

Implementations of automated testing environment for distributed multimedia streaming applications

Wook Hyun*, Ju Young Park*

*ETRI (Electronics and Telecommunications Research Institute), Korea

whyun@etri.re.kr, jypark@etri.re.kr

Abstract — Nowadays, multimedia streaming service is one of the widely used application over Internet. However, it costs much and needs infrastructures to accommodate massive concurrent users in server-client model. Hence, there several solutions and standards are under development based on peer-to-peer networking, and we are implementing prototype based on ITU-T Recommendation. On testing application in distributed systems, operator has to control whole client one-by-one manually on every test scenario, and it was not easy to test time critical scenarios. Hence, we have constructed our own testing environment by using several open source solutions. In this paper, we share our experiences on constructing automated testing environment for distributed multimedia streaming application.

Keywords — automation, testing environment, distributed application, multimedia streaming

I. INTRODUCTION

For multimedia services, server-client model has limitations, such as bottleneck and server failure, on accommodating massive number of concurrent users. It also costs much due to network bandwidth and maintenance of streaming server systems. In order to solve this situation, and provide scalable streaming services, many service providers that do not have their own network infrastructure begin to provide P2P-based streaming services, since it is one of the most efficient method for contents providers. In standardization aspect, ITU-T Q9/11 is developing related Recommendation, and IETF PPSP WG has developed PPSP protocols [1][2].

On developing and testing distributed application such as P2P-based multimedia streaming client, the client needs to be installed over multiple systems. Of course, it is possible to install multiple virtual machines within a system, but it lacks of system performance to run multiple instance at the same time. Furthermore, if the client is GUI application, it is not easy to automate the testing sequence in virtual environment, and to control whole clients one-by-one in every steps. Hence, we decided to construct own automated testing environment for distributed multimedia streaming systems.

On designing automated testing environment, we have listed several requirements for easy deployment and testing, and surveyed appropriate tools available for our purpose. For easy deployment of testing platform, we have dockerized a control server that manages testing unit and provides RESTful API for

delivering configuration to each streaming client application. For controlling distributed client, we made a console terminal to control them within a system using Node.js. This console application retrieves the addresses of client to be tested, and sends command signal to each client. In each client system, it has a TCP server that receive commands and initiate appropriate operation to the client application. On controlling custom GUI application, we have used *AutoHotKey*^[3] that hooks and generates mouse and keyboard event for manipulate the client.

In this paper, we briefly describe our P2P-based streaming client in clause 2, and lists up requirements of testing environment in clause 3. In clause 4, we describe how we implemented core components for the testing environment in details, and development/testing procedures is described in clause 5. This paper is concluded in chapter 6 with future works.

II. UPREP: DISTRIBUTED MULTIMEDIA STREAMING CLIENT

In order to make use of advantages of P2P-based contents delivery, ITU-T has developed ITU-T X.609^[4], X.609.1^[5], X.690.2^[6] based on ISO/IEC TR 20002^[7]. Based on these standards, ITU-T Q.9/11 is also developing X.mp2p-mssr^[8] and X.mp2p-mspp^[9] for multimedia streaming services.

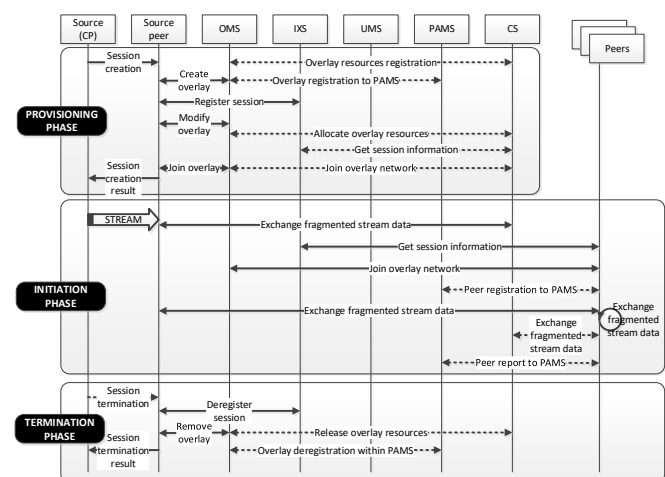


Figure 1. Overview of streaming procedure of ITU-T X.mssr

Figure 1 shows an overall procedure for multimedia streaming over managed P2P networks, and we have been

implementing prototype of streaming client based on ITU-T X.mp2p-mspp and X.mp2p-mssr as shown in Figure 2.



Figure 2. uPREP player

In the previous work ^{[10][11]}, we have implemented PREP protocol and client that can provide streaming services with HD quality. We are now enhancing the quality to support UHD with 4K/8K video in uPREP. Since the size of contents and bitrate are quite larger than HD streaming in UHD, it needs pretty much modifications to enhance the protocol and its behaviors. Since there are many interactions among related components, and these are repeated on multiple client, it is not easy to operate the applications one by one by manually for every developing sequences. Hence, we made an automated testing environment for distributed multimedia streaming clients.

III. REQUIREMENTS OF TESTING ENVIRONMENT

Prior to construct testing environment, we have considered several existing solutions with following requirements to be satisfied.

- Support of easy deployment and management of testing environment
- Support of programmable control of Windows GUI application
- Support of easy and fast distribution and development in any testing system

A. Easy deployment and management of testing environment

For easy deployment and portability of testing environment, it needs to be platform independent. Hence, we considered various container solutions including Vagrant ^[12], CoreOS ^[13] and Docker ^[14].

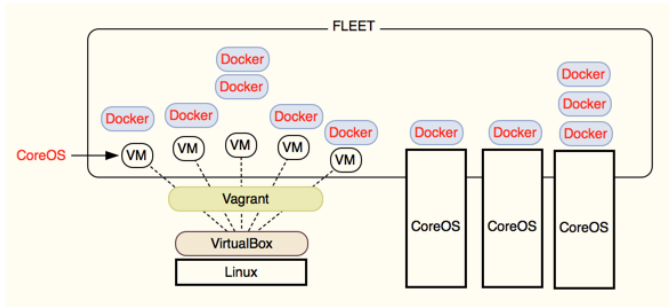


Figure 3. Docker containers over CoreOS using vagrant

1) CoreOS, Vagrant

Vagrant is an open source application that configures and manages virtual instances that are running on various

virtualization environment such as VirtualBox, VMware, KVM, LXC and docker. This software makes it efficient and consistent management interface on handling virtual instance. CoreOS is one of Linux distribution that is specialized for supporting docker containers on multiple computers. When a process is killed or a computer fails, other computer initiates the process for service continuity. We have considered to using vagrant with CoreOS as show in Figure 3.

2) docker-swarm and docker-compose

Docker supports initiation of multiple container with dependency among them easily with 'docker-compose' command. Also, it is also possible to make use of multiple docker host system by use of 'docker-swarm' command. In constructing testing environment, since we do not need service continuity or server scalability, we decided to use docker-compose only.

B. Automated programmable control of Windows GUI application

Since uPREP client is Windows GUI application, it needs arbitrary programmable way to control our application with customized triggering of mouse/keyboard event. There are several tools that records and generates mouse/keyboard event, but the application to be tested can be located in any location of screen. Hence, we have chosen *AutoHotkey* that makes it possible programmable event triggering.

C. Easy/Fast distribution/development in any machine

In development and testing phase, it needs to support debugging in any testing system, and also needs to support of distribution of modified binaries in any system. We constructed our own Git repository to debug in any machine with version control. In order to distribute latest binary, each system established samba shared folder. When the latest binary is located in samba shared folder, other systems copy them to their local folder on receiving request from console terminal.

IV. IMPLEMENTATION OF CORE COMPONENTS FOR AUTOMATED TESTING ENVIRONMENT

The automation testing environment is mainly consisting of three components (RCS, RCT and CUT), and Figure 4 shows an overall topology for testing.

- **RCS (Remote control server):** This server maintains configuration information of registered ET with the database. This server is dockerized to make it easy to deployment and maintenance.
- **RCT (Remote control terminal):** This terminal application interacts with RCS to retrieve the list of ET and configuration files of each ET.
- **CUT (Client under test):** This end terminal opens a TCP server instance to receive commands from RCT, and control the behaviour of uPREP client application.

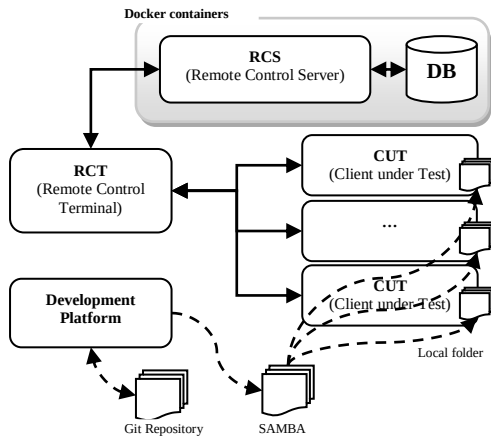


Figure 4. Topology of automated testing environment

A. RCS (Remote control server)

RCS is implemented using Node.js with express^[15], and is dockerized as shown in following *Dockerfile* and *docker-compose.yml* of table 1. This server maintains configuration information of each CUT, and provide them with RESTful interface.

TABLE 1. CONFIGURATIONS FOR RCS SERVER DOCKERIZATION

<i>Dockerfile</i>	<i>docker-compose.yml</i>
<pre>{ "name": "rcs", "version": "1.0.0", "dependencies": { "express": "~4.14.0", "ejs": "~2.5.1", "body-parser": "~1.15.2", "express-session": "~1.13.0", "mysql": "~2.11.1", "net": "~1.0.2" } }</pre>	<pre>version: '2' services: mp2p-mysql: container_name: "mp2p-mysql" build: ./mysql restart: always ports: - "3306:3306" volumes: - "/mysql/db:/var/lib/mysql" rcs: container_name: "rcs" build: ./rcs ports: - "7000:7000" depends_on: - mp2p-mysql links: - mp2p-mysql</pre>

RCS manages configuration information of registered peers, and peer can request their configuration on application initiation. Peer can register and de-register itself using RESTful interface, and retrieve their configuration information through the interface as well.

B. RCT (Remote control terminal)

This application is also implemented using Node.js, and receives command from standard input command line interface from operator, and sends command to all registered CUT. It supports following command set as shown in table 2.

TABLE 2. COMMAND SET FOR RCT

Command	Description	Dest
<i>list</i>	This command is used to retrieve the list of ET to be tested.	RCS
<i>start</i>	This command is sent to all ETs to initiate uPREP client application.	CUT
<i>fcopy</i>	This command is used to update binary of uPREP client application.	CUT
<i>stop</i>	This command is sent to all ETs to stop playing streaming session, but does not terminate the application.	CUT
<i>kill</i>	This command is sent to all ETs to terminate uPREP client application.	CUT
<i>full</i>	This command is sent to all ETs to turn full screen mode for displaying streams.	CUT

C. CUT (Client under test)

CUT receives command from TCP server that is implemented with using *AHKsocket_unicode.ahk*^[16], and dispatches appropriate scripts to be executed for the command.

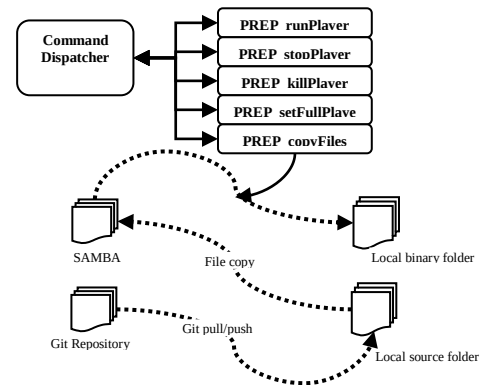


Figure 5. Procedures for distributing latest binary

Figure 5 shows the procedures for distributing up-to-date binary to each CUT. Each CUT clones up-to-date source code from git repository, and locates latest binary to samba folder after compile. On receiving 'fcopy' command from RCT, CUT executes *PREP_copyFiles* script that copies pre-designated folder to local execution folder.

TABLE 3. SCRIPTS FOR TEST AUTOMATION

Command Script	Descriptions
<i>PREP_runPlayer.ahk</i>	This script runs uPREP player and join multimedia session at the same time.
<i>PREP_stopPlayer.ahk</i>	This script stops multimedia streaming session, but does not kill application.
<i>PREP_killPlayer.ahk</i>	This script kills uPREP player completely. On updating latest binary, it needs to be executed prior to <i>PREP_copyFiles</i> .
<i>PREP_copyFiles.ahk</i>	This script copies latest binary files to designated local path from shared samba folder. On copying those files, it uses <i>FileCopyDir</i> command of <i>AutoHotKey</i> .
<i>PREP_setfullPlayer.ahk</i>	This script turns the application to full screen mode. If it is already full screen mode, it will toggle to window mode.

Since resolution of each CUT may be different, it needs to recalculate the coordination of event to be triggered based on the resolution of CUT systems.

V. DEVELOPMENT AND TESTING PROCEDURES

By using the automated testing environment, we can modify the source codes and debug in any system with uniformed way. As shown in Figure 6, each system maintains latest source code by using git repository, and push latest source code to samba server. If the latest binary is located in the shared folder, RCT initiates command to distribute the binary to each CUT. On receiving control command from RCT, all CUT automatically perform pre-designated scripts.

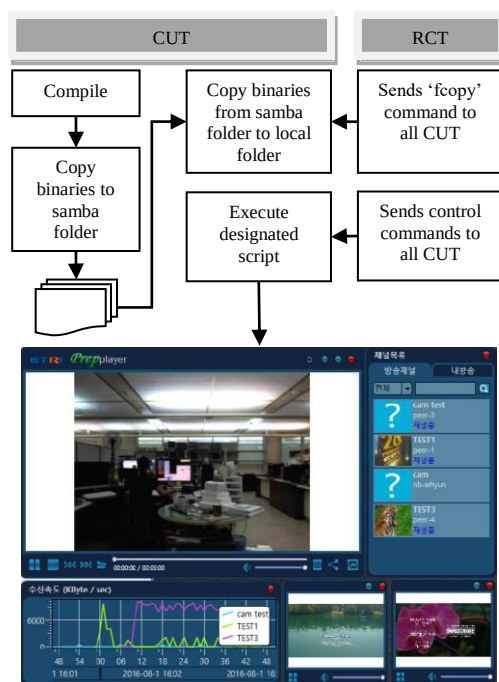


Figure 6. Procedures for development and testing

VI. CONCLUSIONS

In this paper, we described uPREP player that supports distributed multimedia streaming based on ITU-T draft Recommendation, and the necessity of automated testing environment for distributed application over multiple systems. This paper explains about how we implemented automated testing environment using open source technologies including Node.js, AutoHotKey and docker. For describing the testing environment, we have defined several logical components such as RCT, RCS and CUT with explanations on how those operates. By use of this environment, we can test uPREP player application within a single terminal without manipulating multiple systems manually. It also makes it possible to test timely critical testing scenarios with modifications on the scripts. The source codes that have been used in this paper are downloadable at GitHub [17].

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0127-16-1053, Development of standards for distributed UHD streaming protocol).

REFERENCES

- [1] *Peer-to-Peer Streaming Peer Protocol (PPSPP)*, IETF RFC7574, 2015
- [2] *Peer-to-Peer Streaming Tracker Protocol (PPSTP)*, IETF RFC7846, 2016
- [3] *AutoHotKey*, <https://autohotkey.com/>
- [4] *Managed P2P Communications: Functional architecture*, ITU-T Std. X.609, 2014.
- [5] *Managed P2P Communications: Peer activity management protocol*, ITU-T Std. X.609.1, 2016.
- [6] *Managed P2P Communications: Overlay resource control protocol*, ITU-T Std. X.609.2, 2016.
- [7] *Managed P2P: Framework*, ISO/IEC TR 20002, 2013
- [8] *Managed P2P Communications: Multimedia streaming signaling requirements*, ITU-T Draft Std. X.mp2p-mssr, 2016.
- [9] *Managed P2P Communications: Multimedia streaming signaling protocol*, ITU-T Draft Std. X.mp2p-mssp, 2016.
- [10] W. Hyun, SH.Kim, CK.Lee, SG.Kang, "Implementation of PREP (P2P-based streaming Protocol) Client" in Proc. ICACT 2012, pp 628-631, 2012.
- [11] W.Hyun et al, "Study on Architectural Model and Design of P2P-based Streaming Client", ICHIT 2011, CCIS 206, pp.1-7, 2001.
- [12] *Vagrant*, <https://www.vagrantup.com/>
- [13] *CoreOS*, <https://coreos.com/>
- [14] *Docker*, <https://www.docker.com/>
- [15] *Express*, Node.js web application framework, <http://expressjs.com/>
- [16] A simple AHK implementation of Winsock (TCP/IP), <https://autohotkey.com/board/topic/53827-ahksock-a-simple-ahk-implementation-of-winsock-tcpip/>
- [17] <https://github.com/ETRI-PEC/MP2P-UMS.git>



Wook Hyun is a research staff member with ETRI (Electronics and Telecommunications Research Institutes) since 2000. He has received M.S. degree in Information Communication Engineering from Chungnam National University, Korea in 2000. His research interests include VoIP, SIP, NGN, P2P, overlay networking and digital signage.



Ju Young Park is working for ETRI from when he has received his PH.D degree in 2001 from Chungnam National University. Thereafter, he took project editorships both in ITU-T and ISO/IEC/JTC1, and he also has developed three International Standards (IS). His major research areas are smart work, Multicast, QoS protocol and architecture. He also has great concerns on mobile communication and IOT.