iOS application for content sharing in Public transit systems

Yu-Wei Chang
Department of Computer Science, Columbia University
Email: yc2574@columbia.edu

Abstract--Supporting the Internet connection became a part of the modern public transportation system. This paper applies, Bonjour and peer-to-peer communication to provide an Internet connection and make users share their cache data with each other in the same local network.

Index Terms--Opportunistic network, Auto-proxy, Bonjour, Zero configuration networking, Peer to peer.

I. Introduction

Public transportation plays an important role. Using a shared passenger transportation service, people can solve several traffic problems such as a traffic flow. Meanwhile, with the rapid growth of the Internet, more and more people want to access new information from the Interent when they are using public transportation; however, it is difficult to provide Internet access everywhere. Hence, our objective is to develop an iOS application to share web page content in the cache of their devices.

In this paper we first define the system architectures: the network architecture and the application architecture. Section III describes each component of the application architecture. Section IV describes the setting of Xcode[1], which is the IDE tool from Apple.

II. Architectures

There are two architectures. One is for the network model and the other is for application components. The network model is based on creating a local netwok in each bus or train by a proxy node. At each station which has an access point proxy nodes can update and cache new content to satisfy passagers' demand. On the other hand, application components describe our iOS application in detail such as how to access local cache.

A. The Network Architecture

In each bus or train there is a local network that consists of a proxy node and serval peer nodes. A proxy node provides three services: DHCP service, HTTP proxy. Peer nodes access the Internet through the proxy node and they also run Bonjour[2] to discover services among peers. This network model is represented in the following figure.

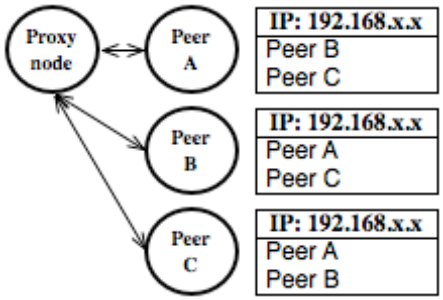

Figure 1. The network model

Bonjour is not only for detection of users, each peer also uses it to shares its cache index dynamically; therefore, users can directly browse each other's cache index to check whether there is interesting content or not.

When having a web page request, a peer may receive the response from three sources: its proxy node, its local cache, and other peer nodes. The proxy node is the first source a peer will ask to receive the latest content. A peer's local cache is the second, and other peers are the lowest priority for searching content. If no one has the content, the proxy

node will save the request and update its cache when arriving at a bus/train station.

B. The Application Architecture

Our iOS application consists of four levels and several components. Users will directly interact with three main components: a web browser, the local cache list, and the peer node list. The application architecture is the following:

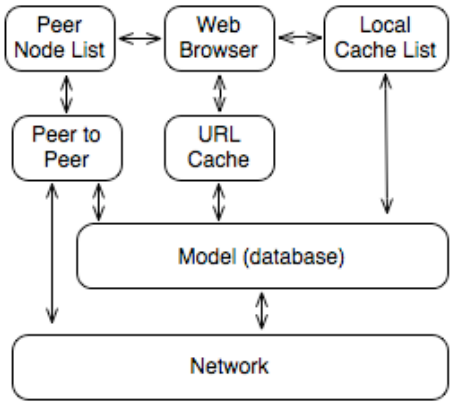

Figure 2. Application components

The web browser will communicate with the URL cache, which is an interface to manage cache data in iOS device. Before the web browser loads any URL request, it will check the URL cache first. The URL cache saves content into a database; therefore, even if users close this application, they will not lose the data. Moreover, the local cache list can directly communicate to the database and show the cache index. On the other hand when a URL request cannot be found in the database, this application will send a URL request to the proxy node or to the other peers. The peer list is similar to the local cache list, but it shows the cache list from other peers; therefore, it interacts with the peer-to-peer component that respond to all peer-to-peer communications.

III. Components

This section describes the application architecture in general; however, there are several technical problems in each component. This section will describe each of these problems and explains our solutions and implementations.

A. The Web Browser and the URL Cache

Since the web browser is built by UIWebView[3], it hides and protects the content of web pages. There is no way to take data out from an instance of UIWebView. However, because the web browser uses an instance of NSURLCache[4] to manage its cache data, the instance of NSURLCache will pass every URL request too. Hence it has become the perfect place to retrieve data. The following code show how to setup a customized NSURLCache.

```
cache = [[MyURLCache alloc] init];
[NSURLCache setSharedURLCache: cache];
```

Code 1. Setup customized NSURLCache

MyURLCache is the subclass of NSURLCache. After the creation of an instance of MyURLCache, the setSharedURLCache method sets it to become the shared URL cache for UIWebView. Since MyURLCache inherits from NSURLCache, the cacheResponseForRequest method is invoked by an instance of UIWebView, in order to check whether there is a appropriate cache data for URL request.

B. Model

This application follows Model-View-Controller[8] pattern in which model means data. We adopted CoreData[5] to build model component. CoreData is based on the managed object model[5]. It describes an object scheme. When an object into the database or it from the database, the object must follow the managed object model. Our managed object model, which is called Content, consists of context (which is binary data), mineType (which indicates media type), rank (which is the page hits count),

timeStamp (which is the date of insertion of the content) and url. The following figure shows managed object model.

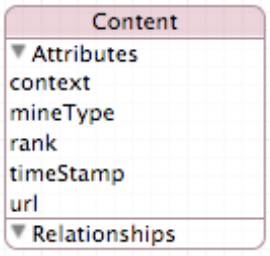

Figure 3. Content Model

C. Peer-to-Peer

Peer-to-Peer communication is one of the most important parts of this application. Bonjour, NSNetService[2] and NSNetServiceBrowser[2], detects each device and exchange cache index information. Every device will share its top five popular URL indexes.

The URL request and response between each peer are based on unicast socket communications. This application adopted event base methods, so the system will not be locked by a URL request. After the device receives a respond, it will refresh its web browser. The following figure is the time flow.

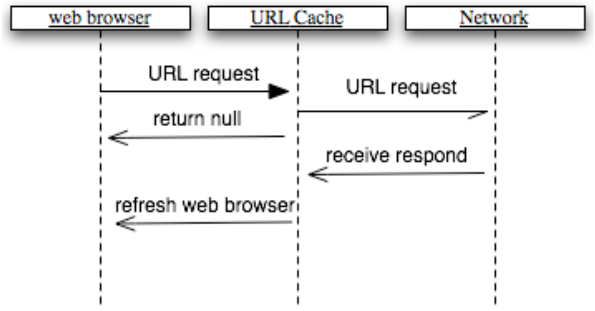

Figure 4. Time flow

In figure 4, after the web browser has sent a request to the URL Cache, if there is no cache content, it will send an asynchronous request and return null to web browser. After the URL Cache receives the response from other peers, it will refresh the web browser. In order to implement this event trigger, there are two sets of delegate protocols[9, 10], which is called interface in Java programming. The following code shows these two delegate protocols.

```
@protocol AsynNetworkDelegate <NSObject>
- (void) asynDownload: (AsynNetwork *)
  asynNetwork didFailWithError: (NSError *)
  error;

- (void) asynDownloaddidFinish:(AsynNetwork *)
asynNetwork;
@end

@protocol AsynModelDelegate <NSObject>
- (void) AsynModelDidInsertContent: (Content
  *) newContent;

- (void) AsynModelWillRequestContent:
(NSString *) reqStr;
@end
```

Code 2. Delegate Protocols

The cache manager must implement the AsynNetworkDelegate protocol to be triggered by socket objects. After the socket stream receives a response, the socket object will invoke its delegator's asynDownloaddidFinish method. The delegator, which is the cache manager, does two things in this method. First, it stores data from socket stream into the database. Second, the cache manager will notify its delegator, which is a web browser, that content has already been stored into the local cache.

```
- (void) asynDownloaddidFinish:(AsynNetwork *)asynNetwork {
    if (asynNetwork.data == nil) {
        return;
    }

    NetworkUtility *utility = [[NetworkUtility alloc] init];
    NSURL *url = asynNetwork.url;
    NSString *mineType = [utility
identifyMineTypeWithNSURL:url];
    [utility release];
    NSString *urlStr = [NSString stringWithFormat:@"%@://%@
%@", [url scheme], [url host], [url path]];
    Content *newContent = [self
insertNewObjectwithContext:asynNetwork.data mineType:mineType
url:urlStr];

    [asyDelegate AsynModelDidInsertContent: newContent];
    [asynNetwork autorelease];
}
```

Code 3. AsynDownloaddidFinish

The first three lines of asynDownloaddidFinish check whether data from socket objects are empty or not. And

then, from 4th line to 9th line store data into the database. The last second line invokes AsyModelDidInsertContent of its delegator, which is the web browser.

In fact the cache manager also will notify its delegator before it sends an URL request out. In order to receive these two notices, a web browser must implement AsynModelDelegate protocol. The AsyModelDidInsertContent will be invoked after data is stored into the database; the AsynModelWillRequestContent will be invoked before an URL request is sent.

After receiving a notice of that data are stored, the web browser will refresh its content. This may cause the web browser to refresh several times; therefore, we adopted a request pool to reduce the refresh. A web browser has a request pool. When AsynModelWillRequestContent is invoked, the web browser just put an URL into the pool:

```
- (void) AsynModelWillRequestContent:
(NSString *) reqStr {
    [downloadPool addObject: reqStr];
}
```

Code 4. Put an URL into the pool

Web browser will delete this request when AsynModelDidInsertContent is invoked. The web browser only refreshes, when the request pool is empty.

```
- (void) AsynModelDidInsertContent: (Content *) newContent {
    [downloadPool removeObject:newContent.url];
    if ([downloadPool count] > 0) {
        return;
    }

    Content *pp = [self.modelManager contentRequest:lastUrl];
    if (pp == nil)
        return;
    [self.webview stopLoading];
    [self.webview loadData:pp.context MIMEType:pp.miniType
textEncodingName:@"utf-8" baseURL:[NSURL
URLWithString:pp.url]];

    ...
}
```

Code 5. AsynModelDidInsertContent

The first line of AsynModelDidInsertContent removes the URL from the pool. And the second and third lines check if the pool is empty or not. If it is empty, the next eight lines refresh the web browser. By using this solution, we reduce refresh requests from 106 times to 36 times when loading the home page of the New York times.

IV. Setting

We used the current version of Xcode, which is Xcode 4.0, and the latest SDK, which is SDK 4.3. This application used six libraries from Cocoa framework: Foundation, UIKit, CoreGraphics, CoreData, CFNetwork, and SystemConfiguration. The first three libraries are required by any iOS application, and the socket programming requires last two libraries.

There are 30 source code files. They are represented in the following figure.

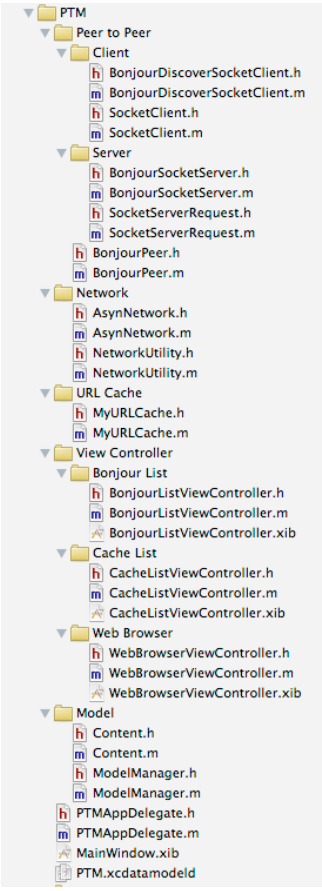

Figure 5. Source code files

PTMAppDelegate is the entry of the application. There is a UITabBarController[6] including three view controllers: Bonjour List (BonjourListViewController), Cache List (CacheListViewController), and Web Browser (WebBrowserViewController). MyURLCache is the implementation of the URL Cache component. The object model scheme is described in PTM.xcdatamodeld[5] and its counterpart, which is the source code of object model, is Content. ModelManager is the implementation of the model component.

There are two parts in the Peer to Peer group: Server (BonjourSocketServer responds to set up NSService and a unicast socket[7]) and Client (BonjourDiscoverSocketClient responds to set up NSServiceBrowser). SocketServerRequest will handle URL request from other peers, and its counterpart, SocketClient, will send a URL request and handle the responses from the peer who has an appropriate content. BonjourPeer is a map linking a peer's NSService and its cache list.

The network group has NetworkUtility (which are some utilities such as mineType detection) and AsynNetwork (which creates asynchronous download connections).

V. Conclusion

In this paper, we show an iOS application that can cooperate with a proxy node to make users share their cache content. This application also provides offline web browser functionality.

VI. Reference

[1] Xcode: https://developer.apple.com/xcode/

[2] Introduction to NSNetServices and CFNetServices Programming Guide: https://developer.apple.com/library/ios/#documentation/Networking/Conceptual/NSNetServiceProgGuide/Introduction.html

[3] UIWebView Class Reference: https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIWebView_Class/Reference/Reference.html

[4] NSURLCache Class Reference: https://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSURLCache_Class/Reference/Reference.html

[5] Introduction to Core Data Programing Guide: https://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CoreData/cdProgrammingGuide.html

[6] Tab Bar Controller: https://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/TabBarControllers/TabBarControllers.html

[7] Introduction to Stream Programming Guide for Cocoa: https://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/Streams/Streams.html

[8] Model-View-Controller Design Pattern: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html%23//apple_ref/doc/uid/TP40002974-CH6-SW1

[9] Cocoa Core Competencies--Delegate: http://developer.apple.com/library/ios/#documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html

[10] Cocoa Design Pattern--Chain of Responsibility: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html