# SOFTWARE STREAMING TECHNOLOGY FOR TELEMETICS APPLICATIONS

Jungsook Kim, Jihoon Choi, and Jungdan Choi

Telematics Research Division, ETRI.
Eoeun-dong 52, Yuseong-gu, Daejeon city, 305-806, KOREA(South)
{jungsook96, cjh, jdchoi}@etri.re.kr, Tel: +82-42-860-1096, Fax: +82-42-860-4844

**ABSTRACT:**

The software streaming technology enables telematics software to be automatically updated through a wireless network. When the software starts running, software streaming system inspects its version and then, automatically download latest one. The software streaming system breaks the software into several pieces that are streamed to the user as needed. In this way, software streaming system can improves the telematics application load time while updating the software through wireless network. In our experiments, the application load time was reduced about 7 times compared to downloading whole software at best case.

**KEY WORDS:** Software streaming, stream enabled software, virtual run time environments

## 1. INTRODUCTION

Extending the telematics services, telematics terminals typically support various applications with different characteristics. With limited storage resources, it may not be possible to keep all features of the applications loaded on telematics terminals. In fact, some software components may never be needed. Thus, the terminals are week in utilizing their storage resources effectively. In addition to that, telematics applications are usually installed and upgraded into telematics terminals with active synch. However, active synch is unfeasible for managing applications on the terminal built in a car. It's not possible to connect the terminal mounted on a vehicle to PC. Furthermore, applications and their data will likely to change over time, and perhaps remarkable in the case of the map data of navigation applications. To make matters worse, the vehicle mounted terminal and its applications must work until scraping the car, namely, at least 10 years.

In order to effectively repair and maintain telematics software, it should be updated automatically through a wireless network. Software streaming enables the automatic update. When software starts to run, its version is inspected. If it is stale, we download the latest software as soon as eliminate the stale one. The software transmitted to the client is a piece of the whole but not entire. In other words, the software has been broken into several pieces and only needed parts are transmitted on demand. Immediately on downloading the first piece of the software, it is executable. The software streaming enables software to be updated through slow wireless network while minimizing user perceived application load time. While the application is running, additional parts of the software can be downloaded in the background. Moreover, software streaming effectively utilizes the client storage resource by downloading only needed parts.

In this paper, we have introduced a new automatic software update method using software streaming technology. It improves the telematics application load time while updating the software through wireless network. We implements the software streaming technology on windows CE 4.1 and 4.2.

The rest of this paper is organized as follows: In section 2, we explain out stream software system composed of stream software generator, streaming client, streaming server. In section 3, we provide performance analysis of the streaming environments. Then, we describes related work in the area of the software streaming in the section 4. Finally, in section 5, we conclude and discuss the future work.

## 2. DESIGN AND IMPLEMENTATION

We will introduce our telematics software streaming system composed of stream software generator, streaming server, and streaming client. We design our system fully supports legacy applications. The stream software generator converts the legacy software into stream enabled software without any modification. The streaming client is designed and implemented stream enabled software to execute while legacy software is running. Furthermore, any stream enabled software is concurrently executable with any other stream enabled software. We try to minimize network overhead in our system. The

local cache in the streaming client allows software blocks to be reused. The streaming server is designed to be scalable. We provide simple load balancer and several container servers transmitting streaming software to the streaming client. The container server is easily joined to the streaming server system in real time.

Figure 1 describes service flow of our telematics software streaming system. In our approach, stream enabled software must be created before it can be streamed to the client. As shown figure 1, the stream software generator creates the stream enabled software with binary image of legacy application complied from application source code by a standard compiler. Once the stream enabled software is generated, it is loaded to the streaming server which effectively transmits a requested software block as maintaining block index. The streaming client easily starts running stream enabled software through a streaming server's Web interface. When the software starts to run, its version is inspected. The streaming client requests an initial piece of the latest software as soon as eliminate the stale one if it is stale. Additional software blocks are requests as the software keep running. In this section, we first explain the say of generating stream enabled software. We then describe our streaming client and streaming server in detail.

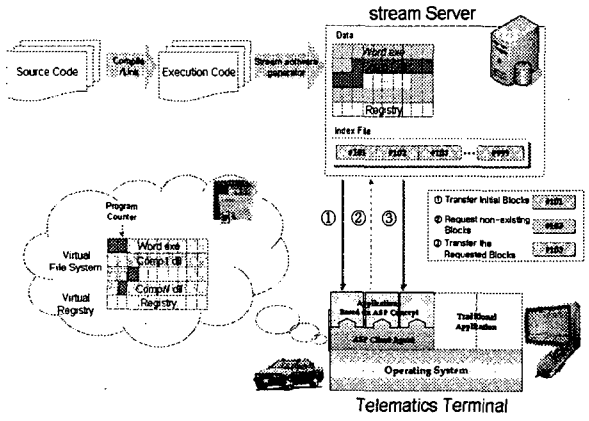


Figure 1. Service flow of software streaming system

### 2.1 Stream software generator

The stream software generator is a tool that packages and virtualizes legacy applications as network application for real-time delivery. This tool makes the conversion without modifying the application source code. The software generator monitors and records all changes during application installation. It analyzes which registry setting, files, and environmental variables are modified or added by the application. The stream software generator uses this information to create a stream enabled virtual application package.

In order to extract the information, we propose a simple method that compares system image differences during installation. Figure 2 shows the system state. State A and B illustrate the system status before and after install, respectively, and state C describes the application itself.

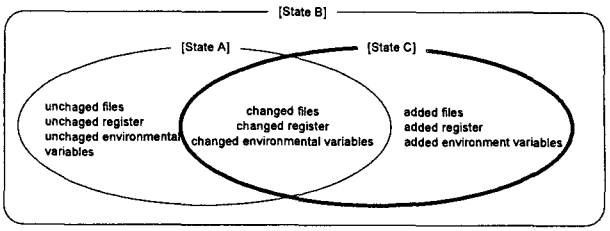


Figure 2. System state diagram

In our approach, each state is expressed with the combination of a file tree and a register tree shown in figure 3. Non-leaf nodes are the path to reaching files or registers. Leaf node contains detail information about files or register. For example, leaf node of the file tree contains file creation time, modified time, and size. The register tree's leaf node maintains register type, key, and value. We recognize the node modification with the change of the detail information.

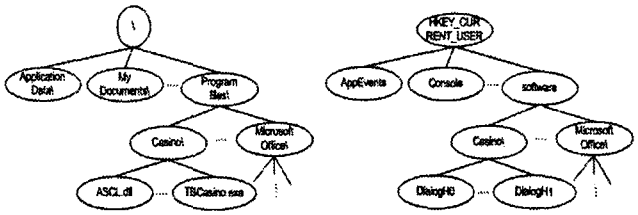


Figure 3. Tree structure representing each state

The trees of state C can be created from state A and B's trees as described in figure 4. Installing software makes the tree of state B added new nodes, i.e., H, I, and J, and changed node E to E`. Using depth first algorithm, we generate the tree of state C. With the extracted information about state C, we makes an information file used to build application specific configurations and virtual run time environment later.

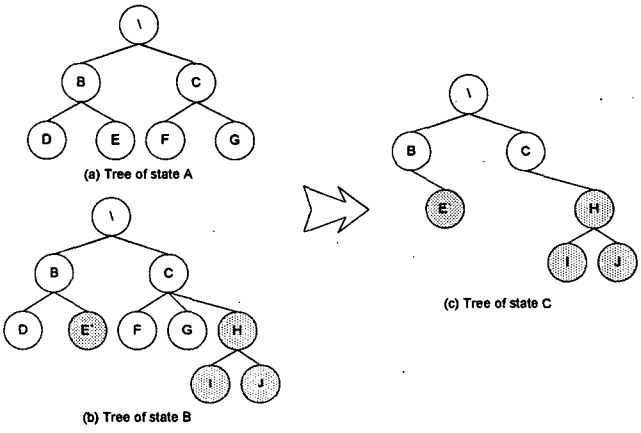


Figure 4. The virtual application extraction mechanism during installation

Finally, our stream software generator consecutively breaks the aggregated files from state C down software blocks sized of 4KB, same with page size of Windows CE.Net. Then, it outputs a stream software pack file containing the compressed data files of the sequenced application and the information file for the application.

### 2.2 Streaming Client

Our stream enabled software is runnable within a streaming client's virtual run-time environments. In the virtual run-time environment, the stream enabled software executes in a same way that it was already installed. Our streaming client allows any stream enabled software to run side-by-side with any other one. That is, the streaming client creates virtual run-time environments for each software. The virtual run-time environment created by one software cannot be seen by other softwares and thus, several softwares are concurrently executable on the same computer without any conflict. The streaming client adopts a local cache which improves the reusability of downloaded blocks. The local cache minimizes network overhead, which is a key issue for stream enabled software through wireless network.

Our streaming clients must be preinstalled at telematics terminals before requesting stream enabled software to the streaming server. Figure 5 shows the architecture of the streaming client, which is composed of terminal agents(TA), IO interceptor(IOT), virtual registry(VR), virtual file system(VFS), cache manager(CM), and network manger(NM). The followings are brief descriptions on components.

- *terminal agent.* NM initializes virtual run-time environment.
- *Virtual registry.* VR maintains software specific registry information and handle it. We implement the virtual registry using overlay method rather than copying the entire registry. The method efficiently handles the registry information. Items in the real registry may be read by the application as long as a virtual copy of that item is not available. All application writes to the registry are contained within its virtual registry and not shared with other.
- *IO interceptor.* IOT intercepts and handles all file IO requests between stream enabled software and file system. IOT checks whether the IO requests is for the file in the virtual file system or windows native file system. According to the results, IOT redirect the requests to the virtual file system or pass it to the windows file system.
- *Virtual file system.* Virtual file system creates same directory structures and empty files with legacy application in a specific directory under C. The created files are used for the local cache. The IO interceptor handles all IO requests made by software to files.
- *Cache manager.* CM manages local cache for improving software block reusability. CM cache reduces search time with index table.
- *Network manager.* NM takes care of all network connection with the server.

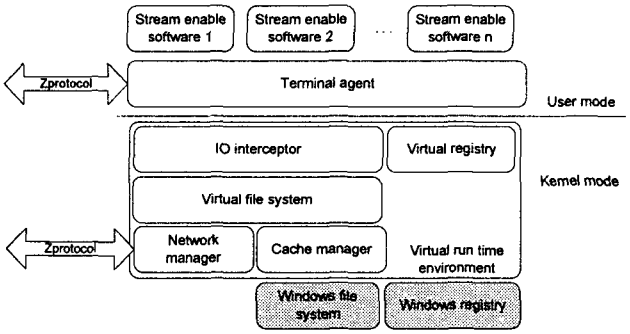


Figure 5. Architecture of the streaming client

### 2.3 Streaming Server

The streaming server is designed and implemented to be scalable. The streaming server transmits a software block as requested, which induces main overhead, i.e. network overhead, for the server. The server system capacity is dependent to capacity of the both. In order to support clients exceeding the capacity of the network card and backbone, the server should be regionally distributed. We design the server using distributed method as shown in figure 6. Our streaming server is composed of several container servers, a control server, and a database server. The container servers may be deployed at several areas. The load balancing between container servers are processed at the control server. The server load is distributed based on connection only one time at initially connecting at a container server. A client keep connecting with the container server until finishing a stream enabled software. Thus, our streaming server system is feasible to flexibly extend or reduce its system capacity.

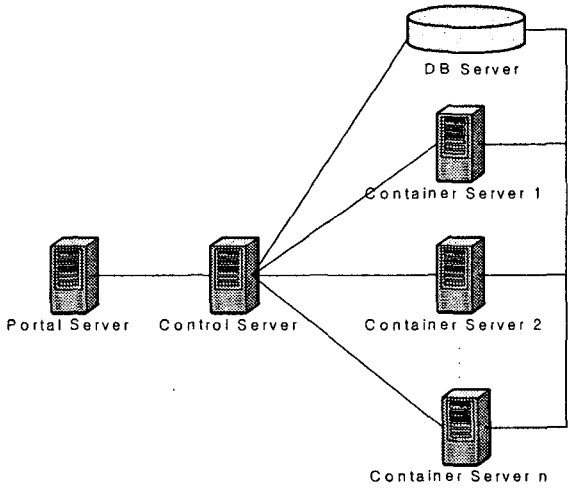


Figure 6. Architecture of the streaming server

## 3. EXPERIMENTS

We evaluated the performance of the software streaming system. The container server, portal server, and the database server are running on a 2.4GHz CPU with 2GB memory and 20GB HDD. The container servers are running on 1GHz Intel Pentium CPU with 2GB memory and 19GB HDD. We deployed apache version 1.33 at the portal server and control server, and MySql version 4.0.18 at the database server. We implemented our streaming client at Samsung NEXIO-XP30 with PXA255 400MHz CPU, 64MB ROM , 128MB SDRAM, and 800*400 resolution. The NEXIO-XP30 worked on windows CE.Net 4.1 and supported just Wireless lan(802.11b).

Table 1 shows the experimental results of our system deployed at NEXIO-XP30 terminal. We can start to run the ZIOGolf2 with only 15% of the whole application. The portion of requested application for starting it is different from each other. The portion is depending on the way of implanting the program.

Table 1. The application load time on NEXIO-XP30

| Application | Total size of application (KB) | Size of the application to start it at our software streaming system (KB) | Time to run the application after downloading whole software (sec) | Time to start the application at our software streaming system (sec) |
|---|---|---|---|---|
| ZIOGolf2 | 4,598 | 720 | 215 | 30 |
| WordWiser | 2,427 | 327 | 98 | 15 |
| Casino | 1,990 | 576 | 82 | 22 |
| Gangi | 1,942 | 1,236 | 93 | 60 |
| Hunmin | 1,901 | 996 | 62 | 32 |

As shown the table1, we can dramatically reduces the application load time. Compared to downloading the entire software, the software streaming method can load the software about 7 times rapidly at best case.

## 4. RELATED WORKS

In Java applet implementation, the typical process of downloading the entire program is eliminated. A Java applet can be run without obtaining all of the classes used by the applet. Java class files can be downloaded ondemand from the server. If a Java class is not available to the Java Virtual Machine (JVM) when an executing applet attempts to invoke the class functionality, the JVM may dynamically retrieve the class file from the server [T1], [T2]. In theory, this method may work well for small classes. The application load time should be reduced, and the user should be able to interact with the application rather quickly. In practice, however, Web browsers make quite a few connections to retrieve class files. HTTP/1.0, which is used by most Web servers [J1], allows one request (e.g., for a class file) per connection. Therefore, if many class files are needed, many requests must be made, resulting in large communication overhead. The number of requests (thus, connections) made can be reduced by bundling and compressing class files into one file [E1], which in turn unfortunately can increase the application load time. While the transition to persistent connections in HTTP/1.1 may improve the performance for the applet having many class files by allowing requests and responses to be pipelined [E2], the server does not send the subsequent java class files without a request from the client. The JVM does not request class files not yet referenced by a class. Therefore, when a class is missing, the Java applet must be suspended. For a complex application, the class size may be large, which requires a long download time. As a result, the application load time is also long, a problem avoided by the block streaming method which we will describe in the next section.

## 5. CONCLUSION

Software streaming technology allows telematics terminals to start executing an application while the application is being transmitted. We presented a method for transmitting the stream enabled software from the server to be executed on the streaming client. Our streaming method can lower load-time delay, bandwidth utilization and memory usages. We verified our streaming method using Samsung NEXIO terminal

## 6. REFERENCES

[J1] E. Nahum, T. Barzilai and D. D. Kandlur, 1999, *Performance Issues in WWW Servers*, IEEE/ACM Transactions on Networking, vol. 10, no. 1, pp. 2-11.

[T2] J. Meyer and T. Downing, 1997, *Java Virtual Machine*, California: O'Reilly & Associates, Inc., pp. 44-45.

[T1] T. Lindholm and F. Yellin, 1999, *The Java Virtual Machine Specification, 2nd ed.*, Massachusetts: Addison-Wesley Publishing Company, pp. 158-161.

[E1] P. S. Wang, 1999, *Java with Object-Oriented Programming and World Wide Web Applications*, California: PWS Publishing, pp.193-194.

[E2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, June 1999, *Hypertext Transport Protocol — HTTP/1.1*, RFC 2616, The Internet Engineering Task Force.