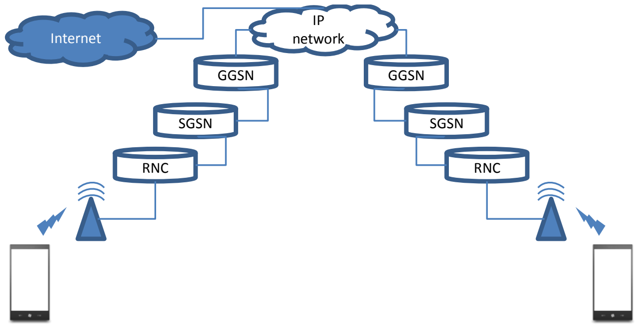

Figure 3: Simplified architecture of a 3G mobile data network. RNC is a Radio Network Controller, and handles radio resource management. SGSN is a Serving GPRS Support Node, and it handles mobility management and authentication of the mobile device. GGSN is a Gateway GPRS Support Node, and interfaces with the general IP network that a mobile operator may have and the Internet.

potential peers for the duration of a future game session, it can appropriately assign players to each other. Due to lack of information from the future, we need to predict the future latency based on information currently available.

We now present our findings from several measurements we have conducted to shed light on how we can predict future latency on 3G networks. Our measurements have been taken over multiple days, in each of several locations: Princeville (Kauai, HI), Redmond (WA), Seattle (WA), Los Angeles (CA), Durham (NC). In almost all cases, each graph that we present is visually similar to those from other locations and days. When they are not similar, we present the dis-similar graphs as well for comparison. Our measurements were conducted primarily using a pair of HTC Fuze phones running Windows Mobile 6.5 on AT&T Wireless, however we also have measurements from a pair of Google Nexus One phones running Android 2.2 on T-Mobile in Durham. Except when explicitly indicated, we restrict all our measurements in this paper to the HSDPA version of 3G, and only consider measurements after the cellular radio has achieved the "full power" DCH mode. Except when explicitly indicated, in each experiment the phones were stationary.

We use the term FRH throughout the paper – it is the "First Responding Hop" – that is, when a traceroute is attempted from a phone to any destination on the Internet, it is the first hop beyond the phone to respond with ICMP TTL Expired packets (typically at a TTL of 2). Based on our measurement experience and textbook understanding of HSDPA 3G cellular networks, we believe this is the GGSN [20]. This device is deep in the mobile operator's network, and all IP traffic to/from the phone traverses this device, as shown in Figure 3. When considering latency variation, we focus on the FRH because much of the variability is to this observable hop, and subsequent latencies to various Internet endpoints show almost no additional variability in comparison. Our measurements use 40 byte packets, and occur at the rate of once every 80ms to a variety of destinations – the rate to the FRH is once per second.

3.1 Predicting the future latency of a phone based on current latency

3.1.1 How does 3G latency vary over time?

For wired connections to the Internet, a common latency metric is the mean of a small number of pings. This generally suffices as a reasonable predictor of future latency for many applications because packet loss is typically minimal, and routing and congestion change at relatively longer timescales. However, it is unclear if the same applies for cellular data. We need to understand how latency varies over time – for instance, are latency measurements from one second representative of the duration of a game (e.g. next several minutes)?

In Figure 4, we show the latency that a phone experiences over a short duration of time. As the figure shows, there is a significant amount of latency variation, and at first glance, it does not appear that a 15 second window of measurements is very predictive of future latencies.

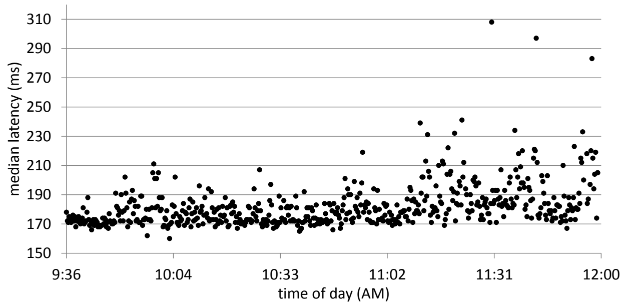

Figure 4: RTT from a phone in Princeville, HI on AT&T Wireless to the FRH. Each point is the median latency over 15 seconds. Graph is zoomed into a portion of the data to show detail. Data from Redmond, Seattle, Durham, and Los Angeles are visually similar.

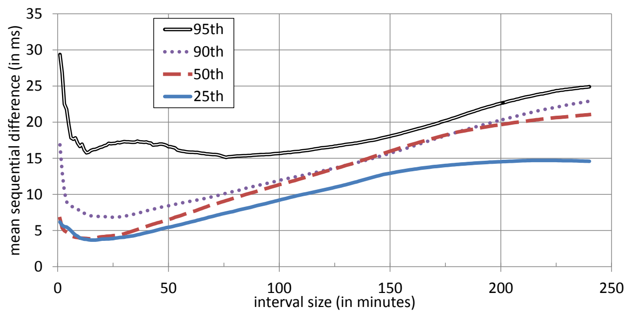

Figure 5: RTT from a phone in Redmond, WA on AT&T Wireless to the FRH. On the horizontal axis, we vary the length of time window over which we calculate the latency at the various percentiles indicated by the different lines. On the vertical axis, we show the difference in ms between two consecutive time windows at the different percentiles, averaged over the entire trace. Data from Princeville, Seattle, Durham, Los Angeles for AT&T Wireless are visually similar.

3.1.2 Over what timescale is 3G latency predictable?

If we pick too short a window of time over which to do latency measurements (e.g. 15 seconds), those measurements do not fully capture the variability of this connectivity and hence are not predictive of future latency. If we pick too long a time window, it may capture longer term drift in network characteristics, and require a larger measurement overhead. Thus we now vary the time window over which we compute a latency distribution, and examine how similar that latency distribution is to the next time window.

In Figure 5, we show the mean time window-to-window change

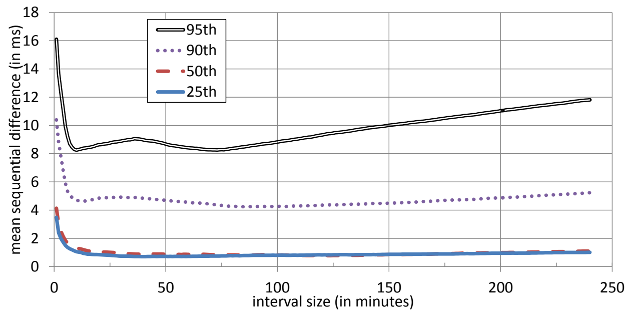

Figure 6: RTT from a phone in Durham, NC on T-Mobile to the FRH. On the horizontal axis, we vary the length of time window over which we calculate the latency at the various percentiles indicated by the different lines. On the vertical axis, we show the difference in ms between two consecutive time windows at the different percentiles, averaged over the entire trace.

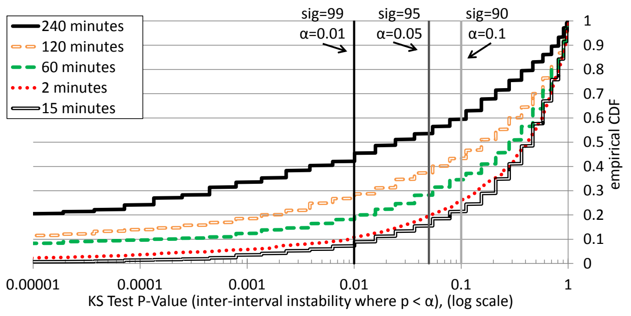

Figure 7: CDF of Kolmogorov-Smirnov (KS) test Goodness-of-Fit P-values for successive time windows by window size (in minutes) for a phone in Redmond, WA on AT&T Wireless. Each data point represents a two-sample KS test using 100 points from each of two successive time windows. The percentage of null hypothesis rejection is shown as the intersection of a distribution with the chosen significance level. A lower percentage of rejected null hypotheses is an indication of greater stability across successive time windows. The horizontal axis is clipped on the left. For clarity, a limited set of window sizes are shown. Data from Princeville, Seattle, Durham, Los Angeles are visually similar.

in latency to the FRH, which shows a dip around 15 minutes. Across different time durations, this is the duration where one measurement window is most similar to the next window. Note that across our different measurements, this analysis for the T-Mobile network in Durham exhibited slightly different behavior, and hence we present Figure 6. However again 15 minute time windows are most predictable of the next for this different network (using similar HSDPA 3G technology but at different radio frequencies). For a more rigorous statistical analysis, we include Figure 7, which confirms the highest stability for the 15 minute duration.

3.1.3 *For how many future time windows is one window predictive of?*

We have empirically established that latency measurements from a 15 minute time window are fairly predictive of the immediately subsequent time window. In Figure 8, we consider how rapidly this predictive power degrades over successive 15 minute time win-

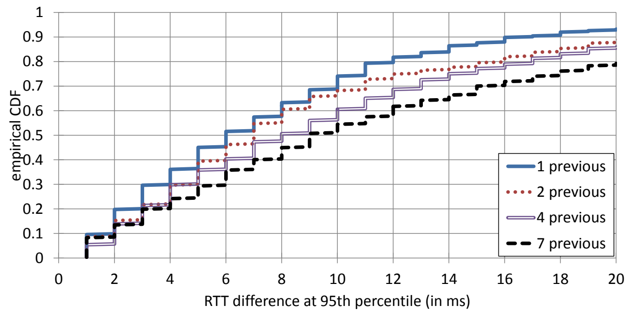

Figure 8: For any given 15 minute time window, from how far back in time can we use latency measurements and still be accurate? The horizontal axis shows the difference in latency at the 95th percentile between a time window and a previous time window. The age of the previous time window is shown in the legend. The vertical axis shows the CDF across all the different 15 minute intervals in this trace. The horizontal axis is clipped on the right.

dows. If a game developer is concerned about the 95th percentile of latency that players experience, we see that using measurements that are 15 minutes stale give an error of under 17ms for 90% of the time. If we reach back to measurements from 105 minutes ago (the "7 previous" line), this error increases to 29ms. For brevity we do not show graphs of other percentiles, but for instance, at the 50th percentile the errors are 8ms and 12ms respectively. For the remainder of this paper and in the design of Switchboard, we use measurements that are stale only by 1 time window to minimize prediction error. However, the measurement overhead of our system can be improved by allowing older, less accurate predictions.

3.1.4 *How many measurements are needed in each time window?*

The results we have presented so far have been generated from latency measurements at the rate of once per 1 second. For a 15 minute window, 900 measurements can be a significant overhead for a phone, both on the battery and on a usage-based pricing plan. In Figure 9, we consider by how much we can slow down this measurement rate while still obtaining a latency distribution that is similar. If the sampling rate is once per 15 seconds, there is relatively little degradation in the latency distribution. There is less than 11ms difference at the 50th percentile for all of the latency distributions for every 15 minute window in the trace, between sampling at once per 1 second and once per 15 seconds. For the 95th percentile, for more than half of the time windows, the difference in latency is only 11ms. We believe that sending and receiving 60 packets over the course of 15 minutes is a reasonable trade-off between expending limited resources on the phone and measurement accuracy.

3.2 Using the latency of one phone to predict the future latency of a different phone

So far, our experiments have found that a phone needs to measure its latency about 60 times in a 15 minute window to predict its latency in future 15 minute windows. While this significantly improves accuracy compared to naively using a few ping measurements, and significantly reduces overhead compared to naively pinging continuously for several minutes, there is still a large network overhead to every phone measuring its own latency when multiplayer mobile gaming becomes popular.

We now consider the extent to which one phone's latency is representative of another phone's, and hence reduce the burden on the

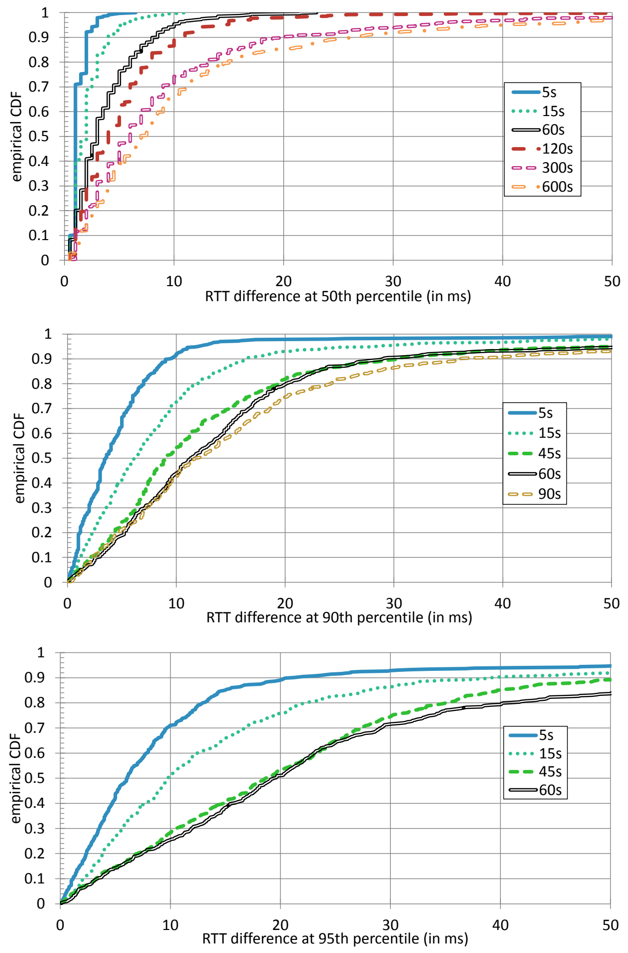

Figure 9: Impact of reducing the measurement sampling rate for a 15 minute window of latency from a phone in Durham, NC on AT&T Wireless to the FRH. The horizontal axis shows the difference in latency at the specified percentile between using a measurement rate of once per 1 second and using a measurement rate of once per 5 to 600 seconds as indicated. The vertical axis shows the CDF across all the different 15 minute intervals in this trace. Note that at a sampling rate of once per 90 seconds for 15 minutes, we have only 10 samples and hence we cannot calculate the 95th percentile. The horizontal axis is clipped on the right. Data from Princeville, Redmond, Seattle, Los Angeles are visually similar.

network by sharing recent measurements with other phones. As we showed in Figures 1 and 2, a phone in Redmond has very different latency to one in Durham, and hence we need to determine what parameters of connectivity that if are the same between two phones also mean that they share similar latency.

3.2.1 *Connectivity parameters with little influence*

We have conducted many experiments in several locations to explore how much different connectivity parameters influence a cellphone's latency. Due to lack of space and for conciseness, we now briefly summarize our negative findings before presenting our positive findings in more detail.

We know from prior work [7] that the public IP address a phone is assigned can vary and does not correlate with its location. Using similar data to that prior work, we reconfirmed this finding and hence do no believe that the public IP address is representative of a phone's latency. In fact, for Figures 1 and 2, each phone had the same IP address regardless of which location it was in.

Our experiments show no discernible correlation between 3G signal strength at the phone and its latency to the FRH. While initially counter-intuitive, this observation is borne out by our "textbook" understanding of modern cellular standards. Unlike earlier 3G standards [10], the HSDPA standard provides a reliable wireless link to the cellular phone by performing retransmissions at the L1 physical layer between the phone and the celltower [20]. Poor signal strength should result in higher BLER (block error rates). However, in general, power control and adaptive modulation keep the channel fast and efficient at a variety of signal strengths. BLER appears to be a concave function of SNR [20], and hence the signal strength has to be extremely low before a bad BLER of above 10% is experienced. Furthermore, the use of dedicated channels, short TTI (transmission time interval) of 2ms, and explicit ACK / NACK mechanisms mean that retransmitting corrupted blocks is extremely fast (in the order of a few ms).

We have also conducted experiments at a variety of speeds while driving in city streets and highways. After accounting for celltower changes, we see little correlation between speed and latency, though unfortunately the highway patrol did not let us conduct experiments much beyond 60mph.

Our experiments do show a long term trend in latency variation that suggests a diurnal effect, that we suspect is due to human-induced load on the network. However, the same time window from one weekday is not very predictive of the same window for the next weekday or the same day the following week.

All of the results we present in this paper are specific to HSDPA connectivity. Earlier versions of 3G do exhibit different latency distributions, and especially 2G technologies GPRS and EDGE which are dramatically different. We do not explore these older technologies further in this paper.

3.2.2 *Phones under the same celltower*

When a phone is connected to a cellular network, certain parameters of that connectivity are exposed to the mobile OS – CID, LAC, MNC, MCC. The CID (Celltower ID) is a number that uniquely identifies the celltower. The LAC (Location Area Code) is a logical grouping of celltowers that share signaling for locating a phone (a phone that roams between celltowers with the same LAC does not need to re-update the HLR and VLR location registers). The MNC and MCC numbers uniquely identify the mobile operator (*e.g.,* AT&T Wireless or T-Mobile).

We conducted experiments where we placed one phone at a fixed location for a long duration of time, and placed a second phone for 30 minutes each in a variety of locations. One such experiment was in Seattle, another in Redmond, and one in Durham. Figure 10 shows maps of each of these locations and Table 1 shows the CID and LAC numbers for the celltowers that the phones connected to.

Figure 11 shows the results of the Seattle experiment. The "S-home" lines show the difference in latency when both phones were placed next to each other, which is about 30ms in most instances. The "Latona" lines show the difference when both phones were connected to the same celltower but one was further away. These lines are almost indistinguishable from the "S-home" lines. The other locations have different CIDs from "S-home". Some of these locations have very different latency to the stationary phone at "S-home", while some are similar. We see similar behavior with experiments in other locations. In Figure 12, the locations with the same CID as the stationary phone experience similar latency. Of the ones with different CIDs, one has very different latency ("REI")

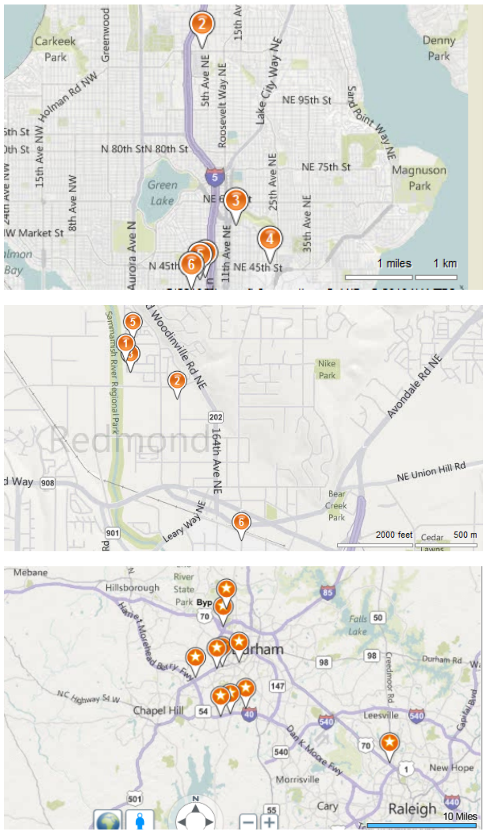

Figure 10: Maps showing measurement locations in (top) the Seattle area of Washington, (middle) the Redmond area of Washington, (bottom) the Durham and Raleigh areas of North Carolina.

Seattle Experiment		
location	**CID**	**LAC**
S-home	10932	42981
Latona	10932	42981
Northgate	11403	42980
U Village	11038	42981
Herkimer	11847	42981
1st Ave	12192	42981

Redmond Experiment		
location	**CID**	**LAC**
M-home	15539	42993
mailbox	15539	42993
J-home	15539	42993
H-home	15539	42993
QFC	15499	42993
REI	15341	42993

Durham Experiment		
location	**CID**	**LAC**
R-home	919	12998
Breakfast Restaurant	919	12998
J-Home	308	12998
Large Retailer	308	12998
Durham Mall	1618	12998
Raleigh Mall	337	12998

Table 1: Network parameters observed at 6 different locations in each of the three experiments – top left is Seattle, top right is Redmond, bottom is Durham. In all cases, the phones were connected to AT&T Wireless with MNC 410 and MCC 310, over HSDPA with 3-4 bars of signal strength. For the Seattle experiment, one phone was left at "S-home" while the other visited each of the 6 locations. For Redmond, the stationary phone was at "M-home", and for Durham it was "R-home".

while another has very similar latency ("QFC"). With Durham in Figure 13, the two locations with the same CID ("R-home" and "Breakfast Rest." have similar latency, while most of the other CIDs are different. We have seen similar behavior across different

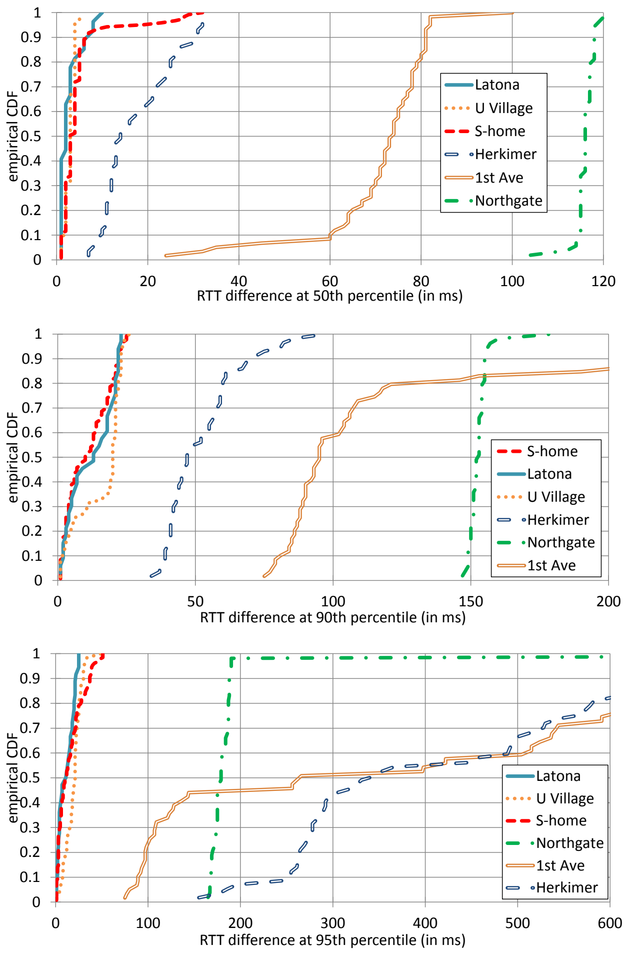

Figure 11: Difference in latency between a stationary phone at "S-home" and a phone placed at a variety of locations in Seattle. Each line is a CDF of ((x^{th} percentile latency over a 15-minute interval from stationary phone at "S-home") - (x^{th} percentile latency over the same 15-minute interval for the other phone at the location in the legend)) computed for all possible 15-minute windows, in 1 minute increments. The x^{th} percentile is 50^{th} for the top graph, 90^{th} for the middle, and 95^{th} for the bottom. Horizontal axis is cropped on the right.

days (both on weekdays and weekends) and several other locations in each of these areas, but we do not enumerate those experiments here for conciseness.

From these experiments, we believe that phones under the same RNC (see Figure 3) experience similar latency. The RNC is a physical grouping of celltowers, where the RNC controls radio resource allocation for the celltowers under it. We believe that latency depends a large part on congestion and provisioned capacity, which varies from RNC to RNC, and this theory is also suggested by prior work based on 3G measurements in Hong Kong [28].

Unfortunately, the identity of the RNC is not exposed to the OS on the phone, as far as we know. While the LAC identity is exposed, LAC is a logical grouping having to do with signaling which has little impact on latency once a phone has initiated a data connection. Not knowing which RNC a celltower is part of, we use the more *conservative* approach of sharing latency profiles between phones connected to the same CID only. There will be phones under other celltowers with similar latency as our experiments show,

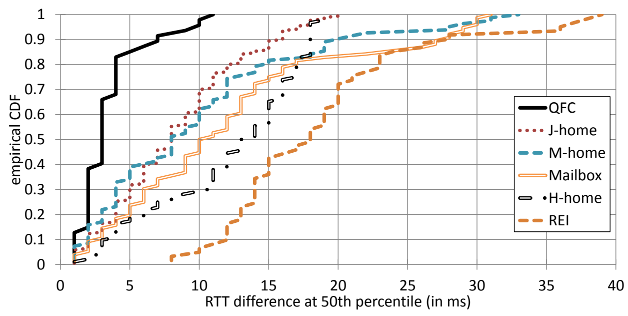

Figure 12: Difference in latency between a stationary phone at "M-home" and a phone placed at a variety of locations in Redmond. Each line is a CDF of ((xth percentile latency over a 15-minute interval from stationary phone at "M-home") - (xth percentile latency over the same 15-minute interval for the other phone at the location in the legend)) computed for all possible 15-minute windows, in 1 minute increments. For conciseness, we present only the 50th percentile graph. Horizontal axis is cropped on the right.

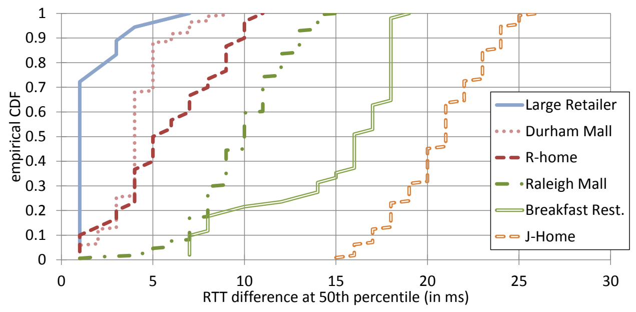

Figure 13: Difference in latency between a stationary phone at "R-home" and a phone placed at a variety of locations in Durham. Each line is a CDF of ((xth percentile latency over a 15-minute interval from stationary phone at "R-home") - (xth percentile latency over the same 15-minute interval for the other phone at the location in the legend)) computed for all possible 15-minute windows, in 1 minute increments. For conciseness, we present only the 50th percentile graph. Horizontal axis is cropped on the right.

but we are unable to reliably identify them.

3.3 Predicting the latency between phones

So far, the results we have presented have been intentionally limited to predicting the latency between a phone and its FRH. We have found the ideal duration of time over which measurements need to be taken, how many measurements are needed, and among which phones these measurements can be shared to reduce overhead.

However, for P2P multiplayer gaming, we need to predict the end-to-end latency between pairs (or more) of phones. From traceroutes we have done between phones in the same location and across many different locations in the US, the end-to-end latency appears to be the sum of the component latencies – phone1 to FRH1, FRH1 to FRH2, FRH2 to phone2. This is obviously expected behavior, and also shown in Figure 14.

The remaining task is to predict the latency between a pair of FRH. This is a very traditional problem of scalable prediction of

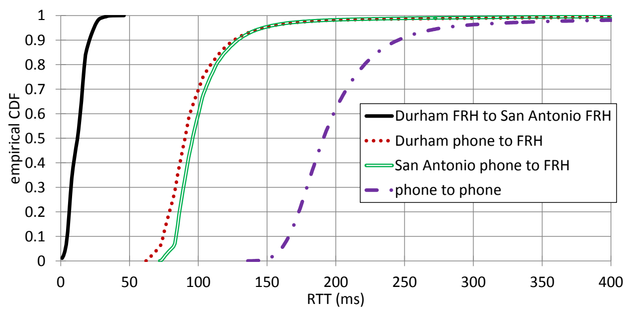

Figure 14: CDF of RTT between a phone in Durham, NC and a phone in San Antonio, TX. Component latencies involving the respective FRH are also included. The FRH to FRH latency is calculated by the difference of pings. The horizontal axis is clipped on the right. Note that the phone-to-phone CDF is not a perfect sum of the other three CDFs due to small variations in latency in between traceroute packets issued at the rate of once per second.

the latency between two points on the Internet. We can use prior techniques such as Vivaldi [13], Pyxida [23], or Htrae [5], to name just a few. These techniques work well if the latency does not vary tremendously over short time scales, which is true of many types of wired Internet connectivity. In Figure 14, we see that the leftmost line, which is the FRH to FRH latency, is fairly straight in comparison. Hence, we rely on the demonstrated effectiveness of prior work to solve this problem and do not discuss it in more depth here.

In the next section, we present the design of Switchboard and describe how we use our findings on 3G latency to improve the scalability of matchmaking.

4. Switchboard

The goal of a matchmaking service is to abstract away the problem of assigning players to game sessions so that each game developer does not have to independently solve this. A successful game session is one in which the latencies experienced by every player meet the requirements of the game and the number of players is as large as possible for the game. In using the matchmaking service, the game developer specifies the latency requirements of the game, and the number of players it can support.

The matchmaking service has to operate in a scalable manner. The amount of measurement overhead for each player and on the network in general has to be as small as possible. This is especially true of games on phones, where the phone has limited energy and the network has relatively limited capacity. The amount of time that a player spends in the matchmaking lobby has to be minimal as well, and should not grow significantly as a game becomes popular and more users participate in matchmaking. We now briefly describe the design of Switchboard, and in particular point out how it scales while trying to achieve low latency but large matches.

4.1 Architecture of Switchboard

As Figure 15 shows, there are two sets of components to Switchboard – components on the clients and components on the centralized, cloud-based service.

The Switchboard client functionality is split into two parts. The developer's game code interfaces with the Switchboard Lobby Browser – this component interacts with the Lobby Service running in the cloud. The API is described next in § 4.2. The other

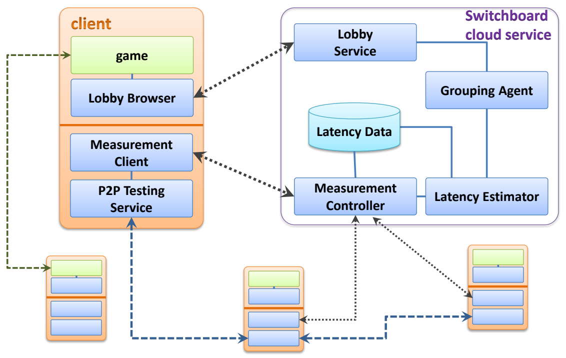

Figure 15: Architecture of Switchboard

```
class BoomLobby : LobbyBrowser {
  private CloudInterface myInterface;
  private BoomClient myClient;
  private StartGameCallback myClientStart;
  private int latencyPercentile = 95;
  private int latencyLimit = 250;
  private int maxPlayers = 16;
  public void BoomLobby(string gameLevel) {
    string Hash = "Boom" + gameLevel;
    myInterface = new CloudInterface(Hash,
      latencyPercentile, latencyLimit, maxPlayers,
      new MatchReadyCallback(this.MatchReady));
  }
  public void Join(BoomClient client, StartGameCallback sg) {
    myClient = client;
    myClientStart = sg;
    myInterface.AddClient(client);
  }
  public void MatchReady(BoomClient[] clients) {
    myClientStart(clients);
  }
}
class BoomClient : MatchmakingClient {
  public void BoomClient(string gameLevel) {
      BoomLobby myLobby = new BoomLobby(gameLevel);
      myLobby.Join(this,
        new StartGameCallback(this.StartBoomGameNow));
    }
  public void StartBoomGameNow(BoomClient[] players) {
      // ... Do game stuff here
    }
}
```

Figure 16: C# client API of Switchboard as would be used in a hypothetical game called "Boom". For brevity, base class definitions are not shown here.

part of the Switchboard client produces network measurements for the cloud service to consume, and this is described in § 4.3. This part of the client does not directly interact with the developer's game code.

The Switchboard cloud service handles matchmaking requests from the Lobby Browser on the client. The Lobby Service interacts with the Grouping Agent to place the client in a game session. The Grouping Agent, described in § 4.4, in turn uses the Latency Estimator to assign clients to game sessions based on their measured or estimated latency profile. If there is insufficient or out-of-date measurement data, the Latency Estimator may ask the Measurement Controller to schedule measurements on various clients.

4.2 Client API for Lobby Browser

Different games have different requirements for successful matches. Our design for the Switchboard API must simultaneously provide ample flexibility in specifying requirements while remaining intuitive. We enable developers to impose three types of constraints on game formation: (1) a simple requirement for the maximum number of players sharing the same game session; (2) a simple restriction on the distribution of end-to-end latency between any pair of players; and (3) segregating players into different matchmaking lobbies based on the type of game they want to play (*e.g.,* players that want to play "Boom" with map "hangar" is a completely different game from players that want to play "Boom" with map "zombie base").

Figure 16 summarizes the API that the game developer uses to interact with the Lobby Browser component of Switchboard. The game has to implement and instantiate a derived class of `MatchmakingClient`. It instantiates a derived class of `LobbyBrowser` by specifying the game level that the player has selected, and the callback function that tells the game client that matchmaking has been done. The `LobbyBrowser` interfaces with the Lobby Service in the cloud, but this interaction or API is hidden from the game developer. The `LobbyBrowser` just needs to instantiate a `CloudInterface`, which specifies the hash for this lobby, the latency percentile of interest, the limit for this percentile, the maximum number of players, and the callback function. In Switchboard, we uniquely identify each matchmaking lobby with a hash of the combination of the game's name and the map or level. Grouping is conducted independently for each unique hash. The `latencyPercentile` of the latency distribution between any two `MatchmakingClients` should be less than `latencyLimit`. We have provided a simple example in the figure, where the developer wants the 95th percentile to be under 250ms.

On the cloud service side, the Lobby Service will interact with the Grouping Agent on the client's behalf. When the Grouping Agent returns with a list of matches, the Lobby Service will hand to every client the list of other clients it has been matched with. A client may wish to re-join the lobby if a null list is returned (because there are no other players at this time, or no others with low enough latency).

4.3 Latency Estimator

The Latency Estimator supports the Grouping Agent. The Grouping Agent may need the latency distribution between any arbitrary pair of clients. Specifically, it will request latency distributions only between those clients that have the same lobby hash. It will apply the distribution test that the game developer has provided. Each client is identified by a unique ID, and details of its connectivity (CID, MNC, MCC, FRH, radio link technology) – this identification is created transparently for the game developer in the `MatchmakingClient` base class definition.

The Latency Estimator relies on data stored in the Latency Data database. This database contains raw latency measurements, of the same form as the data in the experiments in § 3. Each record has a timestamp, client unique ID, client connectivity (CID, MNC, MCC, FRH, radio link technology), RTT latency, destination FRH. The record is identifying the RTT latency between that client and the destination FRH that it probed, which may be its own FRH or a remote FRH. The database keeps a sliding window of measurements from the past 15 minutes, as § 3 shows that older data is of lower quality.

The database is fed by the Measurement Controller. The Measurement Controller divides the global set of clients (that are currently in any matchmaking lobby) into three queues: (1) the *free pool*; (2) the *active pool*; and (3) the *cooloff pool*. Clients in the free pool are grouped by their CID, and one client under each CID is chosen at random. The chosen clients are moved into the active pool and are requested to conduct a batch of measurements (the quantity and duration of measurements is configurable; we assume 10 probes per request in our experiments). Once they report back measurements, they enter the cooloff pool.

Information from the Latency Estimator determines if clients are moved from the cooloff pool into the free pool. The Latency Estimator identifies for which CIDs it does not have sufficient measurements – at least 60 measurements within the last 15 minutes from any clients under that CID to their common FRH. This list of CIDs is handed to the Latency Estimator (every 30 seconds), which moves all clients under any of these CIDs from the cooloff pool into the free pool (and any that are not into the cooloff pool).

When the Measurement Controller asks a client to perform measurements, it hands over three parameters: (1) the measurement duration; (2) the measurement rate; and (3) a list of unique FRHs. The Measurement Client will interleave pings from the phone to its FRH with pings to a randomly-selected distant FRH. At the end of the measurement duration, the results are reported back. The measurement rate has to be high enough to keep the phone radio awake in DCH mode – in our experience, sending a packet every 100ms suffices.

The Grouping Agent calls the Latency Estimator to get a latency distribution between a pair of clients. The Latency Estimator computes this as the sum of three components: (1) latency of the first client to its FRH; (2) latency of the second client to its FRH; and (3) FRH to FRH latency. For between a client and its FRH, the Latency Estimator calculates a distribution among all latency measurements from any client under the same CID to the same FRH, from the past 15 minutes [2]. For FRH to FRH latency, we rely on a system like Htrae [5]. It feeds on the Latency Data database, but subtracts client to FRH latency from client to remote FRH latency to feed the network coordinate system. Since we do not have a geo-location database that works with FRH IP addresses, this is practically similar to Pyxida [23].

4.4 Grouping Agent

For each unique lobby hash, the Lobby Service hands over to the Grouping Agent the list of clients, the maximum number of players, and the latency test parameters [3]. The Grouping Agent treats each lobby hash completely separately (there are multiple instances of the Grouping Agent, each handling one hash).

The Grouping Agent obtains the latency distributions between each pair of clients in this lobby from the Latency Estimator. It constructs a graph of these clients. The weight (or length) of the edge between two clients is the latency between them at the given percentile. This graph, along with the latency and size limits from the game developer, are handed to the grouping algorithm, described next in § 4.5.

Once the grouping algorithm successfully places clients into game sessions, it returns the list of sessions to the Lobby Service. A session is viable only if it has at least 2 players. The Lobby Service removes all clients in viable sessions from the Measurement Controller's client pools and returns the list of session peers to the respective clients. Any clients that were not placed in a viable session remain in the lobby for another round of matchmaking or until they voluntarily leave (that part of the API is not described in Figure 16). Clients can remain in the lobby when there is insufficient latency data for that client's CID, or there are insufficient players with whom they can form a viable session.

4.5 Grouping algorithm

The goal of the algorithm is to assign players to different groups in a way that: i) maximizes the number of players in each group; and ii) satisfies the latency and group size constraints specified by game developer. An important factor that affects the grouping process is the topology formed by players within a group. While we now focus on grouping for the clique topology, our grouping algorithm can be easily adapted to accommodate other topologies.

As mentioned in §2.5, the grouping problem can be casted into the minimum clique partition problem which is NP-hard. Given that a popular mobile game may attract tens of thousands of players, we need an algorithm that is both effective and scalable. We find that cluster analysis [3] is particularly well suited to solve the grouping problem. Clustering refers to the assignment of a set of observations into clusters based on a *distance* measure between observations. If we treat each player as an observation and the latency between two players as their distance, we can leverage a wealth of well-established clustering methods to solve the grouping problem.

While there exist many clustering methods, we pick *hierarchical* [18] and *quality threshold* (QT) [19] clustering because they have low computational complexity and can easily accommodate different group topologies (*e.g.,* clique and star-hub). We do not consider K-means (which is another commonly-used clustering method) because it requires specifying the number of clusters a priori, making it difficult to enforce the latency constraint.

Hierarchical clustering starts with each individual player as one cluster. It progressively merges pairs of closest clusters according to a *distance* measure, until the cluster *diameter* exceeds the latency constraint. In contrast, QT clustering first builds a candidate cluster for each player by progressively including the player closest to the candidate cluster (according to a *distance* measure), until the candidate cluster *diameter* exceeds the latency constraint. It then outputs the largest candidate cluster, removes all its members, and repeats the previous step. Note that when the size of a cluster is too large, we need to further divide it into smaller clusters to meet the group size constraint.

For the clique topology, the *distance* between two clusters is defined as the maximum latency between players of each cluster. The *diameter* of a cluster is defined as the maximum latency between players in the cluster. The time complexity of hierarchical and QT clustering is $o(n^3)$ and $o(n^5)$ respectively.

We emphasize that both clustering methods can work with any type of topology in which *distance* and *diameter* are well defined. For instance, in star-hub topology, the distance between two clusters can be defined as the latency between the hub players of each cluster. The diameter of a cluster can be defined as the maximum latency between the hub player and any star player in the cluster.

The grouping algorithm is polynomial, and hence its running time can be long when there are a large number of players. Waiting in the matchmaking lobby for a long time can degrade the experience of game players. To tackle this problem, we first divide all the players into smaller buckets with at most B players in a bucket, and then apply the grouping algorithm to each bucket. In this way, we can easily parallelize the grouping of all the buckets and control the grouping time by adjusting B. While a smaller B shortens grouping time, it can lead to less optimal groups. We evaluate how B impacts grouping time and group sizes in §5.2. In our current implementation, we randomly assign players to different buckets. In the future, we plan to explore other assignment strategies, such

[2]If there are insufficient measurements, the return value to the Grouping Agent identifies the client(s) with insufficient data and they are removed from the current grouping round (and remain in the measurement pools).

[3]For any particular lobby hash, we expect all game clients to specify the same `latencyPercentile`, `latencyLimit` and `MaxPlayers`. Alternatively, we can incorporate these three parameters into the hash itself to further segregate players.

as based on geographic location or LAC.

While waiting in a matchmaking lobby for grouping to finish, other players may join or leave the lobby. The graph that the grouping algorithm is operating on could be modified in real time as the algorithm runs. However, for simplicity, the Lobby Service in Switchboard calls the Grouping Agent at fixed intervals for each lobby. This not only limits grouping overhead but also allows the accumulation of a sufficient number of players to feed into the grouping algorithm. The choice of the interval needs to balance player wait time with the popularity of a particular game.

5. IMPLEMENTATION & EVALUATION

5.1 Implementation

The service side of Switchboard is implemented on the Microsoft Azure cloud platform. In our current deployment, we use a single hosted service instance and a single storage service instance, both in the "North Central US" region. Matchmaking itself is not very sensitive to small latencies (hundreds of ms) because it is only sets up the game session and is not used during gameplay itself. Hence we have deployed only a centralized instance of the service.

The service is written entirely in C#, and heavily leverages the .NET 4.0 libraries. The Measurement Controller is written in 457 lines of code, with an additional 334 lines of message formats and API that is shared with the client. The Lobby Service is 495 lines of code. The Latency Estimator is 2,571 lines of code, but contains a very large amount of analysis code to support this paper and can be significantly slimmed. The Grouping Agent is 363 lines.

The client side of Switchboard is a mix of C# and native C code. The Measurement Client is 403 lines and the 334 shared with the Controller. The P2P Testing Service which actually handles the probes is written in C due to the lack of managed APIs for getting connectivity information and doing traceroutes. The client side is implemented for Windows Mobile 6.5. However, we have a port of just the P2P Testing Service to Android, which we used to help gather data for this paper. We also use a simple client emulator of 48 lines to to stress test our service on Azure.

5.2 Evaluation of grouping

We now evaluate the importance of pairwise latency estimation for effective grouping and the impact of bucket size on grouping time and group sizes. To evaluate grouping at scale, we need a large number of players and their latency data. Unfortunately, we are not aware of any large corpus of detailed latency measurements from a wide set of phones. Therefore, we attempt to generate a synthetic model of phones, their locations, the locations of towers, the locations of FRHs, and the latencies associated with each. We then evaluate how grouping performs on such a topology.

To generate a realistic distribution of players, we use population data by county from the US census [2]. To cluster users by tower, we use cell tower locations from a public US FCC database [4], which contains detailed information about the cell towers registered with the FCC. Combining these two data sources, we break down the towers by county and compute the fraction of total population served by a tower T in county C as $FracPop(T) = \frac{pop(C)}{ntower(C) \times TotalPop}$. Here $pop(C)$ and $ntower(C)$ are the population and number of towers in C and $TotalPop$ is the total population.

Next, we need to connect the towers to FRHs. Today, US operators have many FRHs, but they are typically co-located in a few datacenters across the US (based on private conversations with operators). Not knowing where these datacenters are, we simply divide the US into four Census Bureau-designated regions (Northeast, Midwest, South, and West), and pick a metropolitan area from each region (Washington DC, Chicago, San Antonio, and San Francisco) as the FRH datacenter location.

Finally, we generate a set of n players using this model. For each tower T, we generate $n \times FracPop(T)$ players. Essentially, we proportionally assign n players to each tower according to the population density of the county in which the tower sits. For each player p under T, we randomly pick its geographic coordinate ($lat(p)$ and $lon(p)$) within a predefined radius of T. The maximum range of a tower varies from 3 to 45 miles, depending on terrain and other circumstances [1]. We picked a radius of 20 miles. We also assign p to the geographically closest FRH datacenter ($FRH(p)$). The RTT between p and $FRH(p)$ ($rttFRH(p)$) is randomly drawn from the latency distribution to the first pingable hop collected by prior work [21] from 15,000 mobile users across the US. As per our findings in §3.2.2, all players under the same tower are assigned the same RTT to their corresponding FRH datacenter.

We can now compute the latency between any pair of players (p_1, p_2) as:

$$rttFRH(p_1) + rttFRH(p_2) + rtt(FRH(p_1), FRH(p_2))$$

$rtt(FRH(p_1), FRH(p_2))$ represents the RTT between $FRH(p_1)$ and $FRH(p_2)$, which we derive from a geographic distance-based latency model from prior work [5]. We compute the geographic distance using the great-circle distance between a pair of coordinates.

5.2.1 *Latency- vs. geography-based grouping*

To contrast with our algorithm, we also try a naive algorithm which groups players by their geographic proximity (instead of latency proximity in Switchboard). In this experiment, we evaluate the effectiveness of geography- *vs.* latency-based grouping. We first generate a game topology of 50,000 players and divide the players into buckets of 1,000 players each. We then run the hierarchical clustering algorithm (described in §4.5) on each bucket, using pairwise player latency or geographic distance. Note that each $GeoGroup$ produced by the geography-based grouping is guaranteed to meet the specified distance constraint. However, unlike in the latency-based grouping, a $GeoGroup$ may include players that violate the latency constraint specified by game developer. We further prune outliers from a $GeoGroup$ to obtain the corresponding *viable* group which fully satisfies the latency constraint.

Figure 17 shows the CDF of group sizes using the two algorithms. We set the game latency constraint to 250 ms and vary the distance constraint from 100 to 800 miles for geography-based grouping. The maximum group size is limited to 16. Clearly, latency-based grouping produces much bigger groups than geography-based grouping, with the median group size of 15 *vs.* 2. Although not shown in the figure, both grouping schemes assign roughly the same number of players to viable groups. Geography-based grouping does not work well because 3G latency between players is poorly correlated with their geographic proximity. This is unsurprising because our latency experiments show it is dominated by the latency to FRH and not by the latency between FRH s.

Figure 17 further shows that geography-based grouping produces larger viable groups when the distance constraint increases. However, this effect diminishes as the constraint surpasses 400 miles. Since there is little correlation between geographic distance and latency in mobile networks, the (viable) group size increase is mainly because a larger distance constraint produces bigger $GeoGroup$'s. Irrespective of the choice of distance constraint,

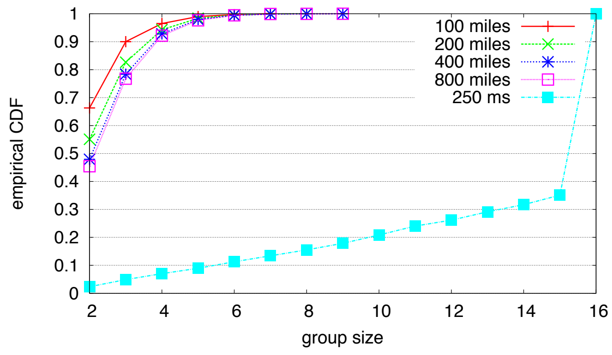

Figure 17: CDF of number of players in each group after grouping 50,000 players split into buckets of 1,000 players each, with a latency limit of 250ms. The top four lines show results from grouping players based on geographic proximity, while the bottom line uses latency proximity.

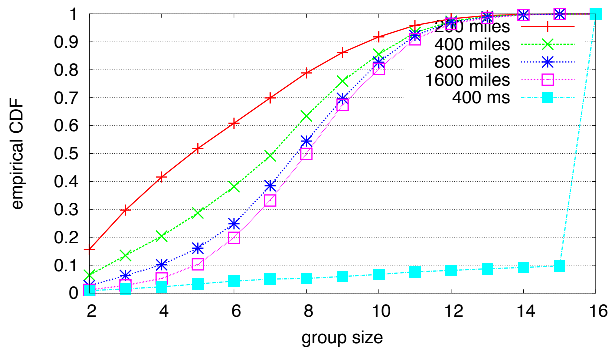

Figure 18: CDF of number of players in each group after grouping 50,000 players split into buckets of 1,000 players each, with a latency limit of 400ms. The top four lines show results from grouping players based on geographic proximity, while the bottom line uses latency proximity.

latency-based group dominates geography-based grouping. We see similar results in Figure 18 with a latency constraint of 400 ms.

5.2.2 *Effect of bucket size*

Having established that latency-based grouping is the better approach, we now consider the impact of bucket size and the particular form of clustering – QT or hierarchical. In Figures 19 and 20 we vary both and examine the impact on group size distribution and running time. While the total number of players who can participate in viable groups is roughly the same in each experiment (not shown in the figure), group sizes steadily grow with bucket size as expected. With a bucket size of 1000 players, 63% of the resulting groups have 16 players. This ratio improves to 75% with a bucket size of 1500. While group sizes are roughly similar between hierarchical and QT clustering, the running time is not. The running time for either grows with a larger bucket size, that for QT clustering grows much faster due to higher computational complexity (§4.5). Game players can be tolerant of a small delay (a minute or two) during matchmaking, as they can be appeased with game storyboard animation, but anything larger is less tolerable.

5.3 End-to-end Evaluation

We evaluate the performance of the complete end-to-end Switchboard system as a whole, including measurement and grouping.

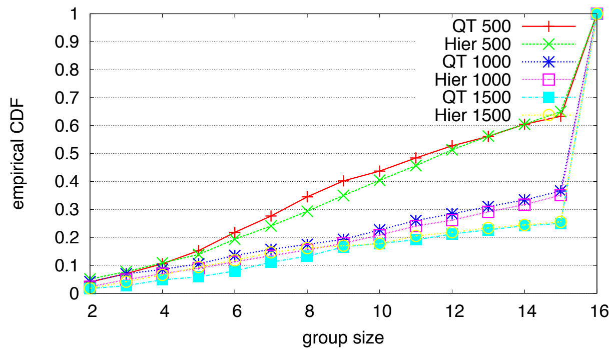

Figure 19: CDF of number of players in each group after grouping 50,000 players split into buckets of varying sizes, with a latency limit of 250ms. The "QT 500" line shows results with QT clustering on a bucket size of 500 players. The "Hier 1500" line shows results with hierarchical clustering on a bucket size of 1,500 players.

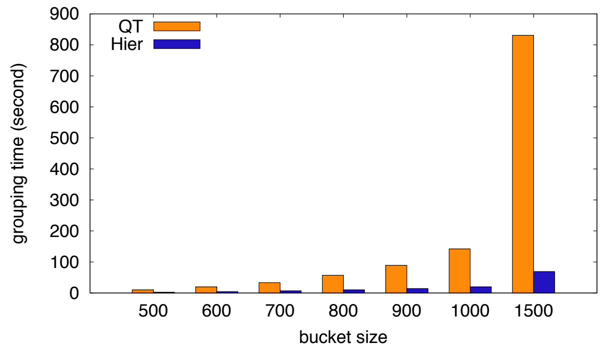

Figure 20: Runtime of grouping algorithms for grouping 50,000 players split into buckets of varying sizes, with a latency limit of 250ms. The "QT" bars on the left show results with QT clustering, while the "Hier" bars on the right show results with hierarchical clustering.

Our fully-functional Switchboard implementation is deployed as a Microsoft Azure cloud service. To consider a large client base, we emulate users by spawning new instances of the client emulator on a high-powered desktop (with new clients connecting at varied Poisson arrival rates). When requested by the Switchboard service to conduct measurement tasks, emulated clients use the same model we used to test grouping based on the US Census and FCC databases. Each client waits for placement in a matchmaking group until (1) such a group is formed or (2) Switchboard determines that the client's first-hop latency is too high, and thus group placement is impossible. We assume that each client only seeks matchmaking for a single game. In reality, clients may amortize the matchmaking costs over multiple games. We summarize experimental parameters in Table 2.

For each of these experiments we compare Switchboard performance across a variety of synthetic, Poisson-distributed, client arrival patterns. Of course, the validity of our results is tied to how well these align with the true arrival pattern of real clients. Thus, they should only be viewed in relative terms. However, a number of key properties emerge regarding performance at scale. As the number of clients using Switchboard increases, (1) server bandwidth requirements scale sub-linearly, (2) per-client probing (and thus

Parameter	Value
Maximum end-to-end latency bound	250 ms
Client arrival distribution	Poisson
Client arrival rate	Varies
ICMP measurement expiration period	15 min
ICMP measurement probes per server request	10
Per-tower measurements required for matchmaking	60
Client "cooloff" period between probe requests	30 - 90 s (random)
Bucket size for grouping	500

Table 2: Experimental parameters for end-to-end experiments.

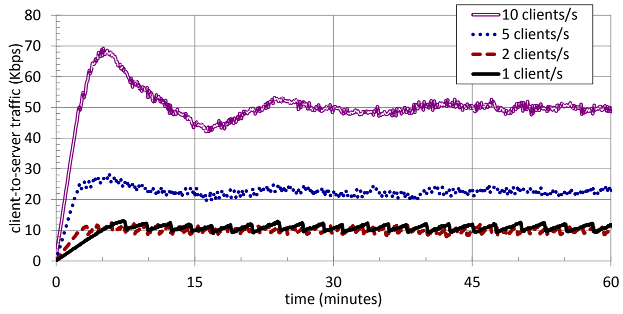

Figure 21: Aggregate client-to-server bandwidth by client Poisson arrival rate for Switchboard running on Azure. The first 15 minutes reflects a warming period with elevated measurement activity as the server builds an initial history.

bandwidth) overheads decrease, (3) larger groups can be formed, and (4) client delays for measurement and grouping decrease.

5.3.1 Bandwidth and Probing Requirements

We now quantify the client-to-server bandwidth requirements of phones reporting latency measurements to the Switchboard server and how probing tasks are distributed among devices. As explained in §4.3, probing is conducted in bursts of at least one packet per 100 ms, ensuring that the sending rate triggers the phone to enter the DCH mode. To amortize the energy cost of this traffic burst, phones report measurements in 10-probe batches. The frequency at which clients conduct these probe bursts depends on complex interactions, such as the rate at which clients arrive and when their measurements expire.

The bandwidth consumed by the matchmaking service to collect measurement data from phones is primarily determined by the total number of towers with active game players as the total number of measurements required for each tower is fixed, irrespective of the number of clients. Figure 21 shows client-to-server bandwidth over time, aggregated across all clients connected to Switchboard running on Azure. We require at least 60 measurements within the last 15-minute interval for each tower. Clients are not considered for groups until this minimum number of measurements have been conducted for their associated tower. The bandwidth consumed stabilizes after an initial warming period during which the Switchboard Measurement Controller builds an initial 15-minute history for many towers. Furthermore, as we increase the client arrival rate, the bandwidth consumed scales sub-linearly – at 10 clients/second, the bandwidth consumed is not 10 times that at 1 client/second.

Figure 22 shows the distribution of ICMP measurements performed by each client. As the client arrival rate increases, greater measurement reuse is possible because there are more clients under each tower that benefit from each other's observations. Further, the distribution of measurement tasks becomes more equitable (the CDF lines shift to the left), reflecting that at greater load,

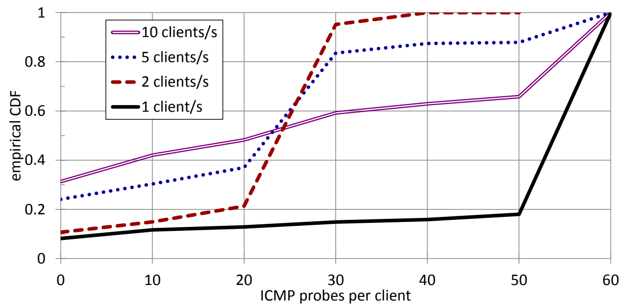

Figure 22: CDF of ICMP probes per client at different client Poisson arrival rates, as conducted by the Measurement Controller in Switchboard running on Azure. Data reflects hour-long experiments and exclude warming period.

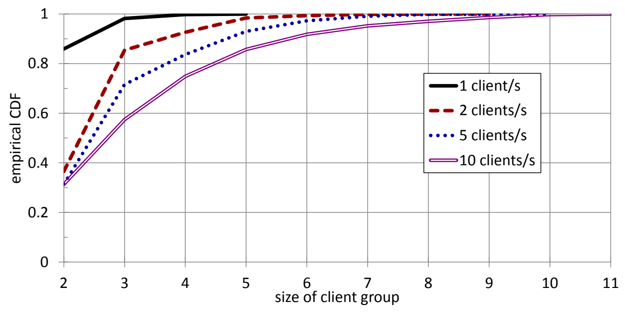

Figure 23: CDF of resulting group sizes at different client Poisson arrival rates. Grouping uses 500-client buckets. Data reflects hour-long experiments and exclude warming period.

Switchboard overhead for each client becomes lower and more predictable.

5.3.2 Client Matchmaking Experience

We now evaluate the size of viable groups that Switchboard creates and how long clients wait for those results.

In Figure 23, we show that at higher client arrival rates, it is possible to form larger matchmaking groups. This is expected, since at higher arrival rates, the steady-state number of waiting clients is also higher—providing a larger pool of clients on which to cluster. Note that the analysis in §5.2 reflects absolute grouping performance, with all clients available for clustering simultaneously and immediately. In this section, we additionally consider the effects of client arrival rate and re-grouping through multiple rounds, more closely reflecting real-world performance. Here, bucket size (chosen as 500) reflects the *maximum* number of clients that may be simultaneously clustered. At insufficient arrival rates, there will be fewer than 500 waiting clients. Further, since clients are placed into groups as soon as one is available, those clients that wait through multiple clustering attempts are likely to be the hardest to place (with relatively higher latency). These factors will typically lead to the creation of smaller-size groups. If larger groups are desired, Switchboard can be configured to reject groups of insufficient size.

Figure 24 shows the total amount of time a client spends in matchmaking, broken down by the measurement delay and the grouping delay. Clients are grouped using random buckets of 500 clients. Each bucket is processed in parallel by separate threads. A client may have to wait through multiple grouping attempts before one or more peer clients are ready with which it can be grouped

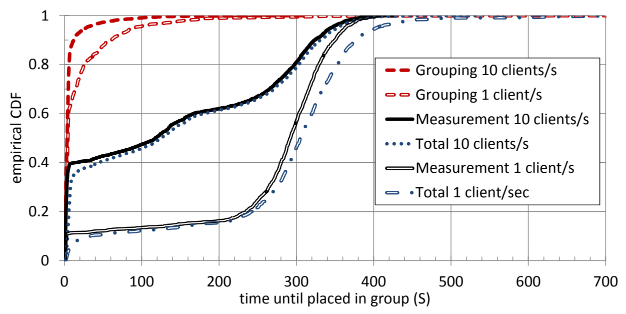

Figure 24: Client time spent in measurement and grouping. Measurement reflects the time from when a client joins a lobby until there is sufficient data for the client's tower. Time required for grouping reflects the total time from when measurement data is sufficient until the client is placed into a viable group (one or more clustering attempts). Grouping performed with randomized buckets of up to 500 clients.

(again, these results are not directly comparable to those in § 5.2 as client arrival rate and re-grouping contribute to grouping performance). Note that since higher arrival rates enable both greater measurement reuse and increase the pool of clients for clustering, these delays substantially reduce with more users.

5.4 Summary of Evaluation Results

Our implementation and evaluation confirm our intuitions for Switchboard performance, especially as function of scale. Switchboard's mechanisms to cluster groups by latency proximity are substantially more effective than geography-based techniques. Comparing QT and hierarchical clustering for group formation, we find that hierarchical clustering is more effective, creating similarly-sized groups to QT at a smaller computational delay. Finally, with increasing utilization, server bandwidth requirements scale sub-linearly, per-client probing overheads decrease, larger groups are formed, and client delays for measurement and grouping decrease.

6. CONCLUSIONS

Turn-based multiplayer games are available on multiple phone platforms and are popular. We want to enable fast-paced multiplayer games over cellular data networks. While 3G latencies can often be within the tolerance of some fast games [12], such games are not common because it is difficult for the game developer to deal with the highly variable nature of 3G latencies.

First, we demonstrate that P2P over 3G is a viable way to both reduce the latency of such games and the cost to the developer to maintain game servers. Depending on the number and location of servers, P2P can save as much as 148ms of median latency. Second, we have built Switchboard to reduce the burden on the game developer for managing this highly variable latency. It solves the matchmaking problem, or specifically assigning players to game sessions based on latency. Switchboard achieves scalability both in measurement overhead and in computation overhead. Based on experiments, we show that a small number of measurements in a 15 minute window sufficiently characterizes not only the latency of that phone, but also of other phones under the same celltower. This latency distribution is also highly predictive of the next 15 minutes. Using this information, Switchboard is able to significantly reduce the measurement overhead by coordinating across many phones. Switchboard then exploits the nature of this latency in a heuristic that quickly assigns players to game sessions.

This is a ripe new research area with many other open problems. Specifically in matchmaking, our work does not consider phones that are moving (*e.g.,* on a bus) – perhaps one can predict future celltowers (and hence future latency) by looking at the phone's trajectory. We do not attempt to estimate and predict bandwidth over 3G. We do not consider remaining energy in assigning measurement tasks to phones, or any other explicit form of fairness. There are interesting challenges in energy conservation during game play, and improving touch-based UI for fast action gaming.

Acknowledgments

We sincerely thank our shepherd, Iqbal Mohomed as well as the anonymous reviewers, for their invaluable feedback. Romit Roy Choudhury and Justin Manweiler were partially funded through NSF grant CNS-0916995.

7. REFERENCES

[1] Cell site. http://en.wikipedia.org/wiki/Cell_site#Range.
[2] Census 2000 U.S. Gazetteer Files. http://www.census.gov/geo/www/gazetteer/places2k.html.
[3] Clustering Analysis. http://en.wikipedia.org/wiki/Cluster_analysis.
[4] Federal Communications Commission Geographic Information Systems. http://wireless.fcc.gov/geographic/index.htm.
[5] S. Agarwal and J. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *SIGCOMM*, 2009.
[6] AT&T Wireless. Wireless IP options for mobile deployments. In *www.wireless.att.com*, Dec. 2010.
[7] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian. Where's that phone?: Geolocating IP addresses on 3G networks. In *IMC (short paper)*, 2009.
[8] N. Baughman and B. Levine. Cheat-proof playout for centralized and distributed online games. In *INFOCOM*, 2001.
[9] Y. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, 2001.
[10] M. C. Chan and R. Ramjee. TCP/IP performance over 3G wireless links with rate and delay variation. In *ACM MobiCom*, 2002.
[11] C. L. T. Chen. Distributed collision detection and resolution. In *McGill University masters thesis*, May 2010.
[12] M. Claypool and K. Claypool. Latency and player actions in online games. In *Communications of the ACM*, Nov. 2006.
[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *SIGCOMM*, 2004.
[14] P. Deshpande, X. Hou, and S. R. Das. Performance comparison of 3G and metro-scale WiFi for vehicular network access. In *IMC*, 2010.
[15] S. Gargolinksi, C. St.Pierre, and M. Claypool. Game server selection for multiple players. In *NetGames*, 2005.
[16] Gartner. Gartner says worldwide mobile gaming revenue to grow 19 percent in 2010. In *www.gartner.com*, May 2010.
[17] Halo 3 Forum. Average multiplayer game length. In *www.bungie.net*, Feb. 2010.
[18] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning (2nd ed.). In *Springer*, 2009.
[19] L. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. In *Genome Research*, 1999.
[20] H. Holma and A. Toskala. WCDMA for UMTS – HSPA evolution and LTE. In *WILEY*, Fourth Edition 2008.
[21] J. Huang, Q. Xu, B. Tiwana, Z. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *ACM MobiSys*, 2010.
[22] F. Kammer, T. Tholey, and H. Voepel. Approximation algorithms for intersection graphs. In *Approximation, Randomization and Combinatorial Optimization Algorithms and Techniques*, 2010.
[23] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. In *NSDI*, 2007.
[24] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang. Experiences in a 3G network: interplay between the wireless channel and applications. In *ACM MobiCom*, 2008.
[25] M.R.Garey and D.S.Johnson. Computers and intractability. In *W.H.Freeman and Company*, 1979.
[26] M. Stanley. The mobile Internet report. Dec. 2009.
[27] J. Stokes. Primal rage: a conversation with Carmack, and a look at id's latest. In *ars technica*, Nov. 2010.
[28] W. Tan, F. Lam, and W. Lau. An empirical study on 3G network capacity and performance. In *INFOCOM*, 2007.
[29] M. Whitten. An open letter from Xbox LIVE general manager Marc Whitten. In *www.xbox.com*, Feb. 2010.