TANGO: Enabling New Services through Cooperation between Cellular Network and Mobile Devices

Nawanol Theera-Ampornpunt, Sambit Mishra, Saurabh Bagchi, Kaustubh Joshi (AT&T), Rajesh Panta (AT&T)

Motivation

- Cellular network collects a lot of low-level information about each mobile device
- Collected information can be used to improve service reliability in real time
  - handoff decision
  - control radio power
- Applications can benefit from knowing the current or predicted network conditions
  - Prefetch content before entering congested region (e.g., for web browser or video client)
  - When a voice call drops, automatically reconnect if the drop is predicted to be short
  - Switch to less-congested older technology (e.g., LTE to 3G)
- Without access to real-time data, these actions must be enabled all the time, which leads to wasted resources or poorer user experience
Motivation

- We propose a framework called TANGO that enables mobile applications to subscribe to services provided by the network
  - general enough for most (if not all) services

![Diagram of TANGO framework](image)

Services

- Implemented two applications utilizing two services:
  - video precaching using congested region alert service
  - call auto-reconnect using drop duration prediction service
- Video precaching
  - Video clients such as YouTube throttle download rate in order to save bandwidth
  - If user enters a congested region where available bandwidth is lower than the video’s bit rate, playback will be disrupted
  - Congested region alert service continuously predicts user location and generates an alert when the user is going to enter a congested region in the near future
  - On receiving the alert, video client precaches video by downloading at maximum rate
Services

• Call auto-reconnect
  – When user is in a voice call, the phone registers for the drop duration prediction service
  – The service continuously predicts whether drop duration will be short (e.g., <10 secs) or long, IF a drop occurs
    • drop duration = duration since call drop where reconnection cannot be made
    • sends prediction to application

![Diagram showing call drop and earliest possible reconnection](image)

Services

• Call auto-reconnect
  – When a call drop actually occurs and predicted drop duration is short, the call server pauses the call while the disconnected party auto-reconnects
  – It is important that the call server and the client agree on what to do when a drop occurs
Framework Design

- Three components:
  - application
  - framework app called TANGO App
  - service component
- TANGO app acts as the middleman between application and service component
  - implemented on Android
  - communicates with application using Intents
  - communicates with service components using UDP
- Application
  - registers for service(s) it is interested in
  - receives notifications from service component
  - initiate appropriate actions

Framework Design

- API for applications:
  - list()
  - register(serviceID [, parameters])
  - deregister(serviceID)
  - is_registered(serviceID)
  - send(serviceID, message)
  - send_once(serviceID, message)

<table>
<thead>
<tr>
<th>RegistrationID</th>
<th>Application</th>
<th>ServiceID</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YouTube</td>
<td>1</td>
<td>t=10</td>
</tr>
<tr>
<td>2</td>
<td>Caller</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Downloader</td>
<td>5</td>
<td>a=3, b=7</td>
</tr>
<tr>
<td>4</td>
<td>YouTube</td>
<td>1</td>
<td>t=25</td>
</tr>
</tbody>
</table>
Framework Design

• Service Component
  – Accesses data directly from the network
  – Continuously runs filter/classifier
  – Send notifications to the phone using push notification service (e.g., Google Cloud Messaging for Android)
  – Multiple instances, partitioned by region and/or user ID
    • Correct instance located by service redirector
    • Need a handoff mechanism

• Heartbeat mechanism
  – Heartbeat messages are sent by TANGO App to service components

Framework Design

• API for service components:
  – timeout(serviceID, registrationID)
  – move(serviceID, registrationID, newServiceComponent)
  – send(serviceID, registrationID)

<table>
<thead>
<tr>
<th>RegistrationID</th>
<th>Application</th>
<th>ServiceID</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YouTube</td>
<td>1</td>
<td>t=10</td>
</tr>
<tr>
<td>2</td>
<td>Caller</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Downloader</td>
<td>5</td>
<td>a=3, b=7</td>
</tr>
<tr>
<td>4</td>
<td>YouTube</td>
<td>1</td>
<td>t=25</td>
</tr>
</tbody>
</table>
Experiments + Results

• Precaching
  – Experiment 1: measure playback disruptions due to rebuffering and amount of bandwidth used when only one user uses precaching

Simulation Setup

• YouTube client
  – We used the Simple PlayerView for our youtube client.
  – It comes as a sample application in the YouTubeAndroidAPIDemo package.
  – We modified the client to be able to measure the time spent in buffering each video.
  – Also this client follows the real time viewer abandonment rate shown in the next slide.
Simulation Setup

• YouTube client contd.
  – This graph summarizes average video abandonment by time spent viewing.
  – Viewer abandonment appears to be a function of time spent in-stream and follows a relatively predictable trajectory.

Simulation Setup

• Cell proxies in the Network
  – We only use one cell proxy for our simulation.
  – We use a lightweight proxy server tinyproxy running on a linux machine for our cell proxy.
  – The movement of a device from one cell to another is simulated by changing the current bandwidth of our cell proxy to the available bandwidth of the new cell.
  – The bandwidth limiting is done using the tc tool.
Experiments + Results

• Precaching
  – Experiment 2: measure playback disruptions due to rebuffering and amount of bandwidth used when everyone uses precaching

Simulation Setup

• Non Android client emulator
  – The non android client emulator gives us an estimate of the bandwidth usage of all the devices, other than our client, in any given cell.
  – All it does is, selects a random video, find out the bandwidth required for it, decide whether to use precaching or not and then based on this make an estimate of the total bandwidth usage every second and send it to the cell proxy.
  – The cell proxy then decides the amount of bandwidth available to the client emulator.
Experiments + Results

• Precaching
  – Experiment 3: measure changes in network traffic pattern when everyone uses precaching

\[ P(C_3 | C_2, C_1) = \frac{P(C_3, C_2, C_1)}{P(C_2, C_1)} \]

• Prediction accuracy for different number of levels (threshold = 0.20)
Experiments + Results

• Precaching
  – Location predictor
    • Prediction accuracy for different threshold (levels = 2)

• Investigate how the predictor can be automatically retrained

Experiments + Results

• Call auto-reconnect
  – drop duration prediction accuracy
    • Data from window \([t - 30 \text{ seconds}, t)\) where \(t\) is the call drop time
    • Classifiers: support vector machine (SVM) and AdaBoost with decision stump