ANGEL Client Manager Software Architecture Design Document

Jason But
Centre for Advanced Internet Architectures, Technical Report 070228E
Swinburne University of Technology
Melbourne, Australia
jbut@swin.edu.au

***Abstract*—The Automated Network Games Enhancement Layer (ANGEL) project aims to leverage Machine Learning (ML) techniques to automate the classification and isolation of interactive (e.g. games, voice over IP) and non-interactive (e.g. web) traffic. This information is then used to dynamically reconfigure the network to improve the Quality of Service provided to the current interactive traffic flows and subsequently deliver improved performance to the end users. Within this scope, the project will develop protocols that allow the adjustment of Consumer Premise Equipment (CPE - eg. cable/ADSL) configuration to provide better quality of service to interactive flows detected in real-time.**

This document describes the basic design motivation of the Client Manager Software Component of ANGEL. The Client Manager is responsible for managing registration and de-registration of individual ANGEL-enabled CPE devices and then delivering flow classification information from the Flow Classifier to all registered users that are routing those flows.

I. Introduction

This document details the software design decisions made when building the ANGEL Client Manager. In particular we discuss how the design takes scalability into account as well as looking at how functionality can be provided to manage concurrent tasks of registering CPE devices within the system and forwarding notification to the end CPE devices. We also consider how the software is designed to meet other requirements of the Client Manager within the scope of the ANGEL Project.

II. Flow Classifier Requirements

We start by repeating the minimum requirements and tasks from the ANGEL Architecture document. These provide a useful reference when considering any specific constraints in the software architecture.

An ANGEL System consists of a single Client Manager. System scalability **SHOULD** be implemented either as a multiple parallel ANGEL systems or through the use of load balancing within a cluster of machines.

The primary tasks of a Client Manager are:

- Accept connections to the ANGEL System from ANGEL enabled consumer routers
- Maintain the details of registered users in the ANGEL Database
- Manage the re-registration and de-registration of ANGEL enabled consumer routers from the system
- Signal all appropriate routers upon detection of flow classification changes by the Flow Classifier

Key design restrictions for all aspects of the ANGEL Client Manager are:

- Configured with (either locally or from the ANGEL Database):
  - IP Address of the ANGEL Database
  - Registration and Flow Notification timeout values
- Store state information for currently registered clients and which range of IP Addresses they are willing to receive Flow Notifications about
- Use the formally specified ANGEL CPE/Client Manager Protocol [1] to allow implementation of a multi-party system - in particular to allow for third-party implementation of ANGEL enabled CPE devices
- Filter flows to only communicate Flow Notifications to current ANGEL enabled users
- Upon receiving notification that a flow has terminated, send the appropriate notification to the CPE device(s) AND remove the flow information from the ANGEL Database

From February 2007 and July 2010 this report was a confidential deliverable to the Smart Internet Technologies CRC. With permission it has now been released to the wider community.

A. Key Restrictions

The Flow Classifier must communicate with ANGEL-enabled CPE devices using the formally specified ANGEL Protocol [1]. The idea is to allow for a multi-party system implementation. It is expected that the majority of third-party devices will be present in the form of ANGEL-enabled CPE devices and this device is both bought and maintained by the user. It is also possible that a third-party may choose to create a new implementation of the ISP-side ANGEL components, in this case the Protocol allows for use with pre-existing ANGEL-enabled CPE devices.

It is not expected that the Client Manager will suffer from a heavy load. The Flow Classifier must classify all flows, the Client Manager need only communicate changes in classification state to registered users. As such both the generated network load and memory requirements are orders of magnitude lower than that required by the Flow Classifier devices.

B. System Configuration

The Client Manager must run using some configuration options. The ANGEL Architecture allows for configurable options to be stored in the database. Even so we must also allow for configuration options to be stored locally on the Client Manager. Further, some configuration options must be known prior to using the database.

1) Local Configuration Information: The Client Manager should use a local configuration file which stores:

- Database Contact Information - Details required to contact and use the external database. This could include details such as IP address, username, password, etc.

2) Database Configuration Information: Other configuration information **must** be stored in the database. This includes:

- Client Registration Timeout - Maximum period before a CPE device must re-register before being considered to have been switched off
- Flow Notification Timeout - Defines the period where the presence of no traffic (at the CPE device) for a specified flow indicates that the flow has expired and that any prioritisation rules should be removed by the Client. Implements a fail-safe where old rules are eventually cleared if not specifically removed by notifications from the Client Manager.

C. Multi-Threaded Design

The Client Manager is implemented as a multi-thread design to take full advantage of processing capabilities while other parts of the code are potentially blocked waiting for system resources.

All threads need to be able communicate data amongst in other. In threaded applications, this is typically done through the use of shared memory and variables. Problems occur when two threads need to access the same memory block at the same time. This situation is typically solved via the use of Mutual Exclusions and Semaphores which cause some threads to block while protected code is being executed.

Common difficulties with this approach arise because:

- To ensure that we have thread-safe code we need to wrap as much as possible inside a protected space. This ensures that we don't *miss* a shared variable modification which could cause the code to misbehave
- To develop an optimal/efficient system, we need to wrap as little as possible inside a protected space. This ensures that code is only blocked from executing when it is absolutely necessary, maximising CPU usage

D. Flow Hash Translation

The Flow Classifier communicates Classification State changes to the Client Manager, these flows are identified using the Flow Hash initially generated by the Flow Meters. The Client Manager **MUST** be able to determine the flow identification tuples - source/destination IP address, source/destination port, protocol - from the Flow Hash. This information should be stored in the database by the Flow Meter when the Hash was initially generated.

As previously mentioned, the load on the Client Manager is not expected to be high, the rate of Classification State Changes (see Flow Classifier Software Design Document) being infrequent. As such, the rate at which the external Database needs to be queuried for this information should not be excessive.

E. Development Tools

The Flow Meter makes use of the ACE library/toolkit. We use this toolkit to provide C++ wrappers around common network functionality (sockets) as well as to provide tools to manage thread creation, thread management and inter-thread communication.

III. ACE RADIUS STACK

The packet format in the protocol used by the Client Manager to talk to ANGEL-enabled CPE devices is based on the RADIUS protocol. To facilitate implementation of the Client Manager and ensure that minimal bugs are generated in implementing the Protocol Stack within the Client Manager, we make use of a pre-existing RADIUS implementation for ACE.

"ACE RADIUS Library" is an implementation of the RADIUS Protocol within the ACE framework and is used to manage multiple connections to a server and format packets for transmission [2]. The Library is distributed with a BSD license [3] and as such we are able to re-use it within the ANGEL code base.

However, the ANGEL Protocol is not a direct extension of RADIUS, but rather a Protocol that shares much in common with RADIUS [1]. This dictates that rather than compile and use the provided ACE RADIUS Library as a basic library, we must take and slightly modify the implementation to perform as required for the purposes of ANGEL.

The code of the ACE RADIUS Library modified for use in ANGEL can be found in the *"protocollib"* source directory which compiles to a library to be linked with the Client Manager code-base. Changes were made to v0.8 of the original library.

A. Features of the ACE RADIUS Library

The *"ACE RADIUS Library"* is an add on for the ACE library that is complian with RFC 2865 (Remote Authentication Dial In User Service (RADIUS)) [4] and implements all features described tehrein. The library contains code to allow for the implementation of both the Client and Server side of RADIUS programs.

The key features of the library include classes to produce and validate RADIUS packets and to handle packet re-transmission and failures (with associated timers). The library also has a number virtual callback functions to allow packet parsing and creation at the application level.

B. Important Classes

The following classes can be found within the modified ACE protocol library:

- **ManagementPacket** - Provides packet definitions and functionality to create and parse ANGEL Management Packets
- **ManagementPacketAttribute** - Provides definitions of ANGEL packet attributes and the functionality to create and parse the attributes
- **ServerConnection** - Handles network communications. Used to receive and send management packets. The class also maintains the state of unacknowledged packets and handles their retransmission. This class is used by the Client Manager
- **ServerStack** - Provides a number of virtual callback functions which are overloaded in the Client Manager. These functions relate to sending and handling received packets
- **ClientConnection** - Handles Client Interface network communications. Creates a connection handle to a server and then listens for and transmits management packets. This class is used by the ANGEL Client
- **ClientStack** - Provides a number of virtual callback functions which are overloaded in the ANGEL Client. These functions relate to sending and handling received packets
- **CTask** - Implements timers that are primarily used for packet re-transmission

This section describes the major classes of the ANGEL Protocol Library and their functionality within the Client Manager.

C. Modifications

A number of modifications to the original RADIUS library have been made to suit the operation of the ANGEL system. The principal modification has been the redefinition of the RADIUS Packet Types and Attributes as per the ANGEL Protocol [1]. In addition, two new classes (**ServerConnection** and **ServerData**) have been added to the library to provide expanded management methods for the Client Manager (handling various new packet types, maintaining client state etc):

These classes are based on the original **ClientConnection** implementation with a number of modifications to the packet-parsing and network socket methods. For example, in the modified ANGEL implementation of the ACE RADIUS Library a Server (Client Manager) uses the **ServerConnection** class to send and receive packets, whereas in the original implementation **ServerStack** is used to transmit packets.

IV. CLIENT MANAGER CODE BASE

The multi-threaded implementation of the Client Manager server is handled by the ANGEL Protocol Library - an extension of the RADIUS implementation for ACE (see Section III). Application dependent processing is managed by inheriting from the **CServerStack** class.

Essentially the Client Manager code base reads some basic configuration values before creating an instance of the **ManagerServerStack** and the **ServerConnection** classes. The underlying library code implements the multi-threaded server while the primary code-base uses the **ClientMap** and **ManagerServerStack** classes to keep track of individual clients and communicate flow notification updates to appropriate clients.

A. *ClientMap*

This helper class is used to map IP addresses to instances of the **Client** class. The **Client** class is used to store per-client information such as:

- IP Address and Port Number to send Flow Notifications to
- Time of last registration for timeout purposes

The class implementation provides a thread-safe database allowing a lookup of client connection information to the multi-threaded Client Manager implementation. Member methods allow determination if a client exists in the map as well as for addition and removal of client data from the map.

B. *ManagerServerStack*

This class is virtually inherited from the **CServerStack** class - which is implemented in the ANGEL Protocol Library. Extensions in this class manage parsing of incomming packets and transmission of Flow Notifications to individual ANGEL CPE devices. The primary implementation of the ANGEL Client Manager is based in the Protocol Library.

The class implementation makes use of the **ClientMap** class to extract and update information about individual clients currently connected to the Client Manager.

ACKNOWLEDGMENT

This work was supported from 2005 to early 2007 by the Smart Internet Technology Cooperative Research Centre, http://www.smartinternet.com.au.

REFERENCES

[1] J. But, "ANGEL Protocol - CPE/ISP Protocol Document," CAIA, Tech. Rep. 070228B, January 2007, http://caia.swin.edu.au/reports/050204A/CAIA-TR-050204A.pdf.

[2] Alex Agranov, "ACE RADIUS Library," January 2007, http://ace-radius.sourceforge.net/.

[3] Open Source Initiative, "The BSD License," January 2007, http://www.opensource.org/licenses/bsd-license.php.

[4] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," IETF, RFC 2865, Jun. 2000, http://www.ietf.org/rfc/rfc2865.txt.