When Multi-touch Meets Streaming

Zimu Liu
Department of Electrical and Computer Engineering
University of Toronto
zimu@eecg.toronto.edu

Yuan Feng
Department of Electrical and Computer Engineering
University of Toronto
yfeng@eecg.toronto.edu

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
bli@eecg.toronto.edu

ABSTRACT

With the advent of mobile devices with large displays, it is intuitive and natural for users to interact with an application on a mobile device using multi-touch gestures. In this paper, we propose that these multi-touch gestures can be *streamed* on-the-fly among multiple participating users, making it possible to engage users in a collaborative or competitive experience. Such *multi-touch streams*, featuring very low streaming bit rates, can be rendered on receivers to precisely reconstruct the states of an application. We present the challenges, system framework, embedded algorithm design, and real-world evaluation of *TouchTime*, a new system that has been designed from scratch to facilitate the streaming of multi-touch gestures among multiple users. By seamlessly combining local computation on mobile devices and services from the "cloud," we explore the design space of suitable mechanisms to represent and packetize multi-touch gestures, and of practical protocols to transport concurrent live multi-touch streams over the Internet. Specifically, we propose an auction-based reflector selection algorithm to achieve the minimal end-to-end delay in a live multi-touch streaming session. To demonstrate *TouchTime*, we have developed a new real-world music composition application — called *MusicScore* — using the Apple iPad Programming SDK, and used it as our running example and experimental testbed to evaluate our design choices and implementation of *TouchTime*.

Categories and Subject Descriptors

H.5 [**Information Interfaces and Presentation**]: User Interfaces and Presentation; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; D.2.13 [**Software Engineering**]: Reusable Software—*Reusable Libraries*

General Terms

Algorithms, Design, Measurement

Keywords

Multi-touch Streaming, Reflector Selection, Mobile Framework

MUM'11 December 7–9, 2011, Beijing, China

1. INTRODUCTION

With *multi-touch*, users interact with computing devices by applying elaborate finger gestures, directly on display surfaces. Since the advent of the iPhone in 2007, there has been a steady trend of adopting multi-touch as the primary human-computer interface in mobile devices. It is expected that it could one day free us from the mouse as our primary way of interacting with computing devices, the way that mice freed us from keyboards [2].

As larger multi-touch surfaces are introduced in recent products, including the Apple iPad and Microsoft Surface, we believe that multi-touch, as a more natural and intuitive interface, will soon be widely used in a new array of applications [18]. Imagine a professional photographer editing his photos in the field; a designer showing her clients an illustration sketch on site; or a child entertained with an interaction intensive game in the family room—all with only finger gestures. In addition, features of these applications can be substantially enhanced if more than one user can be allowed to use them simultaneously. Imagine, again, the level of satisfaction if a photographer can interact with his team members live while editing photos; if a designer can show the illustration sketch to her clients in a remote location; if the entire family can collaborate or compete in the same game in the family room; or if people anywhere in the world can watch a live chess game between two master players in real time.

To realize these scenarios, applications need to share *live streams* of screen snapshots or application-specific data to one or multiple receivers. In a game, for example, this can be readily achieved by streaming game scenes or data of avatars as they move. However, there are two major disadvantages in such solutions. *First,* streaming live videos may not be a scalable solution to a large number of receivers, as it requires a substantial amount of bandwidth resources. Due to the best-effort nature of the Internet and the higher bit rate of the video stream, there are no guarantees of the media quality when the video is received and played back, especially with a large number of participants. To mitigate such adverse effects, lower bit rates and receiver-side buffering are usually used for video, degrading its quality and timeliness. *Second,* even if more condensed application-specific data is streamed instead of video, such a solution requires each application to design a *unique custom-tailored solution* for sharing with other users [1], which cannot be readily used in a different application.

To our knowledge, we are the first to propose that a much simpler and more elegant solution exists for the entire category of multi-touch applications, when users use multi-touch finger gestures to interact with them. To state the solution succinctly, we believe that it is feasible to *stream multi-touch finger gestures* directly to multiple receivers, each of which is represented by a live instance of the same application, but on a different computing device, such as a

different iPad. Rather than playing back a live video stream, multi-touch gestures can be precisely rendered on a receiver to affect the states of an application. Rather than being a bandwidth-intensive streaming solution, multi-touch gestures can be streamed live with a very low bit rate, and as such can be exceedingly scalable. Rather than being a custom-tailored application-specific solution, our solution can be implemented as a library or plug-in that works well with any multi-touch application that needs live sharing features.

In this paper, we start with outlining the fundamental design objectives and challenges when multi-touch gestures are to be streamed live to multiple receivers. We devote the core of this paper to address these challenges with the design of a new system from scratch to stream multi-touch gestures, referred to as *TouchTime*, from metadata presentation to practical protocols to deliver them over the Internet. Once these multi-touch gestures are streamed live to another user, it will be replayed in the same application, as if an "invisible" hand is touching her display. As we design *TouchTime*, we strive to cover the entire spectrum of questions to make this happen: how gestures are presented, packetized, and then transported over a best-effort session to any receiver in the Internet.

A highlight of our design is the integration of local presentation of multi-touch gestures on mobile devices and optimized transport services in the "cloud." Our design philosophy in *TouchTime* is to be simple yet elegant. To facilitate the all-to-all broadcast transmission of multi-touch gestures among all players, we propose to implement an auction-based reflector selection algorithm. By expressing users' and reflector cloud's preferences using *bids* and *asks* in a *continuous double auction market*, the auction mechanism that intends to optimize all players' utilities results in the optimal matching between users and reflectors such that the conflicting objectives of minimizing delays and balancing server load are resolved.

The remainder of this paper is organized as follows. After presenting our design objectives and challenges in Sec. 2, we describe the architectural design and system framework of *TouchTime* in Sec. 3, from the perspective of mobile devices. In Sec. 4, we shift our focus to the "cloud," and explore how multi-touch gestures can be streamed in a secure and practical fashion, accommodating fluctuating delays that mobile users tend to experience. Sec. 5 presents our *MusicScore* application, and evaluates the effectiveness of *TouchTime* when it is used for live multi-touch streaming in *MusicScore*. We conclude the paper with a discussion of related work and final remarks in Sec. 6 and Sec. 7, respectively.

2. MOTIVATION AND CHALLENGES

Our ultimate design objective is to design the best possible system, called *TouchTime*, from the ground up to stream multi-touch gestures to multiple participating users. The *TouchTime* system uses a shared set of components to support a wide variety of touch-intensive applications.

Throughout this paper, as a running example of such an application and an experimental testbed for *TouchTime*, we have designed and implemented *MusicScore*, an application for music composition using multi-touch gestures, from scratch using the iPad SDK. *MusicScore* allows a user to create musical notes by taps with fingers, to change the pitch of notes in a chord by dragging them vertically with two or more fingers, to choose a group of musical symbols by creating a selection box with two fingers, and to remove selected notes by crossing them out. Fig. 1 visually illustrates a conceptual example: when the touching finger moves up, the pitch of the selected half note changes from D to G.

2.1 Streaming Multi-touch Gestures

There are many benefits we can obtain by streaming multi-touch gestures directly rather than live screencast or application-specific data. Compared with the traditional live video streaming, transmitting gestures incurs much fewer streaming bit rates, which alleviates the bandwidth requirement substantially. With an intelligent design, streaming multi-touch gestures rather than video is more likely to satisfy stringent delay requirements. More importantly, multiple users are now possible to interact with the same set of application states at the same time. In *MusicScore*, this implies that one user in New York is able to compose the theme of a piece of music, while another user in Tokyo is working on the chords of the same piece at the same time. Note that as voices in a score are edited by each individual collaborator, the editing conflict problem, which many online collaborative editing systems face [9], is naturally solved.

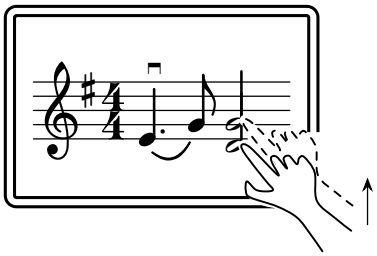

Figure 1: Music composition with multi-touch gestures in *MusicScore*, a music composition application we developed on the iPad.

Since the goal of *TouchTime* is to share collaborative or competitive multi-touch interactions among mobile users in the same session, *TouchTime* should be able to stream multi-touch gestures generated from each participant, replay streamed gestures at remote mobile applications, and change application states accordingly in remote devices. In Fig. 2, we show a conceptual illustration of multi-touch streaming in *TouchTime*. User A's multi-touch gestures (using two-finger to create a selection box in this example) are represented by a multi-touch stream, and transmitted to User B for replay. As if an "invisible" hand is touching B's display, the application states of B is changed correspondingly.

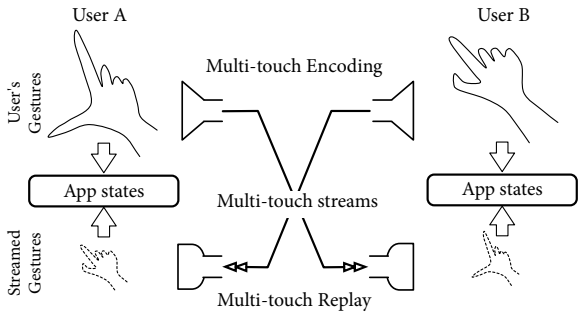

Figure 2: A conceptual illustration of multi-touch streaming in *TouchTime*.

2.2 Characteristics of Multi-touch Streaming

There are a number of challenges when designing the *TouchTime* system, as its design is custom-tailored to fit the needs of streaming multi-touch gestures, a non-conventional kind of data with its unique characteristics.

First, multi-touch gesture streams have a *very low*, yet *bursty*, bit rate. In multi-touch applications, it is usually the case that users

interact with their devices frequently for a while and then stay idle most of the time. *Second,* multi-touch gestures need to be streamed in an *in-order* and *lossless* fashion. Reliability is of utmost importance when streaming gestures: any lost or erroneously transmitted gesture shall lead to a complete failure, as these gestures are used to precisely render and reconstruct application states at the receiver. *Third*, multi-touch streaming prefers the shortest possible end-to-end delay. Most collaborative multi-touch applications are time sensitive, and as such demand a short delay when multi-touch gestures are to be streamed. *Finally* and most importantly, once the playback of streamed gestures starts at a receiver, no matter how long the initial startup delay is, the interval between the playback time of two gestures at the receiver has to be kept precisely identical to the difference between their original timestamps when they are generated at the sender. Otherwise, rendered states of an application may be different from the original. This implies that each gesture has to arrive at the receiver *on time*, before its scheduled rendering time. A longer initial startup delay may be used to mask higher delay jitters, so that most streamed gestures can be rendered on time. Illustrated in Fig. 3, Bob's replay of Alice's gestures can be later than the original ones by an initial startup delay δ, but the intervals between gestures $\Delta t_1, \Delta t_2 ...$ have to be kept precisely the same during playback.

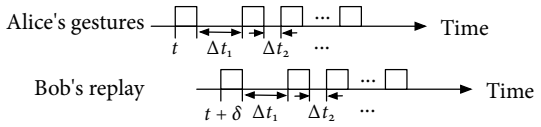

Figure 3: Streaming multi-touch gestures with an initial startup delay of δ.

With these unique characteristics, the design of the *TouchTime* is more challenging than conventional media streaming systems. It needs to be designed so that a bursty and low bit-rate stream from each user can be transmitted in a reliable and timely fashion.

3. TOUCHTIME: WHEN MULTI-TOUCH MEETS STREAMING

We are now ready to present a detailed design of the *TouchTime* system.

3.1 TouchTime: An Overview

We have implemented the *TouchTime* system on the iOS platform, using Objective-C, with support for iOS multi-touch devices such as the iPad. *TouchTime* is designed to provide a complete system framework to support multi-touch gesture streaming in touch-intensive applications, such as *MusicScore*, without requiring application specific development efforts.

Fig. 4 demonstrates the architecture of *TouchTime*. It consists of three components: a *presentation* module, which observes all multi-touch events generated by users and packetizes these events into multi-touch objects for transmission; a *transport* module, which efficiently streams multi-touch objects to one or more devices; and a *replay* engine that renders remote multi-touch objects in the local application context.

3.2 Presenting Multi-touch Gestures

The first natural question is how multi-touch gestures should be presented and packetized in preparation for streaming to multiple participating users.

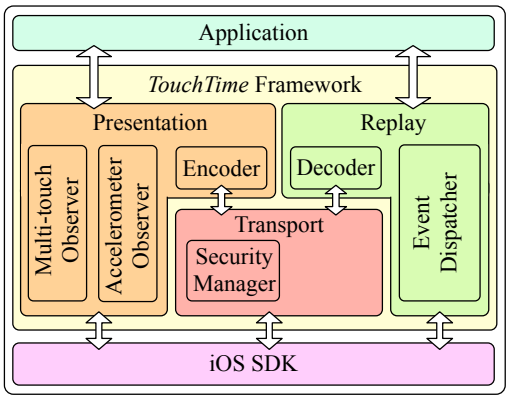

Figure 4: Architecture of software components in *TouchTime*.

In multi-touch applications, *gesture recognizers* are developed to analyze a *raw stream* of touch objects in a multi-touch sequence, and to determine the intention of users based on properties of each gesture. For example, the new gesture recognizer in the recently released iPad Programming SDK facilitates the recognition of basic gestures, such as tap, pinch, and swipe, which are used in *MusicScore* to interact with music elements. Fig. 5 demonstrates a few lines of code to use a gesture recognizer in the iOS SDK to detect an upward swipe gesture with two fingers. It analyzes the number of touches and the direction of touch movement from the raw stream, and compares them with requirements stored in the recognizer to make a decision.

```
// Create & initialize a swipe gesture recognizer
UISwipeGestureRecognizer *upSwipe =
    [[UISwipeGestureRecognizer alloc]
        initWithTarget:self
        action:@selector(handleUpwardSwipe:)];
upSwipe.numberOfTouchesRequired = 3;
upSwipe.direction = UISwipeGestureRecognizerDirectionUp;
// Associate the recognizer with a view
[aView addGestureRecognizer:upSwipe];
```

Figure 5: An example of using the three-finger upwards swipe gesture recognizer in iOS SDK.

What information should be streamed for a precise playback at a receiver? We have two feasible alternatives, as shown in Fig. 6 and Fig. 7. *First,* the *raw stream* of touch objects, represented by a successive sequence of touch locations, can be streamed to the receiver, which uses local recognizers to distill multi-touch gestures. This alternative is suitable for applications that engage touch input without the need for recognizing complex gestures, such as artistic drawing. *Second,* multi-touch gestures can be first recognized and then streamed to the receiver to be replayed so that application states are rendered. In this case, the sender needs to transmit not only the type of the gesture, but also additional *metadata* of the gesture so that it can be replayed precisely. Such metadata is specific to a gesture, but at least includes the location of the gesture in the current view.

In addition to multi-touch gestures, the accelerometer sensor, designed for motion input, is often used for user interface control, especially in action-intensive games. To render application states at the receiver, it is necessary to stream raw three-dimensional (x, y, z) values and their timestamps, stored in `UIAcceleration` objects in the iPad SDK. The time interval of streaming these three-

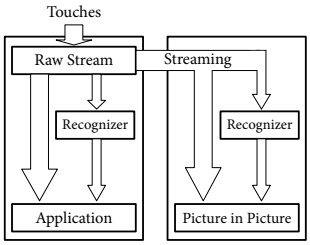

Figure 6: Streaming a *raw stream* of touch objects.

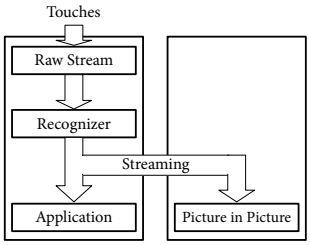

Figure 7: Streaming multi-touch gestures.

```
<?xml version = "1.0"?>
  <gesture>
    <timestamp>
      2011-05-19T14:34:53.21 UTC
    <\timestamp>
    <sequence-number> 246 <\sequence-number>
    <type> tapping <\type>
    <metadata>
      <location> (345, 253) <\location>
      <number-of-tap> 1 <\number-of-tap>
      <number-of-finger> 2 <\number-of-finger>
    <\metadata>
  <\gesture>
```

Figure 8: Presenting a two-finger single-tap gesture with XML.

dimensional values should be the same as the *update interval* specified in the `UIAccelerometer` class in the SDK, and is specific to an application. Being periodic, these values can also used as *soft states* that maintain the liveness of the stream and keep the connection refreshed: it is easy to detect the difference between a terminated stream and an idle stream in *TouchTime*, by checking if periodic accelerometer values are still being received at a receiver.

To packetize and transmit a *raw stream* of touch objects (Fig. 6), we propose to use a simple binary format, due to the fact that there are only four types of events involved (touch-begin, touch-move, touch-end and touch-cancel). With a raw stream, touch objects are continuously generated as a user interacts with her device, and as such they require a more compact form of presentation in preparation for transmission with a low bit rate. Though the binary form of presentation is terse, it is not easily extensible to accommodate a wide variety of new multi-touch gestures. For this reason, if multi-touch gestures are to be streamed instead (Fig. 7), we propose to use XML in *TouchTime*. XML is designed to represent complex and structured data, and frees us from worrying about the basic syntax of how to represent multi-touch gestures to be streamed. It can be used to describe new multi-touch gestures, without the need to write custom-designed parsers. As gestures are entered by a user at a very low rate, the bit rate of the stream remains low and is not a main concern when XML is used.

There are a number of possible *events* to be streamed: touch objects in a raw stream, multi-touch gestures, as well as periodically updated accelerometer values. Regardless of the alternative of multi-touch streaming used (raw stream or gestures), each of these events should always be accompanied by its *sequence number* and *timestamp*. At a receiver, sequence numbers are used to detect out-of-order delivery and losses of the event stream, and timestamps help to replay them with precise time intervals as they are originally generated: any time interval Δt_i can be computed as the difference between the timestamp of the i^{th} event and that of the $(i-1)^{th}$. An example of presenting a typical multi-touch gesture using XML is shown in Fig. 8.

3.3 Replaying Multi-touch Objects at Receivers

Once packetized multi-touch objects are received, they are locally rendered by the playback module in *TouchTime*. Multi-touch objects are first encapsulated in `UITouch` instances, used internally to represent touch gestures in the iOS. With respect to accelerometer information, `UIAccelerometer` instances are restored using received data in the stream. As `UITouch` or `UIAccelerometer` instances have locally been created, *TouchTime* embeds them into `UIEvent` objects and push these events into the event dispatching queue. The iOS will then automatically propagate events through the responder chain and trigger corresponding methods to handle these events, as if they have been entered locally.

4. TOUCHTIME: STREAMING MULTI-TOUCH IN THE CLOUD

In this section, we present the design of *TouchTime* in the cloud, and show how a number of *TouchTime* system components collaborate to provide cloud services for multi-touch streaming.

4.1 The Need for the Cloud

After users established a live multi-party interaction session, it is up to *TouchTime* to relay data traffic in such a session. Intuitively, forming a complete mesh topology among users in the session may be a solution. However, it is not scalable, bandwidth-efficient, or energy-efficient. As an alternative solution, bandwidth usage within a complete mesh can be reduced by nominating a device as a session "host," relaying streams for others. However, power consumption on such a host with inevitably increase, which may not be fair to the host.

Furthermore, most multi-touch devices in reality are mobile devices equipped with EDGE, 3G and/or Wi-Fi modules. In these cellular and wireless networks, network address translation (NAT) is extensively used to bridge private local networks with the public Internet. As NAT breaks end-to-end connectivity, it is difficult to establish a direct connection between two multi-touch devices, even with modern NAT traversal technologies. To overcome this critical problem, we design and implement the *TouchTime* reflector in the cloud to relay multi-touch streams. If two nodes in a session fail to initiate a direct connection between each other, they can take advantage of the *TouchTime* cloud service for multi-touch streaming, and actively connect to a common set of nearby reflectors. Then, multi-touch gestures from the sending node can flow through encrypted tunnels created by *TouchTime* reflectors, and reach the receiving node in the session. A high-level overview of the *TouchTime* cloud is illustrated in Fig. 9.

To build a secure cloud service for *TouchTime*, we propose to build a *TouchTime* controller — a set of dedicated *authentication* and *authorization* servers — to manage user nodes and services in the cloud. Each time a node is logged in and a collaboration session is initiated, the *TouchTime* controller verifies the user's credentials, such as an email and password pair, and grant the node access privileges to transmit its multi-touch stream over reflectors in the cloud. Communications between the *TouchTime* controller and users are secured by the HTTPS protocol. The use of the HTTPS protocol also maximizes compatibility with restricted network connections (*e.g.*, firewalls, proxies, and NAT gateways). Additionally, the *TouchTime* controller issues a digital certificate for each active *TouchTime* session, which will be verified by *TouchTime* reflectors as the forwarding connection is being established. Since malicious nodes cannot join a session in reflectors without a valid certificate,

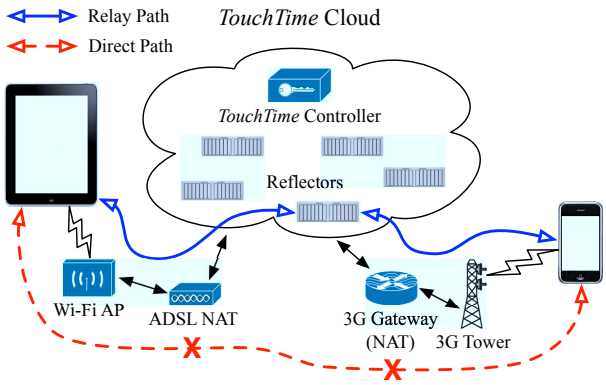

Figure 9: Cloud services in the *TouchTime* system. End-to-end connectivity (shown in dashed line) is disrupted by NAT in a Wi-Fi network and a 3G gateway in cellular network.

wiretapping and destructive manipulation are effectively prevented in the cloud.

In order to implement relay tunnels that are to be established between *TouchTime* reflectors and end users, we choose to use the TLS protocol over TCP connections. The benefits of TLS over TCP are two-fold: (1) TCP naturally provides an in-order, lossless and error-free transport service required by multi-touch streaming; (2) TLS gives an off-the-shelf security solution to ensure the secrecy and integrity of streamed data. Once participating nodes in a session have securely connected to a common set of reflectors, these nodes can freely transmit multi-touch streams over the *TouchTime* cloud. Taking full advantage of event-driven sockets brought by `NSStream` and the *Run Loop* infrastructure in the Cocoa framework, our implementation of *TouchTime* reflectors is able to maximize CPU utilization in each server, and as a result achieve maximum throughput with respect to the relay service provided to multi-touch streams.

4.2 The Problem of Reflector Selection

TouchTime allows applications to implement custom-tailored reflector selection algorithms, by invoking methods in application-provided *delegate* instances. An application with delay-sensitive sessions may wish to favour reflectors with low latencies, while data-intensive interactive sessions may prefer reflectors with higher throughput and lighter workload. That said, we wish to ship *TouchTime* with a number of reflector selection algorithms that are designed to minimize end-to-end delays, which is one of the most important concerns in multi-touch streaming.

Since different users experience different end-to-end delays when using the same reflector, their intrinsic value of using this reflector is naturally different. As end-to-end delays fluctuate over time, it is reasonable to allow users to choose multiple reflectors. Reflectors, provided by dedicated servers in the cloud, also have the capacity to assist multiple users. The relationship between reflectors and users can then be modeled with a *bipartite graph* illustrated in Fig. 10. If a reflector (on the left) is selected to provide relay services to a user (on the right), they are linked with a darkened edge. The preferred-reflector graph, marked with darkened edges, is therefore a subgraph of the bipartite graph. The problem of reflector selection is essentially a matching problem that produces such a preferred-reflector graph, in which each user is matched to one or more reflectors, and each reflector is matched to a number of users, without exceeding its relay capacity.

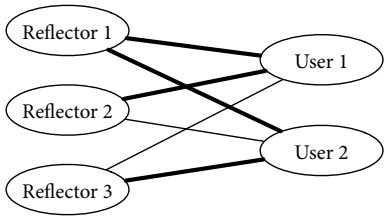

Figure 10: The problem of reflector selection can be modeled as a matching problem in a bipartite graph.

Greedy reflector selection. Provided that TLS connections to nearby reflectors have been established, *TouchTime* should intelligently decide how to transmit multi-touch streams over these reflectors. In order to minimize end-to-end delays — the most critical design objective of *TouchTime* — an intuitive way is that each user *greedily* selects the reflector that provides a relay path with the shortest end-to-end delay. To mitigate delay variations over time, the greedy algorithm can become more dynamic: although only one relay path is chosen to be active, nodes keep other idle paths alive, *i.e.*, idle tunnels are not disconnected, and delay measurements are periodically probed over these idle paths. Once a user finds a "better" reflector that forms a relay path with a shorter delay, it will immediately switch to that reflector.

With the greedy algorithm, however, the load of reflectors may fluctuate over time. Since users are allowed to dynamically switch to "better" reflectors, the load may be shifted back and forth between two nearby reflectors over time. Since a user chooses a reflector with the shortest delay in a greedy fashion, the reflector with the shortest delay will be chosen by a large number of users at a given time with high probability. Such a "flash crowd" of relay requests leads to a substantial increase in the load on the reflector, and a longer processing time due to the high load results in longer relay latencies. As a consequence, users tend to switch to a nearby reflector with a smaller delay, shifting the load to that reflector.

4.3 Auction-Based Reflector Selection

Taking the phenomenon of load shifting into account, it is necessary to design and implement a more elaborate algorithm to select reflectors in the *TouchTime* cloud in order to achieve better performance with respect to delays. Such a new algorithm should resolve the conflicting interests between users, who prefer the shortest delays when selecting a reflector, and the *TouchTime* cloud, who prefers a more balanced load across reflectors. From the perspective of the entire *TouchTime* system, a "pivot point" needs to be chosen to achieve a balanced tradeoff.

Since the delay variation of a single relay path trends to be relatively large, to meet stringent delay requirements, in addition to periodically probe for a reflector with shorter delays, we may send a few replicas of the same packet through *two or more* reflectors, simultaneously. In this case, the delay of one packet is determined by the shortest delay among all relay paths: it is likely that delays and delay variations are reduced with the use of multiple reflectors.

That said, the use of multiple copies of the same packet leads to redundancy, which should not be excessive. Multi-touch gesture streams in *TouchTime* have a *very low*, yet *bursty*, bit rate. Our preliminary measurements reveal that most of time (over 60%) the bit rate is lower than 5 kbps, and occasionally the rate is over 10 kbps. Considering these observations, a few replicas of the same packet are considered acceptable when the bit rate is lower than 10 kbps. The challenge, of course, is how a small number of reflectors can be selected to serve each user.

Due to the delicate relation between delays and the reflector load, it now becomes necessary to design and implement a more elaborate algorithm to match reflectors in the reflector cloud to users in order to achieve better performance with respect to delays. To avoid load shifting when the greedy selection algorithm is used, it will be beneficial if reflectors can give some "feedback" on their load to the users; and more importantly, such "feedback" should be given in a *proactive* fashion, such that users can be "warned" by potentially longer processing times *before* they actually connect to that reflector.

It has been well known that *double auctions*, such as the Preston-McAfee double auction [16], the threshold price double auction [14], and the continuous double auction [10], are widely applied in the research of networking, such as resource allocation in grid computing [3], spectrum trading in cognitive radio [21] and so on. It solves the matching problem of a set of commodities between multiple buyers and sellers, such that the system welfare gained by all players is maximized over a given player-commodity payoff function.

The problem of matching users to reflectors can be related to such a double auction market, in which all participating users behave as *buyers*, all reflectors behave as *sellers*, and relay services are treated as *commodities*. If we define the payoff function of a user connected to a reflector to be a non-increasing function of the delay of using its relay service (*i.e.*, the smaller the delay is, the greater the payoff will be), the matching results arbitrated by the continuous double auction market naturally produce the optimal selection outcome that minimizes end-to-end delays.

The system welfare problem. We consider a system of R reflectors in the cloud and U participating users, represented by sets $\mathcal{R}$ and $\mathcal{U}$, respectively. At any given point in time, we define $\vec{I_u} = (I_u^1, I_u^2, ..., I_u^R)$ to be the configuration for a user u. I_u^r is a binary variable that indicates whether a reflector r helps relay packets from user u, and has the following form:

$$I_u^r = \begin{cases} 1 & r \text{ is assigned to relay packets from } u; \\ 0 & \text{otherwise.} \end{cases}$$

Any reflector $r \in \mathcal{R}$ has a maximum relay capacity C_r, in terms of the number of users it is able to serve. Valid configurations are then constrained by the following inequality:

$$\sum_{u \in \mathcal{U}} I_u^r \leq C_r, \forall r \in \mathcal{R}.$$

We assume that at each user u there is an underlying utility function $F_u(\vec{I_u})$, which reflects how user u is satisfied with configuration $\vec{I_u}$. The system welfare problem — with the objective of optimizing the aggregate utility of users subject to relay capacity constraints of reflectors — can be formulated as follows:

$$\max_{\vec{I_u}} \quad \sum_{u \in \mathcal{U}} F_u(\vec{I_u}) \tag{1}$$

$$\text{s.t.} \quad \sum_{u \in \mathcal{U}} I_u^r \leq C_r, \forall r \in \mathcal{R}. \tag{2}$$

The system welfare problem aims to find the *optimal* configuration $\vec{I_u}$ for all users so that the sum of their utilities is maximized. The approach of defining the system welfare as the sum of individual utilities is widely applied in resource allocation problems in networking [11, 12]. Since our objective is to minimize the delay experienced by every user, it is reasonable to assume that the utility is higher when a configuration assigns reflectors with smaller delays. We proceed to show that, by defining the utility function appropriately, the system welfare problem is equivalent to the social utility maximization problem in a *continuous double auction* market.

The reflector selection market. Our motivation of using double auction markets to solve the reflector selection problem is inspired by the power of markets arbitrating optimal decision of both buyers and sellers in a balanced fashion. Auctions have the ability to determine the optimal buyer-seller matching when considering all participants' utilities. As a consequence, the resulted match of users and reflectors takes into account users' preference of shortest delays and *TouchTime* cloud's preference of a more balanced load across reflectors at the same time.

Though there are numbers of auction mechanisms, the continuous double auction has a number of unique properties that is amenable to a practical and decentralized implementation in cloud systems. *First*, the auctioneer's role is much less emphasized and, apart from certain monitoring tasks such as assuring that offered prices are taken without further bargaining between traders, it can be reduced to collecting the bids and informing traders [8]. *Second*, the continuity in the market caters to asynchronously arrived bids and asks. Users are, of course, inherently asynchronous in the *TouchTime* cloud service. In a continuous double auction market, buyers and sellers submit their *bids* and *asks*, respectively, at any time during the trading period, and the market clears continuously. Specifically, a *trade* is executed immediately when a bid exceeds an ask.

Without loss of generality, we focus on one double auction at an arbitrary trading time. The bid (ask) price $b_u^r(a_r^u)$ is the reported price that buyer u (seller r) is willing to trade with seller r (buyer u). The clearing price p_u^r is the transaction price at which the winning buyer u and seller r trade. A buyer who buys her commodity valued at b and pays p to receive it will obtain a utility gain of $b-p$; a buyer who pays nothing and not receiving the commodity obtains zero utility. Similarly, a seller who sells her commodity valued at a and receives a payment of p obtains a utility of $p-a$; otherwise obtains zero if no trade arises. If all participants bid truthfully, the double auction protocol in a market with M buyers and N sellers solves the following social utility maximization problem [4]:

$$\max \quad \sum_{u=1}^{M}\sum_{r=1}^{N} I_u^r(b_u^r - p_u^r) + \sum_{r=1}^{N}\sum_{u=1}^{M} I_u^r(p_u^r - a_r^u) \tag{3}$$

$$\text{s.t.} \quad \sum_{u=1}^{M} I_u^r \leq C_r, \forall r \tag{4}$$

$$I_u^r \in \{0, 1\},$$

where I_u^r denotes whether a trade is conducted between buyer u and seller r or not, and constraint (4) refers to the reality that the number of trades conducted by each seller r is restricted by its commodity capacity C_r.

By comparing the system welfare problem (1) with the social utility maximization problem (3), it is not difficult to find out that the two problems are equivalent to each other if we define the utility function to be of the following form:

$$\begin{aligned} F_u(\vec{I_u}) &= \sum_{r \in \mathcal{R}} I_u^r(b_u^r - p_u^r) + \sum_{r \in \mathcal{R}} I_u^r(p_u^r - a_r^u) \\ &= \sum_{r \in \mathcal{R}} I_u^r(b_u^r - a_r^u). \end{aligned}$$

If bids b_u^r and asks a_r^u are properly defined so that the utility function $F_u(\vec{I_u})$ is able to specify the satisfaction of user u under the configuration $\vec{I_u}$, the double auction can be applied to solve the system welfare problem in *TouchTime*.

Reflector selection in a double auction market. We now envi-

sion the existence of an online double auction market, where all participating users behave as *buyers* and reflectors behave as *sellers*; services of relaying packets in *TouchTime* are treated as *commodities*. In a continuous double auction, users are matched to reflectors such that the combined satisfaction level of all users is maximized.

Every user u bids b_u^r for the relay service of each reflector r according to the end-to-end delay measured by the user herself, and adjusts her bid accordingly as delays vary over time. Every reflector r, in turn, asks a_r from all users depending on its CPU utilization, since a saturated utilization of the CPU reflects that the relay capacity has been reached. As a consequence, mappings between buyers and sellers are dynamically controlled by both users and reflectors. Note that the ask of one reflector to every user is the same, *i.e.*, $a_r^u = a_r, \forall u \in \mathcal{U}$, since reflectors do not treat any of the users with higher priorities.

Let t_u^r be the end-to-end delay when packets are relayed via a reflector r, measured by user u. To express her preference, each user u in the market bids b_u^r for the relay service of each reflector r according to the end-to-end delay t_u^r measured by the user herself, and adjust her bid accordingly as delays vary over time. We define the bid from user u to reflector r as $b_u^r = \exp\left(-\alpha(t_u^r)^\beta\right)$, where α and β are variables that can be tuned according to delay observations. Fig. 11 shows an example when the tolerable delay is smaller than 100 msec. As shown, the idea is when delays are smaller than 100 msec, the function generates bids that are high in general, since any reflector that can provide tolerable delays are preferred. On the other hand, if the measured delay is larger than 100 msec, the generated bid decreases dramatically until it reaches some maximum delay it can sustain, *e.g.* 300 msec in this example, when the bid goes to zero afterwards. This implies that the user is no longer willing to bid for reflectors with longer delays.

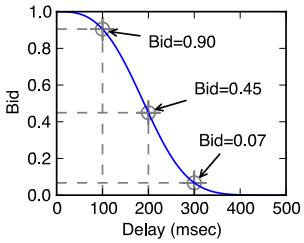

Figure 11: An example of the bid function with $\alpha = 10^{-7}$ and $\beta = 3$.

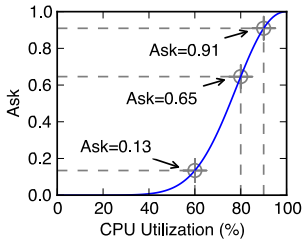

Figure 12: An example of the ask function with $p = 6\times10^{-4}$ and $q = 2.2$.

Let l_r and $\hat{l}_r$ be the average processing time and the threshold of average processing time at reflector r, respectively. The CPU utilization ratio of the reflector r can be expressed as $w_r = l_r/\hat{l}_r \times 100$. Similar to the bid, we define the ask from reflector r to be $a_r = \exp\left(-p(100 - w_r)^q\right)$. Fig. 12 shows an example with $p = 10^{-7}$ and $q = 3$. The intuition is that reflectors tend to decline additional connections when their load goes higher, *e.g.*, 60% in this example, by raising their asks accordingly. Note that the ask of one reflector to every user is the same, since reflectors treat all users with equal preferences. To make bids and asks comparable, we choose the exponential functions to compute bids and asks, so that the produced values are in the range of $[0, 1]$.

In continuous double auctions, the payoff of a trading pair with bid b_u^r and ask a_r is $b_u^r - a_r$. As we can see: the shorter the end-to-end delay t_u^r is, the larger the payoff will be; the lower the load l_r is, the larger the payoff will be, which implies a higher utility F_u. This conforms with the intuition that the shorter the delay, the more satisfied a user is. In addition, our definition of bids and asks also implies that a user would be more satisfied when it chooses *more than one reflector*.

Till this point, we have shown that our definition of bids b_u^r and asks a_r allows the utility function to specify the satisfaction of each user, which implies that the system welfare problem in *TouchTime* is equivalent to the social utility maximization problem in our double auction market. Initially, a user are grant access privileges by the *TouchTime* controller to establish TLS tunnels to all reflectors, and randomly chooses k (k equals to 2 in our implementation) reflectors that have the shortest delays. A user starts a continuous double auction (CDA) market periodically (the period $T = 3$ min in our implementation) by contacting reflectors directly. All reflectors then send their asks to this user, and the market is cleared gradually by the user's decision after comparing each ask to its bid, based solely on local information.

The auction-based reflector selection algorithm is described in **Algorithm 1**.

Algorithm 1 *Auction-Based Reflector Selection*.

1: A user u contacts reflectors to start a CDA market after every period of T.
2: Every reflector r computes its ask $a_r = \exp\left(-p(100 - w_r)^q\right)$, and sends it to user u after a TLS tunnel is established.
3: User u computes its bid b_u^r to reflector r: $b_u^r = \exp\left(-\alpha(t_u^r)^\beta\right)$.
4: When user u receives an ask a_r from a reflector r, it executes the following:
5: **if** $a_r \leq b_u^r$ **then**
6: User u notifies reflector r that they can trade at the trading price $p_u^r = \frac{1}{2}(b_u^r + a_r)$, *i.e.*, reflector r will serve to relay packets from user u.
7: **else**
8: No trade occurs.
9: **end if**
10: The market is cleared gradually by receiving asks a_r, and is closed when all asks have been received by user u.

Economic properties of double auctions. Truthfulness, individual rationality and budget balance are critical properties required to design economically robust double auctions [13]. We now discuss these properties in our double auction markets in *TouchTime*.

▷ *Truthfulness*: A double auction is defined to be truthful if no buyer or seller can obtain a higher utility gain by cheating, *i.e.*, setting bid or ask not equal to its true utility value. Trustfulness is essential to resist market manipulation and ensure auction fairness and efficiency [20]. It also eliminates the overhead of strategizing over other players.

Generally, double auctions are difficult to analyze game-theoretically. However, the trading mechanism here bears some similarity with the 0.5-double auction under the assumptions that the number of buyers and sellers are large [8]. It has been proved that the 0.5-double auction is not incentive compatible, neither for buyers nor for sellers. Therefore, truthful bidding is not an equilibrium in this case. But when the number of buyers and sellers, m and n respectively, grow at the same rate (the ratio of m/n being bounded both from above and away from 0), then all equilibria are within $O(1/m)$ of the truthful bidding and expected inefficiency of any equilibrium is $O(1/m^2)$. When m is large, which is the norm in reflector selection markets, the proposed double auction mechanism

satisfies the truthfulness requirement.

▷ *Ex-post Individual Rationality*: A double auction is ex-post individually rational if the expected utility gain of any truthful participant in the auction is non-negative for all possible outcomes. Ex-post individual rationality ensures a non-negative utility gain if participant reports its true utility value, therefore provides incentives in the auction.

In our double auction markets, the clearing price of buyer u and seller r is defined to be $p_u^r = \frac{1}{2}(b_u^r + a_r)$, where both bid and ask are non-negative by definition. So when the trade is made, *i.e.*, we have $b_u^r > a_r$, $b_u^r - p_u^r = p_u^r - a_r = \frac{1}{2}(b_u^r - a_r) > 0$. This implies that for any winning buyer u and seller r, their utility gains are non-negative, which proves our double auction markets are ex-post individually rational.

▷ *Ex-post Budget Balance*: A double auction is ex-post budget balanced if the mechanism's payoff is non-negative for all possible outcomes, which ensures that the auction will never run into deficit. Since the number of winning buyers and sellers are guaranteed to match, and in each of the trading pairs, the payoff of a buyer is always the same as the revenue of a seller. That is, the auctioneer's payoff is always zero. By the definition of budget balance, our double auction markets are ex-post budget balanced.

5. TOUCHTIME: PERFORMANCE EVALUATION

We believe that the best way to evaluate *TouchTime* is to implement it, with a real-world multi-touch application that takes advantage of *TouchTime* to enable multi-touch gesture streaming. Towards this objective, we have implemented *MusicScore*, a professional-grade multi-touch application for music composition (shown in Fig. 13), entirely from scratch and using only multi-touch gestures. We developed *MusicScore* using the recently released iPad Programming SDK from Apple Inc., with more than 58,000 lines of code (LOC) in Objective-C. We designed *MusicScore* to make it natural and intuitive to compose music, for serious composers and amateur hobbyists alike. *MusicScore* takes full advantage of our implementation of the *TouchTime* system to allow composers to enjoy a live collaborative session, and students to benefit from a live educational experience, all without any knowledge of the architectural design choices in *TouchTime*.

Figure 13: *MusicScore* in action: two users are collaboratively composing a musical piece.

In addition, we have implemented the entire *TouchTime* system from scratch, from mobile devices to our cloud service, using Objective-C based on the Cocoa framework, with over 4,000 LOC in total. *MusicScore* produces actual multi-touch gesture streams, which are used to evaluate the performance of *TouchTime*. In order to collect run-time traces, a compact logging module has been implemented to anonymously record performance metrics as multi-touch gestures are streamed. We dedicate this section to analyze traces we collected from *MusicScore* and to evaluate the performance of *TouchTime*, with a focus on the reflector selection algorithm based on double auctions.

Problems with Greedy Reflector Selection

We start our studies with the intuitive greedy selection algorithm. In Fig. 14, our measurement shows that the average end-to-end delay is acceptable in general (*e.g.*, 140.2 msec for Wi-Fi users connected to different access points at various ISPs); however, we observe significant variations of delays over time in our trace. To quantify such delay variations, we plot the CDF of standard deviation of end-to-end delays in different Internet access types in Fig. 15. As shown, in both faster Wi-Fi networks and slower cellular networks, half of the end users experienced more than 98 milliseconds of delay variations. This can be attributed to the lack of stability in real-world wireless networks and last-mile ISPs, mostly due to cross traffic.

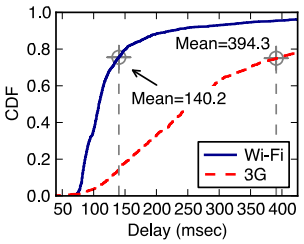

Figure 14: CDF of end-to-end delay between Wi-Fi/3G users with the greedy algorithm.

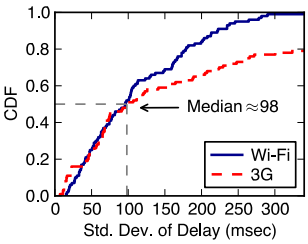

Figure 15: CDF of standard deviation of end-to-end delays in different Internet access types.

When we measure the server load — defined as the number of active connections relayed — at our reflectors in Fig. 16, results confirm the existence of the reflector fluctuation problem discussed in Sec. 4.2: the server load may be shifted back and forth between two nearby reflectors over time, as nodes are allowed to greedily and dynamically switch among reflectors.

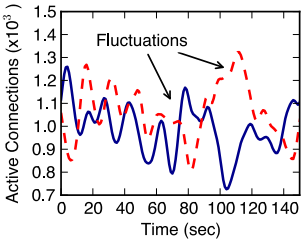

Figure 16: Variations of server load in two nearby reflectors with the greedy algorithm.

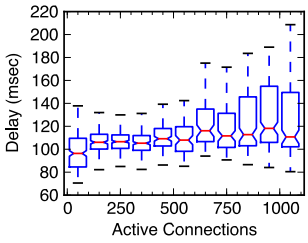

Figure 17: Correlation between the load of a reflector and end-to-end delays.

To gain a better understanding of the greedy algorithm, we examine the correlation between the server load of a reflector and the end-to-end delay. In Fig. 17, we plot the box-and-whisker diagram to study five statistical characteristics: the smallest observation (within 1.5 IQR), lower quartile, median, upper quartile, and

Table 1: Percentage of overhead in *TouchTime* with the auction-based reflector selection algorithm.

Overhead (%)	Delay Probing	Load Notification
Mean	1.613	0.277
95% CI	0.095	0.049

the largest observation. As we can see, when the server load increases, the median of delay increases slightly, and the upper quartile as well as the maximum delay increase significantly. This implies that relay paths over a heavily-loaded reflector may experience higher end-to-end delays. These results have further confirmed our explanations to the observed problem when the dynamic greedy algorithm is applied.

Evaluations of Auction-Based Reflector Selection

Since we have proposed the auction based mechanism to address the observed load fluctuation problem in the greedy algorithm, a reality check is conducted in our experiments to verify its effectiveness. To quantitatively measure the load variations, we define the normalized load difference between two reflectors as the maximum difference of loads during a 5-min period divided by the average of server loads in that period of time. As shown in Fig. 18, we plot the CDF of the normalized load difference between two "nearby" reflectors located in the same ISP. Thanks to the auction-based reflector selection algorithm, the mean of normalized load difference is reduced from 4.2% to 1.1%.

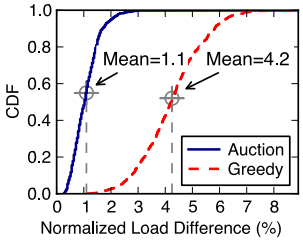

Figure 18: CDF of normalized load difference (%) between two nearby reflectors.

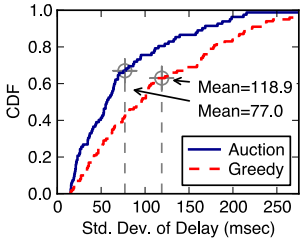

Figure 19: CDF of std. dev. of delay in Wi-Fi end users with greedy and auction-based reflector selection algorithms.

Furthermore, we compare end-to-end delays between the auction-based and the greedy algorithm. Since the reflector load variation has been significantly reduced, we have also observed a reduction of delay variation in Fig. 19, which shows the CDF of standard deviation of delays with two algorithms. Note that similar results are also observed in traces from nodes with other types of Internet access. By plotting the CDF of average end-to-end delays between Wi-Fi nodes in Fig. 20, we notice that the average delay is improved by a margin of 19%. It can be explained as users are more likely to enjoy a shorter delay when we allow packets to be transmitted over multiple relays in the auction-based reflector selection algorithm. This is verified by Fig. 21, which shows that two trades succeeded at half of auction markets in a user's session, *i.e.*, there are two active relay paths in around 50% of time.

As the auction based algorithm introduces additional signaling overhead and packet redundancy, we also wish to investigate the flip side of the same coin in our performance evaluation. Fig. 22 reveals the CDF of a typical original multi-touch stream bit rate and the corresponding uploaded stream rate. Although redundant packets consume more upload bandwidth, the total upload bit rate

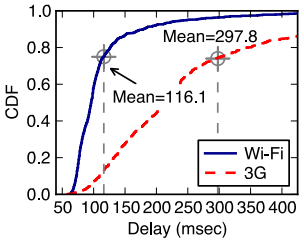

Figure 20: CDF of the end-to-end delay between Wi-Fi/3G users with the auction-based reflector selection algorithm.

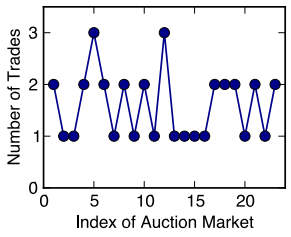

Figure 21: Number of conducted trades in auction markets during a typical session of a *TouchTime* user.

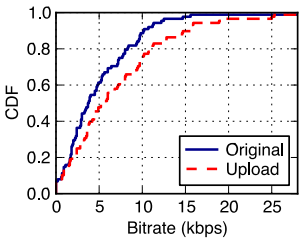

Figure 22: CDF of the original multi-touch stream bit rate and the uploaded stream rate.

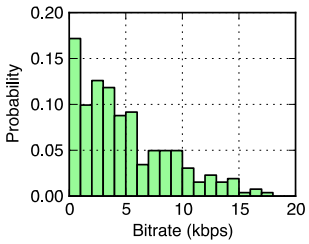

Figure 23: Histogram of the bit rate of multi-touch streams in *MusicScore* traces.

is still reasonably low, considering most multi-touch streams are bursty but low as shown in Fig. 23. In Table 1, we summarize the mean of two major categories of overhead: delay probing messages and server load notifications (*i.e.*, asks in double auction markets), along with their 95% confidence intervals. Since both types of messages are less than 20 bytes, average values of the overhead are 1.6% and 0.3%, respectively. Finally, our trace studies show that the overall overhead ratio, including handshake signaling, is no higher than 3%, which is reasonably low in practice.

Comparisons with Random-K

To study the performance of the greedy and auction algorithms in a suitable context, we have also implemented and evaluated the performance of the Random-K algorithm, one of the simplest and most recently proposed relay selection algorithm in the literature [19]. With the Random-K algorithm, a user probes for end-to-end delays of relay paths over a random set of K reflectors in the reflector cloud, and then use the first reflector who provides a response as its

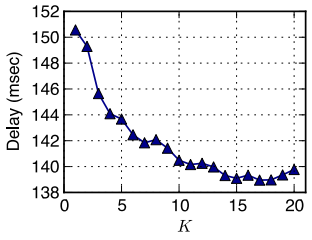

Figure 24: Correlation between K and the average end-to-end delay in Random-K reflector selection algorithm.

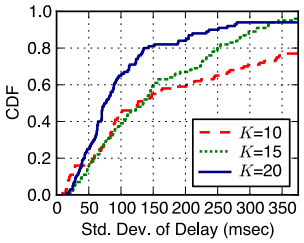

Figure 25: CDF of standard deviation of delay in Wi-Fi end users with Random-K algorithm.

relay server.

We first study the correlation between the value of K and the average end-to-end delay experienced by *TouchTime* users. As shown in Fig. 24, users tend to experience longer delays when the K is small; as K increases, the average delay gradually settles to 146 msec, which is similar to the performance of the greedy algorithm. With respect to the standard deviation of delays, Fig. 25 reveals that users suffer from significant fluctuations of end-to-end delays when K is small. These results imply that a large enough K, *e.g.,* $K = 20$, must be used to provide a probabilistic guarantee on a low and stable delay.

6. RELATED WORK

In recent years, there is a strong trend of closely integrating mobile devices with the cloud to provide various feature-rich services. Cuervo *et al*. proposes MAUI to offload computing workload from mobile devices to the cloud to maximize energy saving [6]. CloneCloud designed by Chun *et al*. makes it possible for unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution [5] to the cloud. However, to our best of knowledge, no literature has reported system frameworks supporting multi-touch gesture streaming in mobile devices with cloud-based backends.

With respect to reflector selection algorithms, a number of systems has been proposed to predict latencies in the Internet. Network coordinate systems, such as Vivaldi [7], are excellent examples of using synthetic coordinates to infer Internet latencies without measurements. iPlane [15] is another example of a complete system designed to predict latencies of Internet paths by using existing measurements, which can be used to select nearby servers. ISP/AS-aware relay selection mechanisms are also used to transmit streams with high bit rates, such as high-quality voice [17]. In contrast, *TouchTime* attempts to select reflectors based on balanced considerations of both latencies and server load, and since the *TouchTime* cloud only contains a small number of reflectors in practice, the overhead of periodic measurements is not only acceptable but also necessary, since end-to-end delays over reflectors may vary substantially over time and depend on the server load.

7. CONCLUDING REMARKS

This paper presents the design and implementation of the *TouchTime* system, a first attempt to stream multi-touch gestures between end users with mobile devices across the Internet. We found that streams in *TouchTime* have low yet bursty bit rates, and require reliable transport. *TouchTime* combines local presentation of multi-touch gestures on mobile devices and optimized transport services in the cloud, without perceptible boundaries to applications. One of the highlights in the design is the use of continuous double auctions to achieve a balanced tradeoff between delays and server load. A unique upshot of *TouchTime* is that all results are based on our solid implementation in Objective-C based on the Cocoa SDK, including over $58,000$ LOC in *MusicScore*, our new real-world music composition application on the iPad, and $4,000$ LOC in the *TouchTime* foundation framework itself. The entire project reflects over 26 person-months of development work.

8. REFERENCES

[1] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In *Proc. ACM SIGCOMM '08*, pages 389–400.

[2] S. F. Brown. Hands-On Computing: How Multi-Touch Screens Could Change the Way We Interact with Computers and Each Other? *Scientific American*, July 2008.

[3] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Econimic Models for Resource Allocation and Scheduling in Grid Computing. *Concurrency and Computation: Practice and Experience*, 14(13):1507–1542, 2002.

[4] L. Y. Chu and Z.-J. M. Shen. Agent Competition Double-Auction Mechanism. *Management Science*, 52(8):1215–1222, 2006.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proc. EuroSys '11*, pages 301–314.

[6] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proc. ACM MobiSys '10*, pages 49–62.

[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. ACM SIGCOMM '04*, pages 15–26.

[8] Z. Despotovic, J.-C. Usunier, and K. Aberer. Towards Peer-to-Peer Double Auctioning. In *Proc. 37th Annual Hawaii International Conference on System Sciences*, 2004.

[9] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. SPORC: Group Collaboration using Untrusted Cloud Resources. In *Proc. USENIX OSDI'10*.

[10] S. Gjerstad and J. Dickhaut. *Price Formation in Double Auctions*, pages 106–134. 2001.

[11] L. Hu, J.-Y. Le Boudec, and M. Vojnoviae. Optimal Channel Choice for Collaborative Ad-Hoc Dissemination. In *Proc. IEEE INFOCOM '10*.

[12] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *The Journal of the Operational Research Society*, 49(3):237–252, 1998.

[13] P. Klemperer. What Really Matters in Auction Design. *Journal of Economic Perspectives*, 12(1):169–189, 2002.

[14] S. M. M. Yokoo, Y. Sakurai. Robust Double Auction Protocol Against False-Name Bids. In *Proc. IEEE ICDCS '01*, pages 137–145.

[15] 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. USENIX OSDI '06*, pages 367–380.

[16] R. P. McAfee. A Dominant Strategy Double Auction. *Journal of Economic Theory*, 56(2):434–450, 1992.

[17] S. Ren, L. Guo, and X. Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. In *Proc. IEEE ICDCS '06*, pages 70–80.

[18] J. Schöning. Touching the Future: The Rise of Multi-touch Interfaces. *Per Ada Magazine*, 2010.

[19] M. Venkataraman and M. Chatterjee. Effects of Internet Path Selection on Video-QoE. In *Proc. 2nd ACM Conference on Multimedia Systems (MMSys)*, pages 45–56, 2011.

[20] S. Wang, P. Xu, X. Xu, S. Tang, X. Li, and X. Liu. TODA: Truthful Online Double Auction for Spectrum Allocation in Wireless Networks. In *Proc. IEEE Symposium on New Frontiers in Dynamic Spectrum*, pages 1–10, 2010.

[21] X. Zhou and H. Zheng. TRUST: A General Framework for Truthful Double Spectrum Auctions. In *Proc. IEEE INFOCOM '09*, pages 999–1007.