TIMECARD

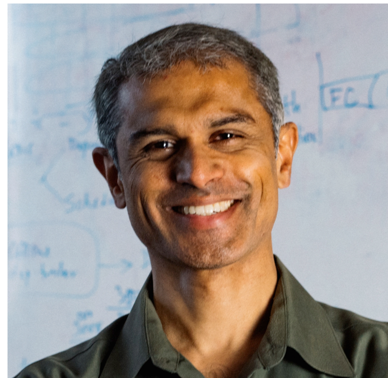
Controlling User-Perceived Delays in Server-Based Mobile Applications

University of Warsaw | Faculty of Computer Science

Lecture: Distributed Systems

Lecturer: Dr. Konrad Iwanicki

Date: 25.11.2015



Publication Details

November 2013,
SOSP 2013 -The 24th ACM Symposium on Operating Systems Principles

Lenin Ravindranath
M.I.T. & Microsoft Research

Jitendra Padhye
Microsoft Research

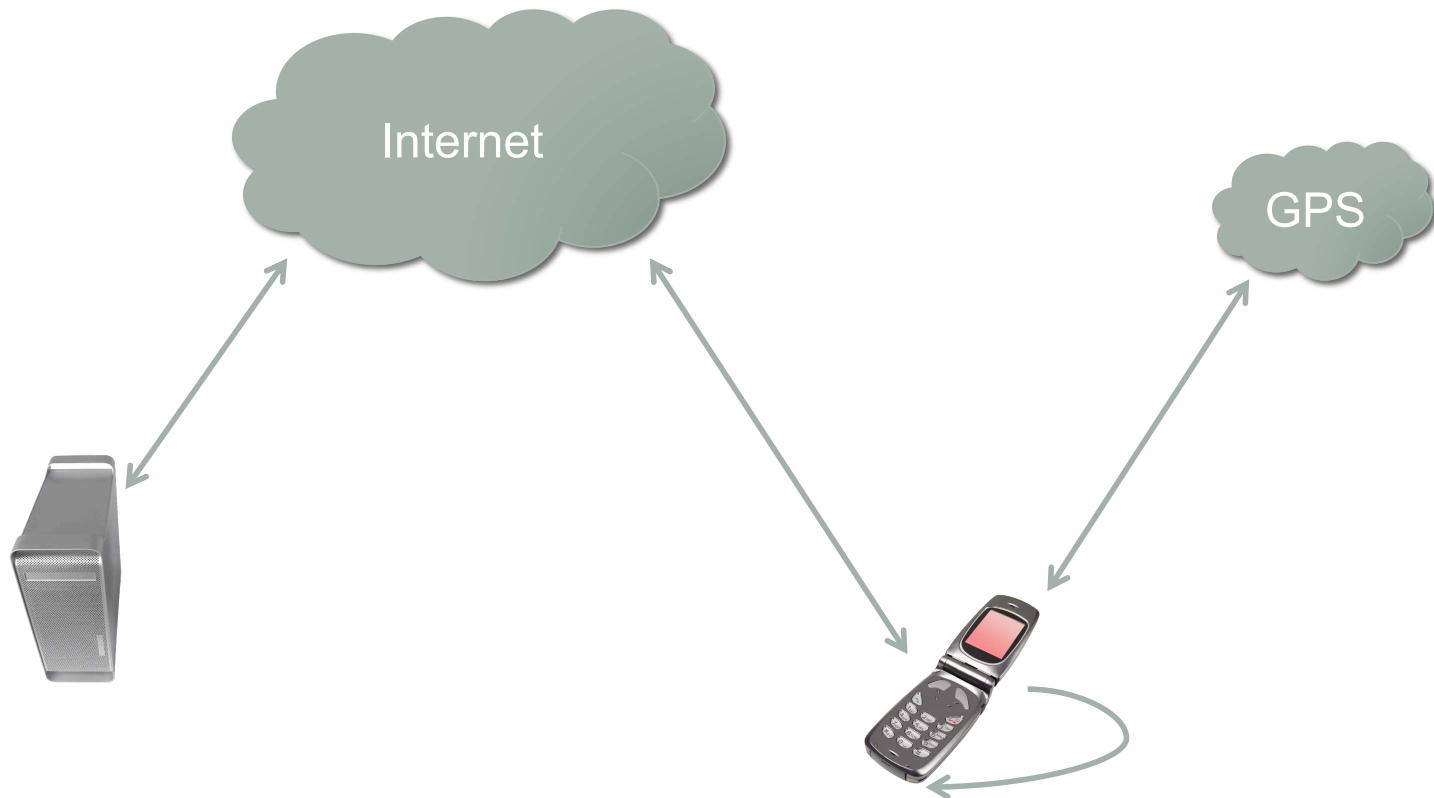
Ratul Mahajan
Microsoft Research

Hari Balakrishnan
M.I.T.

Table of content

1. Introduction
2. Timecard Architecture
3. Tracking elapsed time
    1. Transaction Tracker
        1. Transaction context
        2. Collecting data to predict C_2 / N_2
        3. Tracking transaction completion
    2. Time Synchronization
4. Predicting remaining time
5. Implementation
6. Evaluation
7. Limitations

Problem Scenario



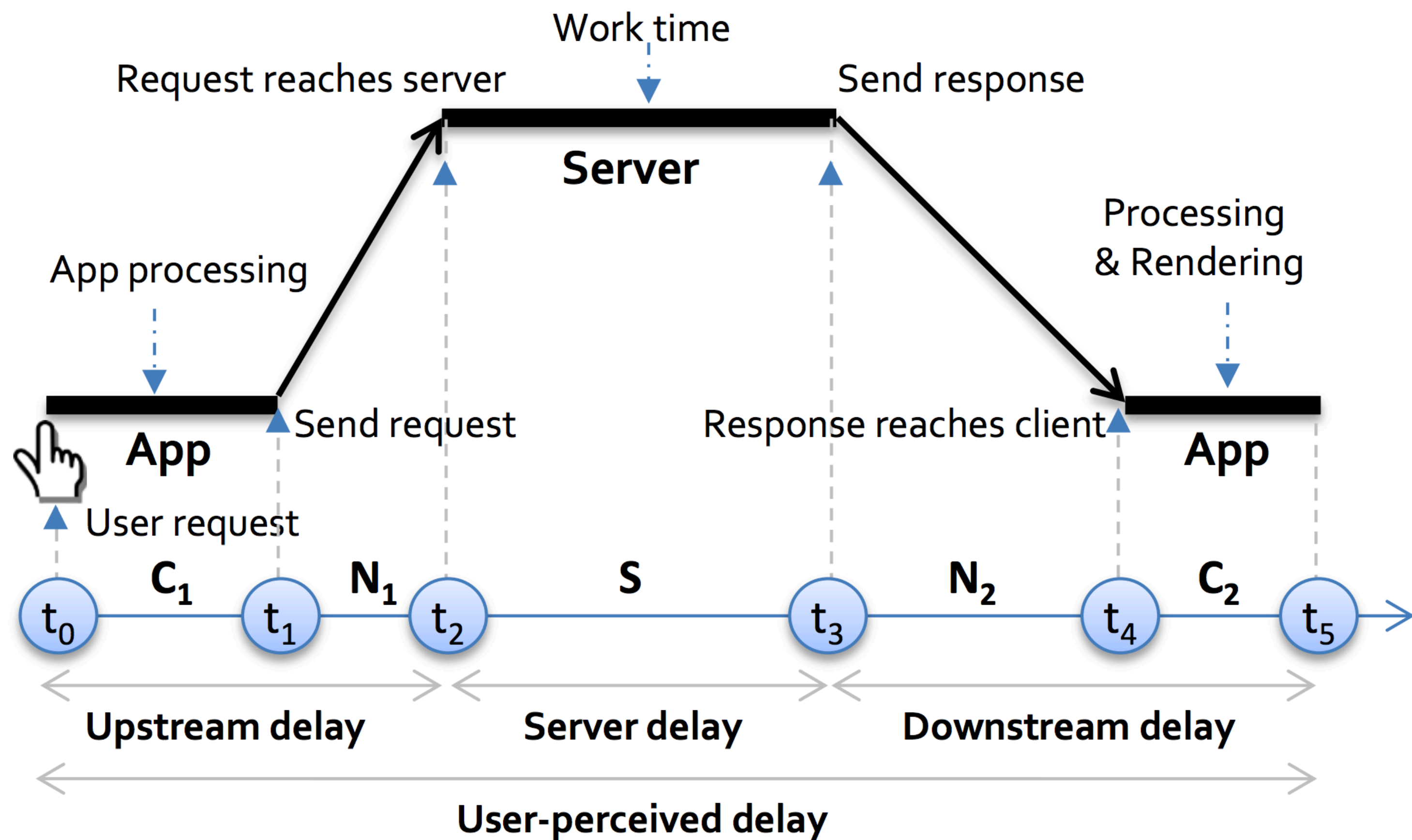
Goals

Given the desired end-to-end delay, the idea is to allow the server to obtain answers to two questions:

1. *elapsed time*: How much time has elapsed since the initiation of the request?
2. *predicted remaining time*: How much time will it take for the client to receive an intended response over the network and then process it?

➔ *work time*

Anatomy of a user transaction



Timecard API

1. `GetElapsedTime()`

   Any component on the processing path at the server can obtain the time elapsed since t0.

2. `GetRemainingTime(bytesInResponse)`

   At the server, a component can obtain an estimate of N2 + C2. Timecard provides this estimate as a function of the size of the intended response.

work time < *desired user-perceived delay*

- `GetRemainingTime(x)`
- `GetElapsedTime()`

Timecard Architecture

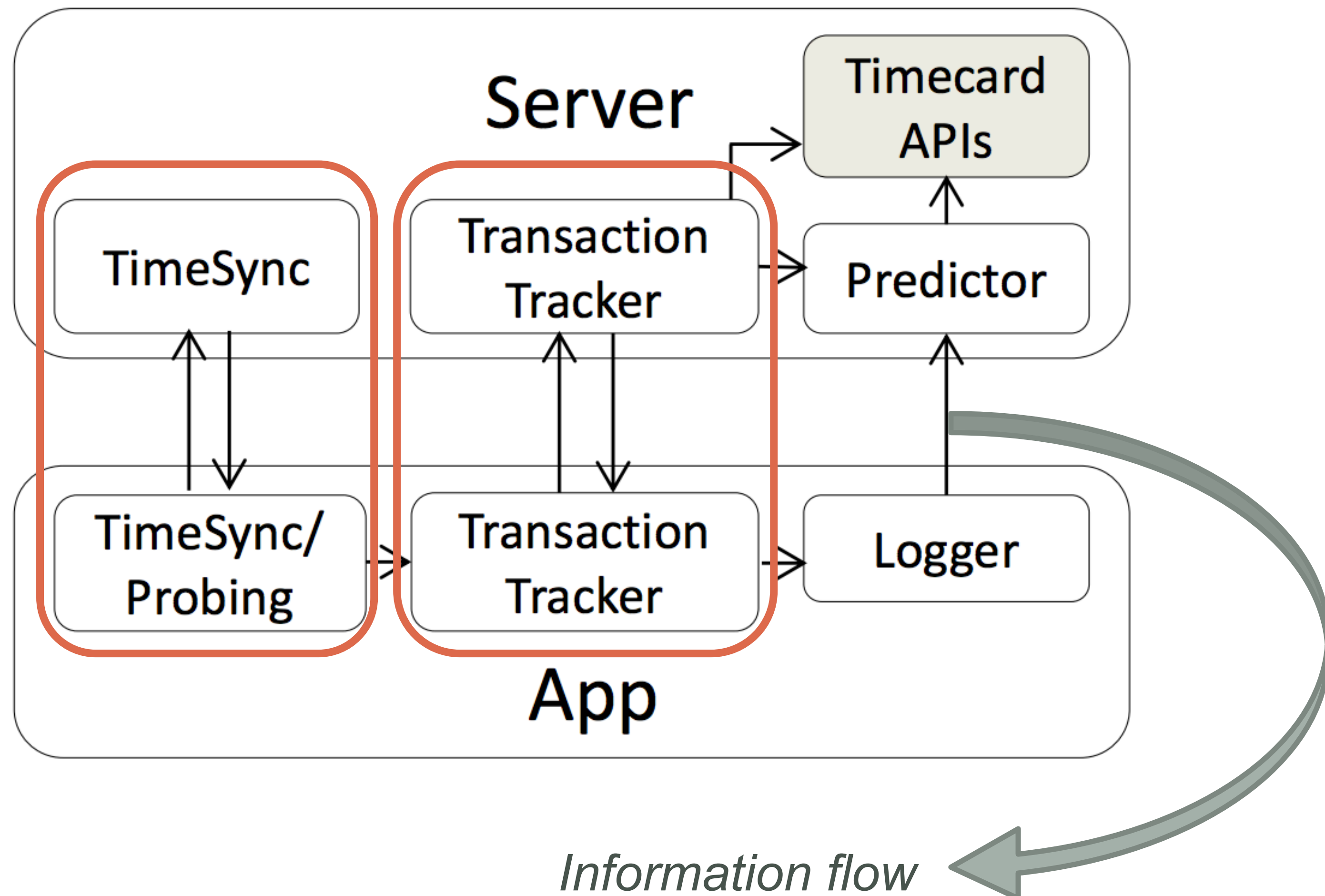


Table of Content

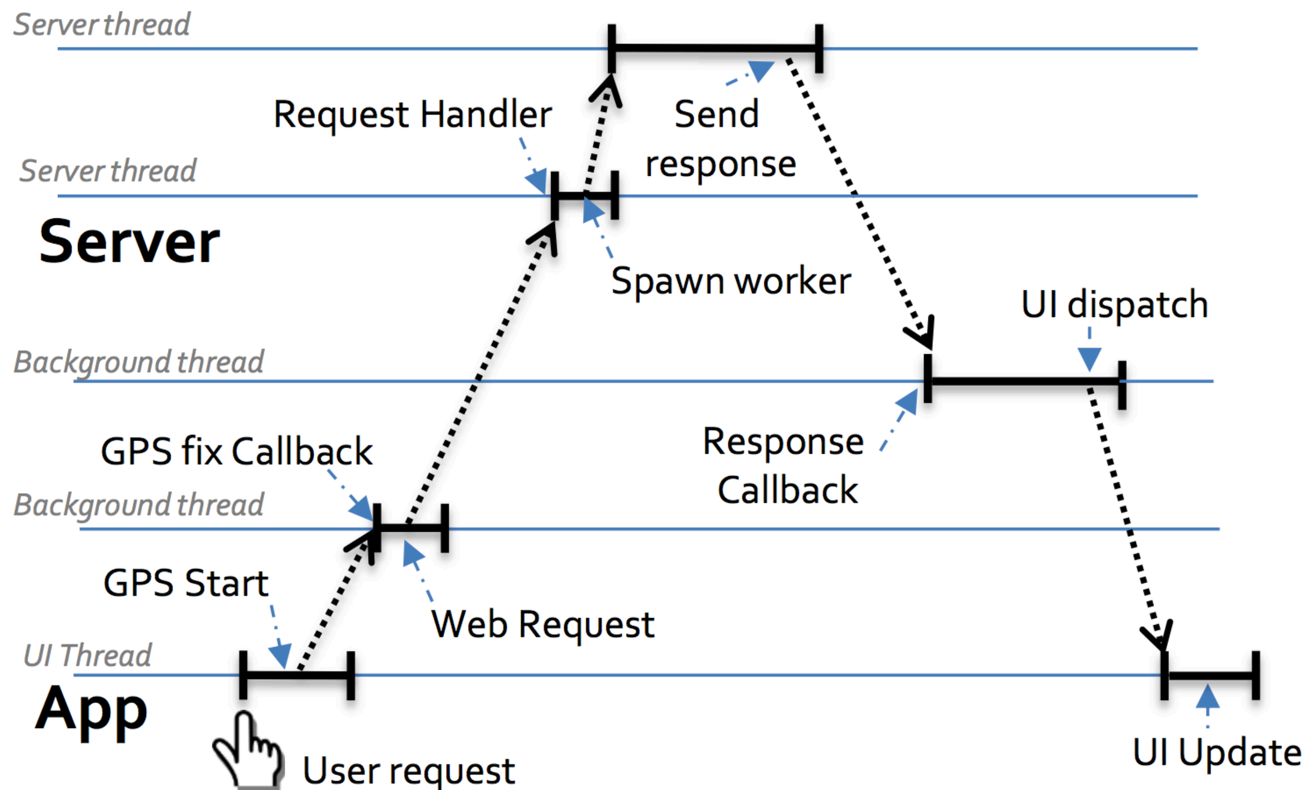
1. Introduction
2. Timecard Architecture
3. Tracking elapsed time
    1. Transaction Tracker
        1. Transaction context
        2. Collecting data to predict C_2 / N_2
        3. Tracking transaction completion
    2. Time Synchronization
4. Predicting remaining time
5. Evaluation
6. Limitations

Tracking elapsed time

- Identify each user transaction
- Maintain transaction context (TC) objects
- Synchronize time between client and server

Challenges

Example : Location-based App querying a server



Transaction tracking

Timecard instruments client and server can collect information in transaction context objects (TC)

- TCs are available to every client and server thread working on transaction
- transaction information can be passed across client-server boundry
- extends AppInsight framework

Transaction context

New UI event

➔ create new TC (unique ID and timestamp t_0)

➔ maintain reference to TC in thread's local storage

Tracked information	*Purpose*	*Set by*	*Used by*
Application Id	Unique application identifier	Client	Server and Predictor
Transaction Id	Unique transaction identifier	Client	Client and Server
Deadline	To calculate remaining time	Client	Server
t_3	To calculate N_2 for training data	Server	Predictor
t_4	To calculate N_2 and C_2 for training data	Client	Predictor
t_5	To calculate C_2 for training	Client	Predictor
Entry Point	To predict C_2, and to label training data	Client	Server and Predictor
RTT	To predict N_2, and to label training data	Client	Server and Predictor
Network type	To predict N_2 and to label training data	Client	Server and Predictor
Client type	To predict N_2 and to label training data	Client	Server and Predictor
Size of response from cloud service	To predict N_2 and to label training data	Server	Predictor
Pending threads and async calls	To determine when transaction ends	Client	Client

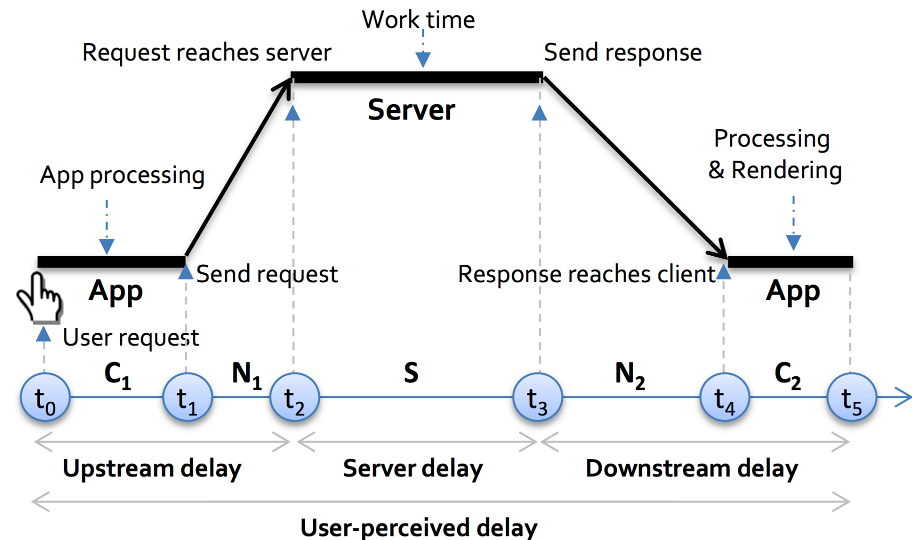
What happens with a TC ?

- Tracking a transaction across asynchronous calls
- Passing TC from client to server
- Tracking transaction at the server
- Handling server response and UI updates

Collecting data to predict C_2 and N_2

Transaction tracking also enables Timecard to collect the data to train the N_2 and C_2 predictors for subsequent transactions.

- logs t_3 just before it sends the response to the client
- records the number of bytes sent in the response
- client‘s callback handler logs t_4 as t_5



Tracking transaction completion

Timecard maintains a list of currently active transactions

- Timecard keeps track of active threads and pending asynchronous calls
- List empty ➔ application is „idle“

When a transaction is completed Timecard can remove the TC on the client (on the server the TC can be remved as soon as t_3 is recorded)

Synchronizing time

A timestamps in the TC are meaningful across the client-server boundary only if the client and the server clocks are synchronized

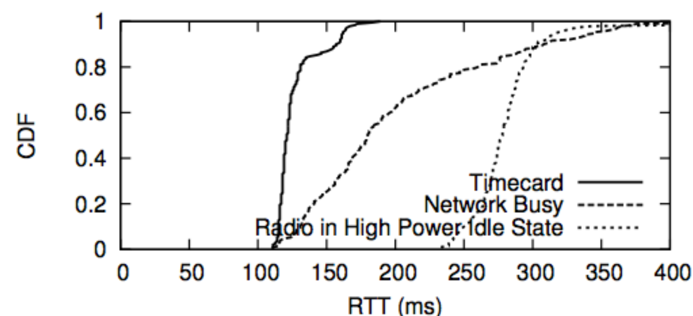
Given the linearity of the drift and the symmetry assumption, client and server clocks can be synchronized using Paxson's algorithm

Paxson's algorithm

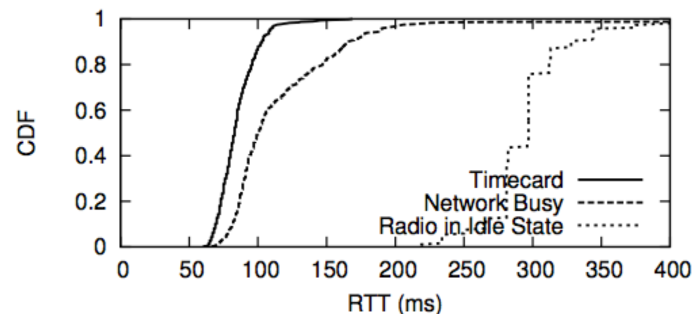
1. At time $\tau 0$ (client clock), send an RTT probe. The server responds, telling the client that it received the probe at time $\tau 1$ (server clock). Suppose this response is received at time $\tau 2$ (client clock).
2. Assuming symmetric delays, $\tau 1 = \tau 0 + (\tau 2 - \tau 0)/2 +$, where is an error term consisting of a fixed offset, c, and a drift that increases at a constant rate, m.
3. Two or more probes produce information that allows the client to determine m and c. As probe results arrive, the client runs robust linear regression to estimate m and c

Synchronizing time

RTTs of probes from an app to a server with Timecard, when the network is busy, and when the radio is either idle or busy



(a) HSPA network. When the Radio is in Idle state, pings take 1800 ms to 2300 ms! (Not shown.)



(b) LTE network.

Table of Content

1. Introduction
2. Timecard Architecture
3. Tracking elapsed time
    1. Transaction Tracker
        1. Transaction context
        2. Collecting data to predict C_2 / N_2
        3. Tracking transaction completion
    2. Time Synchronization
4. Predicting remaining time
5. Implementation
6. Evaluation
7. Limitations

Predicting Remaining Time

Timecard's `GetRemainingTime` function returns estimates of N_2 and C_2 for a specified response size.

$N_2 + C_2$ = total amount of time required to receive and render the response at the client

The estimates are generated by decision tree algorithms that use models built from historical data

Predicting N_2

Empirical data-driven model to predict N_2

1. The response size

2. Recent RTT between the client and server

   re-using the ping data collected by the TimeSync component

3. Number of bytes transmitted on the same connection before the current transfer

Predicting C_2

To understand the factors that affect the processing and rendering time on the client after the response is received : analysis third party apps with 1653 types of transaction

➔ C_2 ~linear in response length

Prediction with empirical data-driven model similar to the one used for N_2

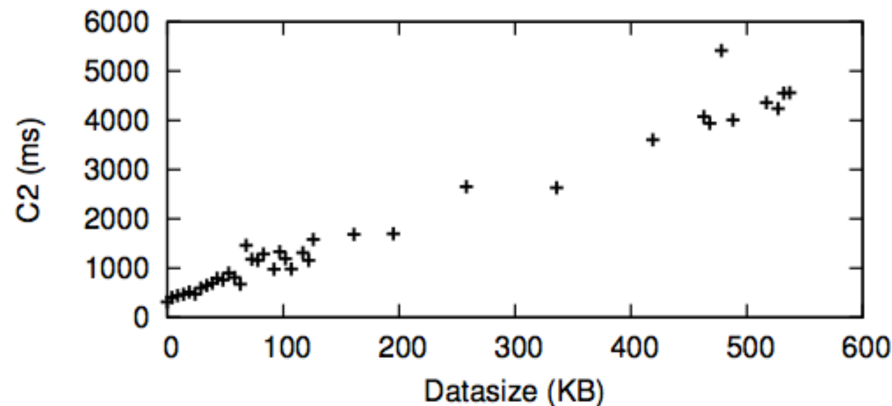


Table of Content

1. Introduction
2. Timecard Architecture
3. Tracking elapsed time
    1. Transaction Tracker
        1. Transaction context
        2. Collecting data to predict C_2 / N_2
        3. Tracking transaction completion
    2. Time Synchronization
4. Predicting remaining time
5. Implementation
6. Evaluation
7. Limitations

Implementation

- implemented in C#
- 18467 lines of code
- targeted for Windows Phone Apps and .NET services

Table of Content

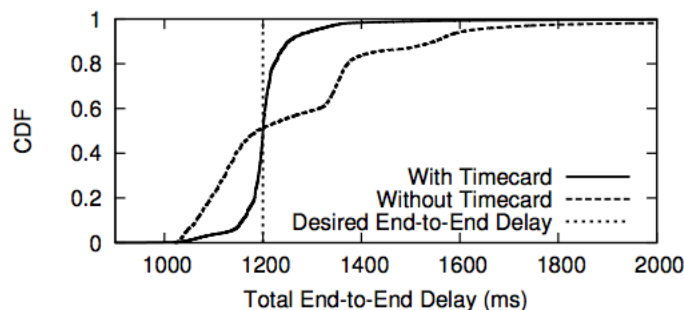
1. Introduction
2. Timecard Architecture
3. Tracking elapsed time
   1. Transaction Tracker
      1. Transaction context
      2. Collecting data to predict C_2 / N_2
      3. Tracking transaction completion
   2. Time Synchronization
4. Predicting remaining time
5. Implementation
6. Evaluation
7. Limitations

Is Timecard Useful?

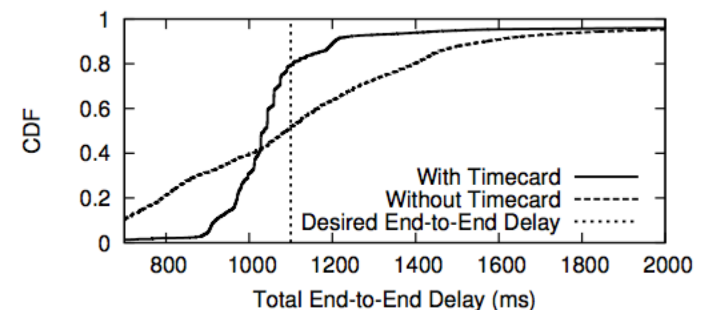
1. How common is the single request-response transaction in mobile apps?
2. How variable are user-perceived delays?
3. How variable are the four components (C_1, C_2, N_1, N_2) of the userperceived delay that Timecard must measure or predict?

End-to-End-Evaluation

- Incorporation of Timecard into two mobile services and their associated apps
- The apps were installed on the primary mobile phones of twenty users, configured to run in the background to collect detailed traces



(a) Context ads delay.

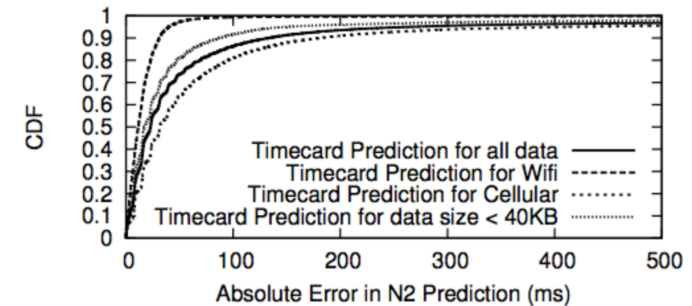


(b) Twitter analysis delay.

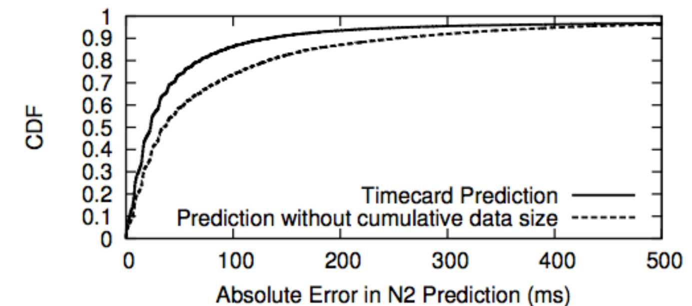
Figure 10: User-perceived delays for two apps. With Timecard, delays are tightly controlled around the desired value.

Accuracy of predictions

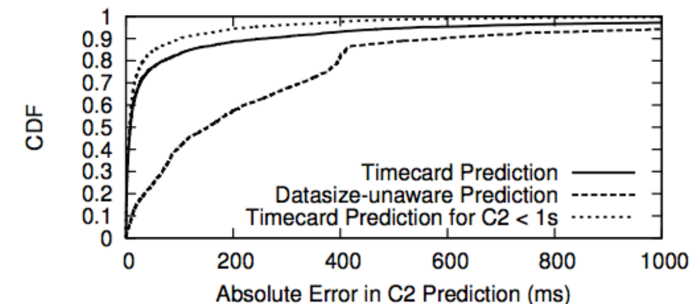
- N_2 predictions more acurat for WiFi then for cellular networks
- C_2 predictions must take size of dowloaded data into account
- C_2 predictions are more accurat for short transactions



(a) N_2 Prediction error



(b) N_2 Prediction with and without cumulative data



(c) C_2 Prediction error

Figure 14: Accuracy of N_2 and C_2 prediction

Table of Content

1. Introduction
2. Timecard Architecture
3. Tracking elapsed time
    1. Transaction Tracker
        1. Transaction context
        2. Collecting data to predict C_2 / N_2
        3. Tracking transaction completion
    2. Time Synchronization
4. Predicting remaining time
5. Implementation
6. Evaluation
7. Limitations

Limitations

- Limitations of N_2 predictor
- Complex transactions
- Privicy and Security
- Applicability to other platforms

Thank you for your attention

University of Warsaw | Faculty of Computer Science
Lecture: Distributed Systems
Lecturer: Dr. Konrad Iwanicki
Date: 25.11.2015