Internet Cache in Opportunistic Networks

Taotao Li

Dept. of Computer Science, Columbia University

New York, NY 10027

tl2453@columbia.edu

ABSTRACT

I made several improvements based on the former version. The improvements include removing the hard-coded part in the source files, importing the proxy auto-configuration and porting the project to Soekris net5501.

1. INTRODUCTION

1.1 Overview of Web Cache Project

The system works between a client and an AP. The basic idea is that a proxy node caches the web content when Internet connection exists. When a client asks for the web content, a proxy node checks the Internet connection and try to get the web connect which the client has asked for. Using a local cache, the proxy node caches requested content. Simultaneously, it provides the client a list containing the web content already cached by proxy node. The client can browse the cached content he or she is interested in. If the proxy node successfully caches the web content the client originally asked for, it pushes a notification on the client's browser. Hence, the client can retrieve the content.

1.2 The improvements

There are mainly three improvements based on the previous version:

1. Created the project configuration file, and make the parameters such as server directory and IP address configurable.

2. Imported proxy-auto configuration approach to simplify the configuration task in the user side.

3. Installed OS, wireless driver in Soekris net 5501. And also ported the project code to Soekris net 5501.

2. Architecture

2.1 System architecture

The proxy node works between a client and an AP. When a client sends the request to our proxy node, it checks the Internet connection. If it exists, the proxy node fetches the web content. But if the Internet connection is not available at that time, our proxy node provides the list of cached web content, and then the client can browse the cached web content. The proxy node puts the request in request queue, so if the Internet connection is available later on, the proxy node fetches it and also push the notification on the client browser. The proxy node makes the corresponding adjustment for storing the web content in local file system, so the client can browse the correct and completed web content.

2.2 Architecture of proxy auto-configuration

I implemented the feature proxy auto-configuration in our Web Cache Project, so the client does not need to manually set up the proxy such as an IP address and a port number. Our proxy auto-conf is based on the Web Proxy Autodiscovery protocol [1] [2] [3]

Note that this WPAD protocol is not a standard protocol, which means that not all browsers would support it. However, I have proved that it works in IE 8 and 9, Firefox, Chrome and Safari.

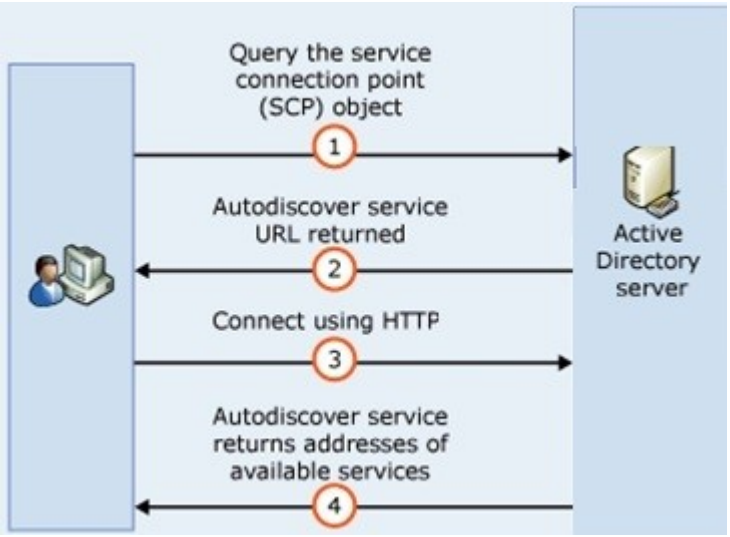

Figure 1

The basic idea of WPAD is that the proxy node includes the related proxy information in a specific file (.pac file). Hence the proxy information such as IP addresses and port number is recorded in this file. The proxy node puts .pac file in the directory of web server, so the client can fetch the file via corresponding URL. The proxy node needs to combine such URL information in some specific the response to the client. For example, when the client sends the DHCP request or DNS request to the proxy node, the proxy node combines the URL of .pac file in its DHCP/DNS response. With the support of browsers, the client fetches such file and parses the content, and then setup the proxy automatically. By such approach, the client does not need to manually specify the proxy configuration anymore.

3. Details of Web Cache Project Code

OppNetCache.java

It includes the most important functions in the project. It implements a separate thread and runs as a daemon to listen and wait for clients' requests.

isInternetReachable() Function

The isInternetReachable () function is called from the function run() which runs as daemon. The function tries to check the Internet connection by resolving a common website. In the original version, the interface name for Internet connection must be specified. So I rewrote this function by removing such restriction. Hence the user does not need to specify such information anymore.

OppnetHelper.java

This file retrieves the information in the project configuration file. It also tells from the network card for Internet connection from the card for ad-hoc connection.

loadOppnetConf()Function

This function locates the project configuration file mentioned above and loads the corresponding parameters.

analyzeNetworkCard()Function

This function is used to analyze the usage of network card. Basically, the network card whose ESSID contains special words (which can be defined by user in our project configuration file) are treated as ad-hoc connection card.

getIpAddressByCardName()Function

The function is responsible for returning the corresponding IP address by its network card name.

OppnetCache.java, OppnetFilter.java, Server.java, ShowStatus.java and ShowNothing.java

I made the corresponding adjustments so that the "hard-coded" part can be removed from our source code. It includes tomcat directory, port number and network card for Ad-hoc connection.

4. How to build and run the project

4.1 Requirement of Web Cache Project

To build and install Web Cache Project, the hardware and software should satisfy the following requirements:

1. Make sure you have two wireless network interface cards on your machine, so you can use one to set up an ad-hoc network and use the other to connect to access point (AP).

2. Make sure you have installed JDK 1.6

3. Make sure you have installed Tomcat 6.0. For Linux machine, you should follow the default installation directory, i.e. under the path "//user/local/tomcat/"

4. For development purpose, you also need to install Eclipse 3.6, but this is not necessary if you just want to run our application

5. If you want to install Web Cache Project on windows machine, you need to install windows SDK to identify the network connection and modify some source path, this document is for Linux system.

4.2 Steps for run Web Cache Project

Step 1 Set up an Ad Hoc network on your proxy server machine, and manually set its IP address for Ad Hoc to *10.42.43.8*. Use a client machine to join this Ad Hoc network.

Step 2 Unzip and open folder ***OppNet_src_v3***.

Step 3 Copy folder ***CacheFiles*** and ***StatusBar,*** and then paste them under directory "/usr/local/tomcat/webapps/" as shown below.

Step 4 Open a prompt window and start up Tomcat using the following command:

```
oppnet@irt:~$ cd /usr/local/tomcat/bin

oppnet@irt:/usr/local/tomcat/bin$ ./startup.sh restart
```

Step 5 Follow the instructions in the source directory to install and configure the servers.

Step 6 Set the path to "OppNet_src_v3/muffin-0.9.3a/src" and start up Muffin.

```
oppnet@irt:~$ cd /home/oppnet/OppNet_src_v3/muffin-0.9.3a/src
```

Step 7 Run Muffin using the following command:

```
oppnet@irt:/home/oppnet/OppNet_src_v3/muffin-0.9.3a/src$ java Muffin
```

Step 8 The Muffin is running now and you can monitor what's happening on the proxy server side. On your client machine, open a web browser and it should automatically setup the proxy.

Step 9 Use Firefox to send an HTTP request for a webpage. And you will see on proxy server side that this request is in the queue right now. If the proxy server side has network connection to AP, it will send out the request and cache the response then push notification to client, if it doesn't, it will keep checking network status.

Step 10 To stop Muffin, press key Ctrl+C; to shut down tomcat, use command ./shutdown.sh

5. The steps to install Ubuntu on Seokris net 5501

5.1 Install OS

These instructions describe how I installed Ubuntu 10.04 on Soekris net5501[4] by using debootstrap to build a CF card on a host system running Ubuntu 10.04. I need to use Minicom to capture the output of net5501, and I need to set its speed rate to 115200 and use "minicom –s –n on" to run minicom.

1. Partition the CF card and mount the target / filesystem at /mnt/target.

   ```
   sudo mkdir /mnt/target
   sudo cfdisk /dev/sdb
   sudo mke2fs -j /dev/sdb1
   ```

sudo mount /dev/sdb1 /mnt/target

2. Mount the installation ISO

   sudo mkdir /mnt/iso

   sudo mount –t iso15200 –o ro.loop=/dev/loop0 /home/oppnet./ubuntu 10.04-server-i386.iso /mnt/iso

3. Run debootstrap

   sudo apt-get install debootstrap

   sudo debootstrap –arch i386 feisty /mnt/target file:/mnt/iso

4. Chroot into the target

   sudo chroot /mnt/target /bin/bash

5. Configure keyboard

   dpkg-reconfigure console-setup

6. Setup a nonroot user

   adduser foo

   echo 'foo ALL=(ALL) ALL' >> /etc/sudoers

   chmod 0440 /etc/sudoers

7. Create the file /etc/event.d/ttyS0:

   start on runlevel 2

   start on runlevel 3

   start on runlevel 4

   start on runlevel 5

   stop on runlevel 0

   stop on runlevel 1

   stop on runlevel 6

   respawn

   exec /sbin/getty -L ttyS0 115200 vt102

8. Edit the file /etc/initramfs-tools/modules and add the following two lines at the end of the file:

   ext3

   ide_generic

   Then, run update-initramfs –u

9. Install grub and linux generic kernel

   apt-get install linux-image-generic grub memtest86+

   mkdir -p /boot/grub

   cp /usr/lib/grub/i386-pc/* /boot/grub

   editor /boot/grub/menu.lst

   exit

10. Run the command to finish the installation.

    # Run this from outside the chroot()

    sudo grub-install --no-floppy --root-directory=/mnt/target /dev/sdb1

11. unplug the CF card form your laptop and insert it to the Soekris net 5501.

5.2 Build the wireless driver to linux kernel

Since Ubuntu Server 10.04 cannot recognize the wireless driver successfully, I need to manually download the source code and compile it.

1. Install the essential build tool.

   Sudo apt-get install build-essential bin86

2. Download the Madwifi source code [5]

3. Extract the Madwifi source code

   Tar –xzf madwifi-0.9.4.tar.gz

4. Download the linux source code I are using

   Sudo apt-get install kernel-headers-$(uname -r)

5. Go to the directory of Madwifi and run

   Sudo make clean

   Sudo make

   Sudo make install

   This takes around half an hour to compile the source code, make sure there are no errors in the output screen.

6. Make the kernel to probe the module automatically

   Echo ath_pci > /etc/modules

7. Reboot the box and the wireless card should be recognized successfully.

5.3 The steps to port the code to Seokris net 5501

1. Install dhcp server

   Sudo apt-get install dhcp3-server

2. Install dns server

   Sudo apt-get install bind9

3. Install apache server

   Sudo apt-get install apache2

4. Copy the corresponding configuration file to the right directory, follow the instructions in ReadMe file.

5. Install tomcat server

   Download the zip version of tomcat and extract it to /usr/local/ directory

6. Install ssh server

Sudo apt-get install ssh

7. Connect to the box via ssh, and I can run the program by "ssh –X foo@x.x.x.x" (x.x.x.x is the IP address of box which I can connect via network)

6. Conclusion

By removing the hard-coded parts in the previous source code, the current version is more configurable and flexible. By adding the proxy auto-configuration, the client browser can automatically set up the proxy. So the user does not need to manually set up the proxy parameter. Also, I successfully installed OS on Seokris net 5501 and set up the WLAN driver.

7. Reference

[1].http://en.wikipedia.org/wiki/Web_Proxy_Autodiscovery_Protocol

[2].http://en.wikipedia.org/wiki/Proxy_auto-config

[3].http://www.itbully.com/articles/auto-configuring-proxy-settings

[4]. http://soekris.com/products/net5501.html

[5].http://madwifi-project.org/