DESIGN AND IMPLEMENTATION OF AN ELECTRONIC SERVICE GUIDE FOR MOBILE VIDEO SYSTEMS

by

Kaushik Choudhary

B.Tech., West Bengal University of Technology (India), 2006

A Project submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2012

APPROVAL

Name: Kaushik Choudhary

Degree: Master of Science

Title of Project: Design and Implementation of an Electronic Service Guide for Mobile Video Systems

Examining Committee: Dr. Arrvindh Shriraman,
Assistant Professor, Computing Science
Chair

Dr. Mohamed Hefeeda,
Associate Professor, Computing Science
Senior Supervisor

Dr. Jiangchuan Liu,
Associate Professor, Computing Science
Supervisor

Dr. Joseph Peters,
Professor, Computing Science
SFU Examiner

Date of Defence: 12 July 2012

Date Approved: _____

Partial Copyright Licence

Abstract

We study the problem of designing a configurable testbed for mobile TV research. Using simulations to conduct experiments on mobile TV may not reproduce real life field scenarios and challenges. To address this problem, we design and implement a configurable Electronic Service Guide (ESG) server capable of delivering TV programming information to mobile TV subscribers and communicating tuning and multimedia encoding information to mobile devices. We also implement link layer signaling mechanism for the testbed and integrate all components for an end-to-end mobile TV system for research. We validate the implementation with a variety of tools such as offline and USB based online transport stream analyzers capable of capturing and analyzing signals in real time. Furthermore, we complement our stream analysis with a live demonstration of the ESG server.
Keywords: Mobile TV, ESG server, link layer signaling, DVB-H

Acknowledgments

I am deeply indebted to my senior supervisor Dr. Mohamed Hefeeda, for his continuous support, guidance and encouragement. He patiently answered my questions, providing valuable insights and motivated me to work hard. I highly appreciated that Dr. Hefeeda also listened empathetically and encouraged me to pursue my graduate career with vigour. This project would not have been possible without his guidance.

I would like to thank my supervisor Dr. Jiangchuan Liu and my thesis examiner Dr. Joseph Peters, for being on my committee and reviewing this report. I would like to thank Dr. Arrvindh Shriraman, for taking the time to chair my defense. I would also like to extend my gratitude to the faculty and staff in the school of computing science at SFU, particularly Dr. Joseph Peters for being very kind and encouraging both during my course with him as well as when I worked as a TA for him, Dr. Bob Headley for answering my numerous questions in his course, and our IT administrator Jason Ashby for providing me support with the work stations at my desk.

I would like to thank my colleagues at the Network Systems Lab, specially Ahmed Hamza and Somsubhra Sharangi for all their help and encouragement. I would also like to thank my friends Kenneth Wong and Vivek Anand for encouraging me to believe in my capabilities.

Most significantly, I cannot express enough gratitude to my girlfriend and now fiancée, Mimi He, for being there and providing encouragement, support, love and care when I needed it the most and for being there in my life.

Last but not least, I would like to thank my parents for all the sacrifices they made for me and for giving me their unconditional love and support. I would also like to thank my brother for his unrelenting and unspoken expectations which provided me with the drive.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In this chapter we introduce features of mobile video broadcast. We describe the challenges in presenting Electronic Service Guide (ESG) to mobile users under practical constraints. We introduce the design proposed in this project and summarize our contributions. We then present the organization of the rest of this report.

1.1 Introduction

In recent years, streaming technology has become an increasingly popular mechanism to deliver multimedia content to end-users. Recent studies [7] have shown that there exists a large diversity in end-user devices. Among the multitude of devices, mobile phones have become immensely popular in recent years since they have been significantly enhanced with sophisticated operating systems and powerful hardware. Smartphones offer capabilities similar to that of a modern computer and are capable of installing third-party applications. Since the introduction of Apple's iPhone and smartphones based on Google's Android operating system, the usage and demand of smartphones have increased significantly. Figure 1.1 illustrates predictions for mobile data traffic for the next several years, suggesting that there will be a considerable increase in mobile data traffic (with smartphones contributing to as much as 48% of all data traffic by 2016).

Furthermore, video data has unequivocally taken over as the predominant type of data consumed by mobile devices. Figure 1.2 shows that mobile video traffic is expected to grow exponentially in the next several years dominating web data traffic and file sharing services like peer-to-peer sharing. To meet this growing demand, mobile service providers need

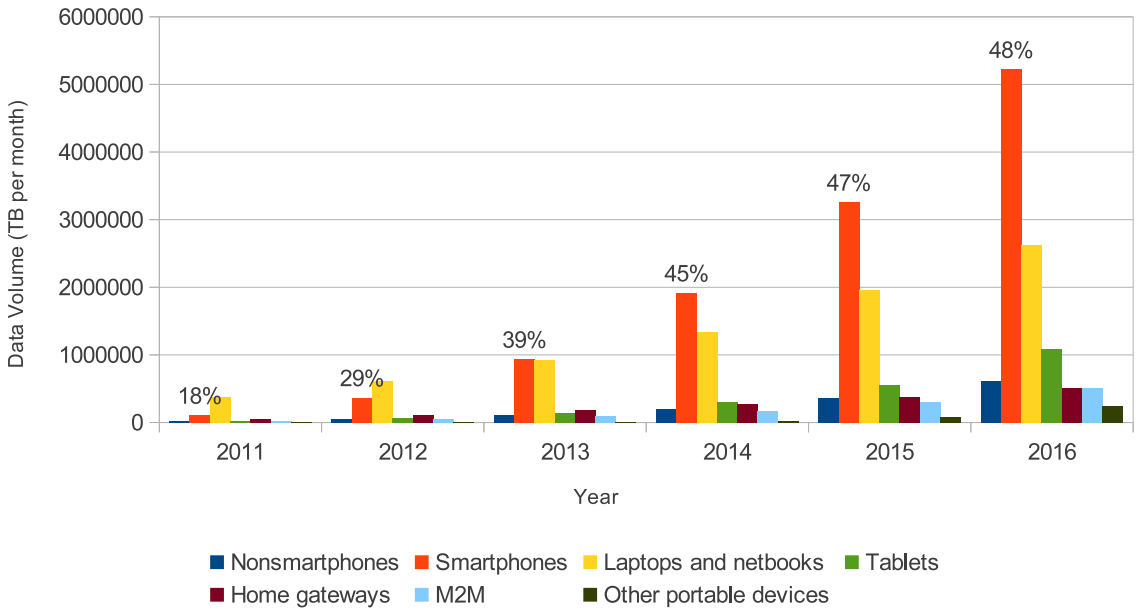


Figure 1.1: Global Mobile Data Traffic. Extracted from [7].

massive investments to increase mobile network capacity and video traffic will significantly consume that increased capacity during peak hours of video consumption.

Mobile broadcast networks defined by standards such as DMB [6], DVB-H [14, 22], CMMB [1], ISDB-T [3] and ATSC-M/H [4] could support unlimited numbers of users in the coverage area without risking network saturation. Thus they are useful for delivering high-demand video programs (relatively large number of viewers) and protecting the scarce unicast spectrum of mobile networks. A Broadcast Mobile Convergence (BMCO) forum report [46], shows that using broadcast networks reduces video traffic load on the network by 60% in a given area during high-demand video programs like sporting events. Hence, broadcast technologies allow efficient usage of delivery networks both in terms of delivery cost and service provisioning.

ESG acts as a service discovery tool for both mobile TV users as well as mobile terminals. ESG provides users of mobile TV continuously updated information about broadcast programming and scheduling information for current and upcoming programs. Further, ESG provides the mobile terminal middleware with signaling data for service lookup and playback capability negotiation to decide connection and video decoding parameters. There

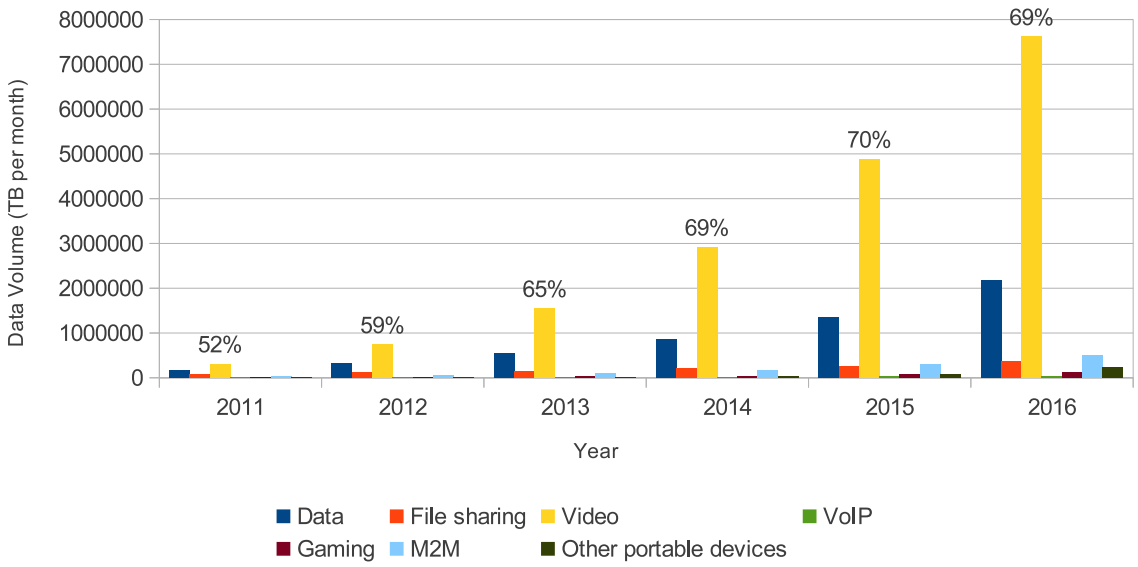


Figure 1.2: Global Mobile Data Traffic categorized by type of traffic. Extracted from [7]

are two standards for ESG - one defined by the European Telecommunications Standards Institute (ETSI) [18, 20] and another defined by the Open Mobile Alliance [2]. ETSI refers to ESG as the Electronic Service Guide while OMA refers to it simply as *Service Guide*. In this project we design and implement the service layer ESG server as defined by the ETSI standard TS 102 471 [18, 24].

1.2 Problem Statement and Project Contributions

Our goal is to implement an open source ESG server that can be used in various broadcast networks. This report describes the design, implmentation and validation of our ESG server.

1.2.1 Problem Statement

Mobile TV technologies are an important resource for reducing stress on unicast networks delivering video content to subscribers. Mobile TV also brings traditional television programs to mobile devices where users can view programs on the go. Hefeeda and Hsu [30] developed an open-source mobile TV testbed for use in mobile TV research. The DVB-H

standard, only specifies the transmission behavior below the IP layer. Other IP-based service layer standards over the IP layer are required to implement an end-to-end DVB-H based mobile TV system. In the aforementioned testbed, a basic service layer was implemented without any configuration capabilities. Since the service layer is above the IP layer, applications developed at the service layer may use other IP-based mobile TV systems. Thus the problem addressed in this project can be stated as follows:

Problem 1:*Design and implement a service layer, configurable, electronic service guide server. Validate the server in a real mobile TV testbed.*

1.2.2 Project Contributions

We implement an ESG server in a real mobile TV testbed developed at the Network Systems Lab at Simon Fraser University [30]. Our contributions are:

- We design and implement a modular, configurable ESG server based on the guidelines specified in the DVB-IPDC standards [18, 24, 20]. We integrate the ESG server to the mobile TV testbed mentioned earlier. The ESG server allows configuration of channels and programming to be delivered to end users through XML configuration files.
- We design and implement the link layer signaling mechanism for DVB-H to integrate the ESG server with the mobile TV testbed. The signaling is realized through Program Specific Information/System Information (PSI/SI) tables.
- By integrating the ESG server with the mobile TV testbed, we develop an end-to-end open-source mobile TV testbed system useful for mobile TV research. The ESG server is implemented at the application layer. It does not handle mobility which is a concept handled in other layers of our mobile TV testbed.

1.3 Report Organization

The rest of this report is organized as follows. In Chapter 2, we briefly provide a background of ESG and summarize related works in the literature. In Chapter 3, we describe the design, implementation and validation of the ESG server and PSI/SI tables. And finally in Chapter 4 we conclude this report and outline potential extensions of this work.

Chapter 2

Background and Related Work

In this chapter we briefly review mobile TV technologies, describe ESG and summarize related previous works.

2.1 Background

2.1.1 Mobile TV

In this section we present an overview of different mobile TV architectures. There are several mobile TV standards that have been defined such as DMB [6], CMMB [1], ISDB-T [3], ATSC [4], and DVB-H [14, 22]. Digital Multimedia Broadcasting (DMB) is the world's first official mobile TV service started in South Korea in 2005. The DAB [15] standard was extended to develop DMB as part of a Korean national project. It can operate via satellite (S-DMB) or terrestrial (T-DMB) transmission. China Mobile Multimedia Broadcasting (CMMB) [1] is a mobile TV standard developed in China by State Administration of Radio, Film and Television (SARFT). CMMB provides up to 25 video channels and up to 30 radio channels using both satellite and terrestrial transmission. Specified by Association of Radio Industries and Businesses (ARIB) in Japan, Integrated Services Digital Broadcasting (ISDB) [3] is the mobile TV standard in Japan. The ISDB-T design divides channels into 13 segments. 12 of these segments are used by an HDTV broadcast signal while the remaining one channel is used for mobile TV service.

In 2009, the Advanced Television Systems Committee ratified the ATSC-M/H mobile TV standard for mobile devices in North America. Substantially different from the terrestrial

ATSC standard due to limitations of mobility, the ATSC-M/H standard defines a fixed transport stream (TS) structure for easier processing by mobile receivers. ATSC 2.0, a yet to be ratified standard, allows for more advanced features like video on demand services, targeted advertising and MPEG-4 compression.

We implement our ESG server in a mobile TV testbed based on DVB-H. DVB-H [14, 22] is an extension of the DVB-T [13] standard for terrestrial broadcast. Tailor made for mobile devices DVB-H is an open international standard and one of the most widely used mobile TV standards. An important feature of the DVB-H standard is that it provisions for reducing energy consumption on mobile devices by broadcasting multimedia data in bursts allowing mobile terminals to turn off their RF circuits periodically. Aside from this, it is also possible to deliver TV programs to mobile devices over wireless cellular networks. The 3G partnership project defined the Multimedia Broadcast and Multicast Service for Universal Mobile Telecommunications Systems (UMTS) to support broadcast and multicast multimedia delivery models [29, 37]. In this report however, we focus on dedicated broadcast networks.

2.1.2 Overview of DVB-H Standard

The DVB-H mobile TV standard defines Physical and Link Layer protocols and uses IP to interface with higher layer protocols such as Real-Time Transport Protocol (RTP) [48] and UDP. In order to implement an end-to-end mobile TV system howDever, another set of standards for service layer applications has been defined. The DVB-IPDC [16] standard not only defines higher layer protocols but also enables cooperation with cellular networks such as UMTS. It also enables bi-directional communication and thus supports interactive services like ESG. The ESG standards [18, 24] are defined under the DVB-IPDC standard. Further details about ESG are described in subsection 2.1.3. DVB-H uses MPEG-2[35] transport streams for coding and transferring data. IP packets are encapsulated in the transport streams.

IP Datacast (IPDC) uses UDP protocol at the transport layer. However, since it is defined above IP, any other IP capable system may use IPDC at the higher layer. On top of the UDP protocol IPDC uses RTP and File Delivery Over Unidirectional Transport (FLUTE) [44] that was built over the definition of Asynchronous Layered Coding(ALC) protocol [44] for delivering multimedia data and metadata. IP Datacast also defines XML-based ESG that transmits channel and programming information to subscribers. ESG also

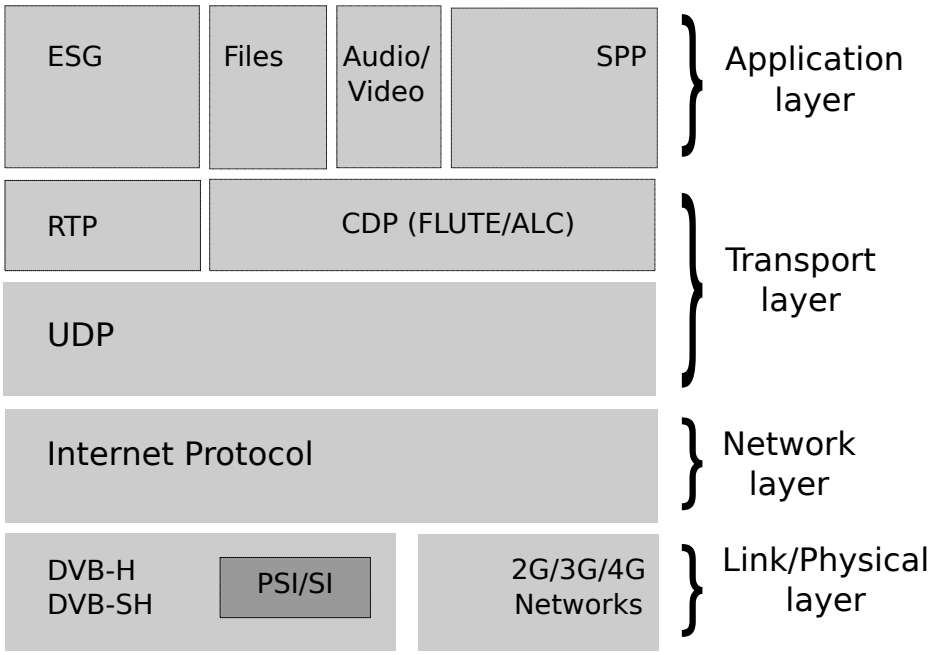


Figure 2.1: Simplified DVB-IPDC Protocol Stack

delivers initialization parameters to the mobile terminal. Figure 2.1 shows the complete protocol stack for video broadcast over DVB-H networks. In the figure, SPP stands for Service Purchase and Protection which is a service that allows the service provider to protect content and enable a range of different subscription models. The DVB-SH(Digital Video Broadcast - Satellite services to Handhelds) [12] standard defines delivery of IP based multimedia content and data to handheld terminals such as mobile phones, based on a hybrid satellite/terrestrial downlink.

DVB-H employs Orthogonal Frequency Division Multiplexing(OFDM) at the physical layer. IP packets are encapsulated using Multi-Protocol Encapsulation (MPE) sections to form MPEG-2 transport streams. Two of the most important features of DVB-H are time slicing and error correction. We describe each of them below:

Forward Error Correction: Broadcast videos suffer from high error rates due to factors like fading, shadowing and interference. DVB-H employs a Forward Error Correction (FEC) mechanism to minimize the aforementioned error rates. FEC also corrects errors whenever possible. Therefore, the sequence of MPEs carrying data from a specific channel are FEC-protected before broadcast. DVB-H uses Reed-Solomon coding and time interleaving in

the link layer to protect IP packets transmitted. This mechanism allows us to improve the signal-to-noise ratio for the transmitted data.

Time Slicing: To reduce energy consumption in mobile devices from video playback, DVB-H channels are transmitted in bursts at a bit rate much higher than the encoding rate of the channels being transmitted. This allows mobile devices receiving the bursts to turn off their RF circuits until the next burst arrives. During this period when the radio circuit is turned off, the mobile device can continue playing the video data received. The time slicing mechanism also allows for seamless handovers - important for mobile devices.

2.1.3 ESG Overview

An ESG contains information about services offered including information such as titles and genres of the different content items. Through the information in the ESG, the user can select the services and items he/she is interested in. Aside from this, ESG also contains information required by the terminal on how to access these services. ESG also provides mechanisms for managing subsystems like billing and accounting. ESG operations take place after the DVB-H receiver on the terminal has been started and has tuned to a particular transport stream carrying IPDC services. On the server end, these operations translate to encoding of XML files containing content and tuning information for transmission to the terminal.

ESG contains information about available multimedia services on offer from the service provider. ESG contains both human readable information visually presented to the user in the form of program listings, descriptions and schedules as well as media initialization information used by the terminal to tune into a service selected by the user. ESG data is transmitted in the form of XML files transported over the FLUTE/IP protocol [17] .

Electronic Service Guide [18] (superseded by [24]) was described as part of the DVB IPDC specification [16]. The DVB IPDC standard was created with the objective of providing the higher layer protocols for both DVB-H based on IP (or any other IP capable system) as well as an optional access to a cellular communication system such as UMTS [38]. We implemented ESG server on top of the open source DVB-H mobile TV testbed developed in the Network Systems Lab [30]. Figure 2.1 shows the DVB-IPDC protocol stack.

ESG operations begin after the terminal is synchronized to a particular transport stream. The ESG process flow consists of three operations - ESG bootstrap, ESG acquisition and ESG update. These operations are discussed in more detail in Chapter 3. Once the terminal

has tuned into a transport stream, it receives the ESG bootstrap information from a well known IP address advertised through PSI/SI tables encoded in the stream. With this information the terminal can locate the IP stream carrying ESG information and start receiving the information.

The ESG specification covers the ESG data model, ESG representation, ESG encapsulation and ESG transport. We briefly describe the relevant sections of the ESG specification below. The detailed description of the operations is available in the ESG standards [18, 24]. The ESG data model specifies a set of data structures that are instantiated to describe available service. It is specified based on the XML schema[49]. The XML schema specification allows for consistency of the ESG data. ESG representation provisions the fragmentation of ESG XML data into XML fragments which enables efficient representation and transport of data, minimizing size of the metadata delivered to subscribers. The ESG encapsulation process packs ESG XML fragments into containers of considerable size. The containers also carry management information for processing the fragments.

The ESG data model provisions authentication mechanisms for levying service charges to subscribers through the Purchase and the Purchase Channel Fragments. We note that in our ESG server, we implement only four necessary types of ESG fragments - Service Fragment, Content Fragment, Schedule Event Fragment and Acquisition Fragment. We describe each of these fragments as follows:

- Service Fragment - The service fragment describes an IPDC service such as a traditional TV channel.
- Content Fragment - The content fragment contains metadata that describes the content independent of an instantiation of that content.
- Schedule Event Fragment - The schedule event fragment contains the broadcast time of a scheduled item which is a content item of a service.
- Acquisition Fragment - The acquisition fragment contains information to access a service. It specifies among others information displayed to the user such as component characteristic, information relevant to the ESG application such as contentType and Session Description used by the media player for initialization.

The DVB IPDC standard is undergoing revisions to include new features like on-demand ESG through 3G networks (unicast as an option), a notification feature to deliver notification

Table 2.1: List of implemented PSI/SI tables and their corresponding PIDs

Table	PID
PAT (`PatTable` class)	0x0000
NIT (`NitTable` class)	0x0010
SDT (`SdtTable` class)	0x0011
PMT (`PmtTable` class)	User Defined
INT (`IntTable` class)	0x0016
TDT (`TdtTable` class)	0x0014

information to terminals, and more advance interactivity features. The standard will be ratified as DVB IPDC 2.0.

2.1.4 PSI/SI Tables overview

To support signaling of ESG information, we also developed the link layer signaling mechanism of DVB-H as part of this project. The PSI tables enable a mobile device to demultiplex the services available on a transport stream for viewing. The tables are structures segmented into sections and inserted into TS packets, some with preassigned packet IDs (PID) and some with user selectable PIDs. Of the tables implemented in our base station the values of Pids are given in table 2.1. We implement two types of PSI tables mentioned below:

Program Association Table (PAT): PAT defines the correlation between the program number and the PID of the TS packets carrying DVB service definitions. The program number is a numeric label assigned to a DVB service.
Program Map Table (PMT): PMT associates program numbers and program elements that comprise them.
Aside from this, we also implement four SI tables that we define below:
Network Information Table (NIT): NIT conveys information relating the physical organization of the transport stream and the DVB network itself.
IP/MAC Notification Table (INT): INT defines the location of IP streams on a given network.
Service Description Table (SDT): SDT contains information that describes the services available on the DVB network such as DVB service name, service provider information etc.
Time and Date Table (TDT): TDT transmitted in a single section informs the terminal

of the UTC-time and date at the server. The time is coded in the Modified Julian Date format.

2.2 Related Work

This project is an extension to the mobile TV testbed developed by Hefeeda and Hsu [30]. The testbed is used for mobile TV research and implementing the ESG server and PSI/SI link layer signaling allows for exploring further research problems. Some previous works implement models or simulations of PSI/SI tables and ESG servers. Jokela et al [36] measure metrics for efficient transmission of PSI/SI tables over the DVB-H network. The variations in the PSI/SI tables including variations in network parameters and table section sizes are achieved through simulations. They also present field measurements taken over a real DVB-H network in the city of Turku in Finland. However, in this case, the PSI/SI tables were not customized. Combining the observations from simulations and field measurements, the authors conclude that when PSI/SI table transmission is optimized with respect to the used network capacity and the impact of section sizes is considered, then using the smallest section is not necessarily the best option. An optimal section size for efficient transmission can be found.

Earlier, Oksanen et al [43] developed a mobile TV testbed in Tampere university in Finland. This testbed was developed to establish the performance of DVB-H networks. But, since it is not open source, the system does not provide avenues for customization at the specification level for research work. Hsieh et al [32] developed a DVB-H middleware which parses TS packets according to the DVB-IPDC specification to achieve interactive multimedia services. The middleware, based on Java, interacts with the mobile terminal and enables a voting server and interactive video-on-demand service. On the server end, the middleware, parses the transport stream using information available in the PSI/SI packets and also parses the ESG XMl data. Additionally, the middleware is able to detect the type of data from the TS packets and calls corresponding modules to process video data or ESG metadata.

Hammershøj et al. [27] developed components of a larger DVB-H mobile TV system. The project is a collaborative work between Aalborg University and the Center for Communication, Media and Information Technologies. As part of this project, the authors developed

an IP encapsulator, encoder and an ESG server. The ESG server is based on the OMA Broadcast standard [2] while the IP encapsulator is based on the Python-based open source FATCAPS encapsulator. Our testbed replaces FATCAPS with a C++ based encapsulator along with PSI/SI signaling developed in this project. This allows for configuration consistencies while encoding transport streams.

Schatz et al [47] developed a hybrid mobile TV testbed for deploying a DVB-H based mobile TV service with the backup of using 3G/UMTS cellular technologies. The authors built an all-IP intermediary layer for bearer-agnostic multimedia transmission. This project is validated on a WLAN network which may not create real life field scenarios for mobile TV services.

Chapter 3

Design and Implementation of the Proposed ESG Server

This chapter explains each of the elements of the overall system (depicted in figure 3.1). The most important entities under the scope of this report are PSI/SI and ESG. We discuss the details of the design of this system in Section 3.1 including the design for PSI/SI link layer signaling and the ESG server design. In Section 3.2 we present our implementation of the system based on the recommendations in [20]. Finally, in Section 3.3 we present the validation of our system.

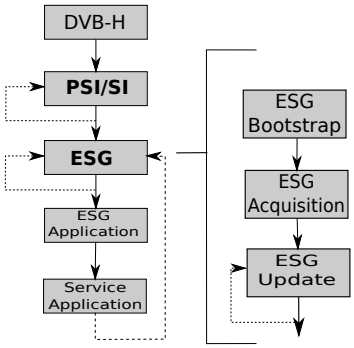


Figure 3.1: ESG Operations on a DVB-H based implementation [18]

DVB-IPDC does not specify the physical media since it was designed to be system independent. In order to implement ESG on top of the DVB-H [14, 22] system, we had to implement the PSI/SI tables [26, 25] in the DVB-H system for low level access to the IP

streams.

3.1 ESG Server Design

In the following subsections we describe the design of the PSI/SI tables in subsection 3.1.1 and the ESG server in subsection 3.1.2.

3.1.1 PSI/SI Tables

The PSI/SI tables were designed for the mobile TV testbed [30] project developed in the Network Systems Lab in SFU. The classes for the PSI/SI tables were implemented as part of the `Transmitter` module of the testbed. The `Transmitter` module in the testbed periodically broadcasts the PSI/SI tables. The PSI/SI tables are managed by `Psisi` class as a priority queue on next broadcast time, and `Transmitter` pops the next PSI/SI `Table` whenever the air medium is idling, and that table is inserted back to the priority queue with a new broadcast time after being broadcast. The class diagram for the `Transmitter` module and the `PSI/SI` module designed as part of this project is shown in figure 3.2. The broadcast times are obtained by the transmitter using the `getNextPsisiTime` method. `Psisi` implements an `initTables` and an `uninitTables` methods to initialize and free the priority queue `tables`.

To initialize the tables, `Psisi` reads a configuration file containing a mapping of the table name with the corresponding transmission frequency and creates the table for mapping entry. Once created these tables are then put into the priority queue `tables` in the order of next nearest transmission time. Whenever `Transmitter` has no data to send, it calls `getPsisiPkt` to find the PSI/SI packet that has the earliest transmission time. The `Transmitter` uses the getNextPsisiTime method to find the transmission time of the PSI/SI packets. The `getPsisiPkt` method also updates the next transmission time of the returned packets and reinserts it back to the priority queue.

Several PSI/SI table variables are shared among all tables and thus are abstracted in `Table` class, e.g., `frequency` indicates how often we need to broadcast this table, `pid` is a unique identification number specified by standard documents, `continuityCnt` keeps track of the continuity counter of this PSI/SI table, `nextPktIdx` points to the next TS packet to be sent, and `pkts` are the packed TS packets. In addition to these common variables, `Table` also implements `CreatePkts` method and `update` method. The `update` method synchronizes

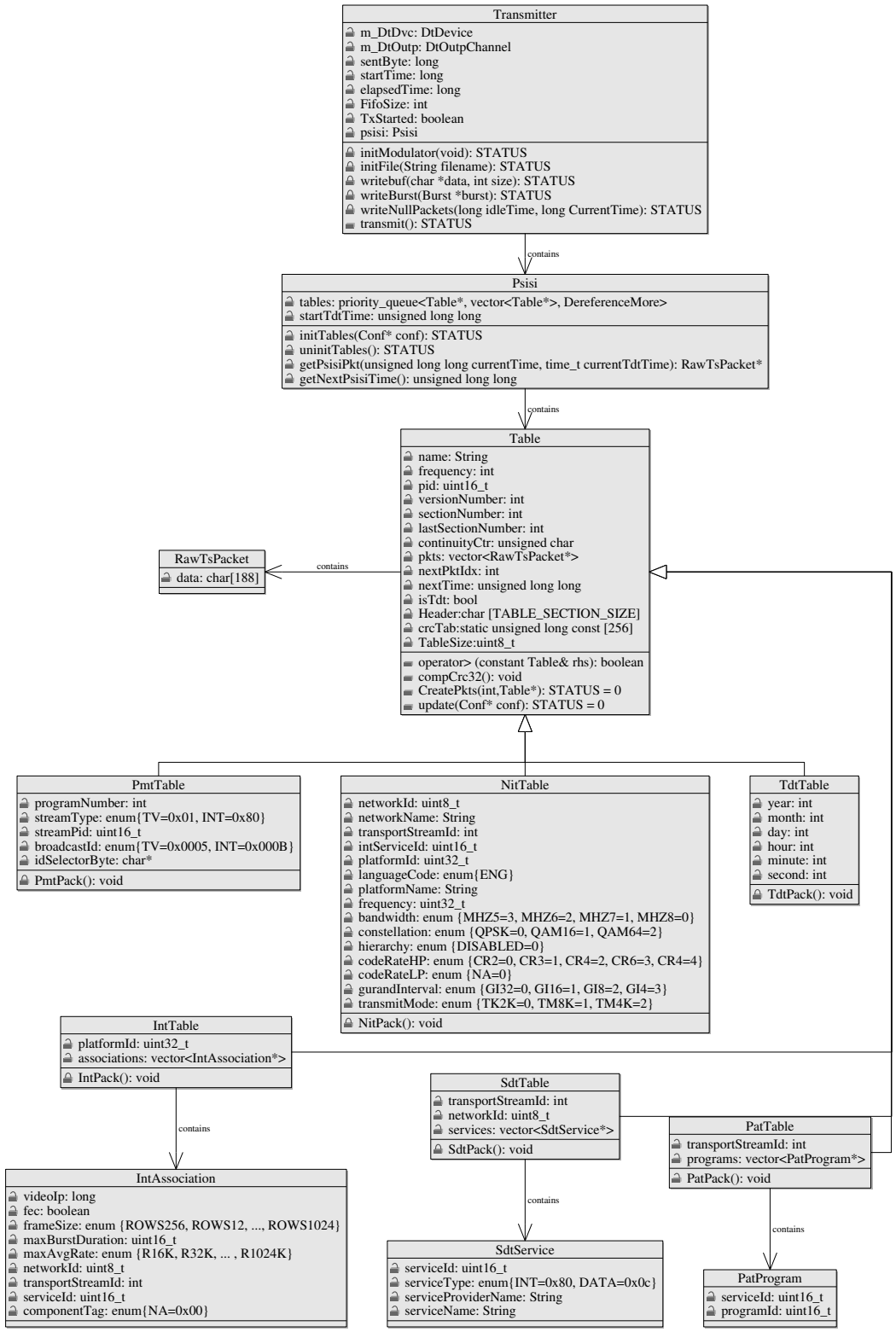


Figure 3.2: Class diagram for transmitter.

PSI/SI table information with the settings in `ConfMgr` while the `CreatePkts` method creates ts packets from the tables and stores them in the `pkts` vector in the `Table` class.

Although there are many more PSI/SI tables, we only implement six essential tables:

- Program Association Table (PAT).
- Program Map Table (PMT).
- Network Information Table (NIT).
- IP/MAC Notification Table (INT).
- Service Description Table (SDT).
- Time and Date Table (TDT).

Each PSI/SI table has a specified Pid and a tableId. The PSI/SI table packets are made up of sections. A section is a syntactic structure used for mapping all the table information into the ts packets. The tableId contained in each section defines the table to which the section belongs. The Pid values mentioned in table 2.1 are used to create the TS packets from the table sections. These sections are stored in the `Header` array of the `Table` class.

We further describe each table in sequence. `PatTable` realizes program association table, which contains the transport stream id and a vector of `PatProgram`, where each `PatProgram` contains the mapping between `serviceId` and `programId`. A Program is a concatenation of one or more events controlled by the broadcaster e.g. news show, entertainment show. `PatTable` lists all available programs and allows mobile devices to extract more detailed program information from `PmtTable`. `PmtTable` implements program map table, which gives the stream pid and stream type for each of the programs available in the network. The stream pid is a unique identification number for each stream and stream type can be either TV or control (INT) channel. With a pid value, mobile devices can identify and reassemble all MPEG-2 TS packets of the same stream.

`NitTable` implements network information table. It contains all physical layer settings of the broadcast network, such as network id, frequency, bandwidth, and modulation scheme, among many others. This table is crucial for mobile devices to decode the received radio signals. `IntTable` realizes IP/MAC notification table, which contains a vector of `IntAssociation` instances. Among other flags, each `IntAssociation` instance associates

a link layer address with an IP address. More specifically, it maps service ids to destination IP addresses. The main purpose of `IntTable` is to inform mobile devices to wait at particular IP addresses for broadcast streams. `SdtTable` implements service description table, which contains a vector of `SdtService` instances. Each `SdtService` maps a service id to a `serviceProviderName` and a `serviceName`, both are descriptive strings. `TdtTable` realizes time and date table, which notifies mobile devices the current date and time at the server side. Unlike all previous tables, `TdtTable` packets are frequently updated during broadcasting.

Each PSI/SI table class also has its corresponding `Pack` function. The `Pack` function in each table class determines the section [1] mapping of the table and segments the table data into a contiguous stream of bits in network byte order which is required to put the data into valid TS packets [35].

3.1.2 ESG Server

The ESG process flow may be broadly classified into three operations - Bootstrap, Acquisition and Update as shown in figure 3.1. We briefly define each of these operations as under:

- ESG Bootstrap - This operation informs the terminal about the available ESGs and how to acquire them.
- ESG acquisition - This operation entails gathering and processing of ESG information by the terminal.
- ESG update - The terminal updates the already acquired ESGs with the latest versions received from the server in this operation.

We describe each of the above operations in the following subsections.

3.1.2.1 ESG Bootstrap

The ESG operations commence once the DVB-H receiver has been started and the terminal has tuned in to a particular transport stream carrying IPDC services. The receiver terminal

[1] A section is a syntactic structure that is used for mapping each ITU-T Rec. H.222.0 — ISO/IEC 13818-1 defined PSI table into Transport Stream packets [35].

receives information about the ESG bootstrap stream from the PSI/SI tables (IP/MAC Notification Table) which contain the well known IP address for the IP stream carrying the bootstrap information. This stream contains two descriptors namely, ESGProviderDiscovery Descriptor and ESGAccessDescriptor. The descriptors are delivered through a FLUTE session with a well known IP address and port (220.0.23.14 for IPv4 and FF0X:0:0:0:0:0:0:12D for IPv6 on port 9214) also indicated in the PSI/SI tables. The ESGProviderDiscovery descriptor indicates the ESGs available from the providers on the given IP platform. The user may use this descriptor to select the ESG to boot with. The ESGProviderDiscovery descriptor is represented as a textual XML file. We provide the syntax of this descriptor in listing. 3.1 and an example in listing 3.2. The ESGAccessDescriptor which is transported as a binary file is a representation of ESG acquisition information related to ESGs indicated in the ESGProviderDiscovery descriptor. The ESGAccessDescriptor is implemented by the `ESGAccessDescriptor` class. We provide a class diagram for the class in Figure 3.3.

Listings 3.1: ESGProviderDiscovery Descriptor Syntax

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <schema targetNamespace="urn:dvb:ipdc:esgbs:2005" xmlns:bs="
       urn:dvb:ipdc:esgbs:2005" xmlns:mpeg7="urn:mpeg:mpeg7:
       schema:2001" xmlns="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified" attributeFormDefault="
       unqualified">
3    <import namespace="urn:mpeg:mpeg7:schema:2001" />
4      <complexType name="ESGProviderType">
5        <sequence>
6        <element name="ProviderURI" type="anyURI"/>
7        <element name="ProviderName" type="mpeg7:TextualType"/
             >
8        <element name="ProviderLogo" type="mpeg7:
             TitleMediaType" minOccurs="0"/>
9        <element name="ProviderID" type="positiveInteger"/>
10       <element name="ProviderInformationURL" type="anyURI"
             minOccurs="0"/>
11       <element name="PrivateAuxiliaryData" type="anyType"
             minOccurs="0"/>
12       </sequence>
13       <attribute name="format" type="anyURI" use="optional"
             default="urn:dvb:ipdc:esg:2005"/>
14       </complexType>
```

```
15        <element name="ESGProviderDiscovery">
16        <complexType>
17        <sequence>
18        <element name="ServiceProvider" type="bs:
              ESGProviderType" maxOccurs="unbounded"/>
19        </sequence>
20      </complexType>
21    </element>
22  </schema>
```

Listings 3.2: ESGProviderDiscovery Descriptor Example

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <ESGProviderDiscovery xmlns="urn:dvb:ipdc:esgbs:2005" xmlns:
       mpeg7="urn:mpeg:mpeg7:schema:2001">
3    <ServiceProvider format="urn:oma:xml:bcast:sg:fragments
         :1.0">
4      <ProviderURI>NSL Broadcast Service</ProviderURI>
5      <ProviderName>NSL Broadcast Service</ProviderName>
6      <ProviderID>1</ProviderID>
7    </ServiceProvider>
8  </ESGProviderDiscovery>
```

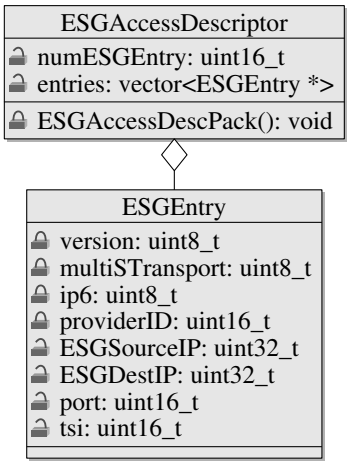


Figure 3.3: Class diagram for ESGAccessDescriptor.

The ESG bootstrap information is transported over a fixed single stream using the FLUTE protocol. Upon completion of the bootstrap process, the terminal acquires the

ESG as selected during the bootstrap operation. Once an ESG entry has been selected by the terminal, the terminal examines the MultiStreamTransport field. We note that we implement only a single transport flow for ESG and hence the value of this field should be set to 0. This means that we do not define ESG session partitioning information which is required for multistream transport. In the single stream mode, the ESG containers are transported as Transport Objects (TO) in a single FLUTE session. These ESG containers encapsulate ESG XML fragments as defined by the ESG data model described in chapter 2.

3.1.2.2 ESG Acquisition

ESG is transported in the form of XML fragments. Once the terminal has acquired the bootstrap information, it is equipped with information required to acquire the ESG XML fragments from the service provider. ESG fragments may be represented in three different formats [18] namely, uncompressed, compressed with GZIP [8] or compressed with BiM [33]. In our implementation we support uncompressed fragments and fragments compressed with GZIP. We now describe the processing operations required to transport ESG to the terminal - ESG Representation and ESG Encapsulation. ESG Representation is the signaling process of indicating which ESG fragments are transported in the current ESG session and in what format. The ESG representation information is sent in the form of the ESG Init Message which initializes the reception of ESG and is transported in the ESG Init Container in single stream transport mode. We present the class diagram for the ESG Init Message in figure **??**.

In the above figure, the `indexingFlag` indicates the availability of fragment indexes which optimizes access to the set of fragments present in the current ESG. The `DecoderInit` class in the ESG Init Message transmits information required for the initialization of decoding or parsing of the ESG XML fragments. The structure of the `DecoderInit` class depends on the encoding used for representation of the ESG XML fragments. We note that we use the `TextualDecoderInit` implementation of the `DecoderInit` class in our ESG server as we only support compressing the XML fragments in GZIP format. The `TextualDecoderInit` class contains information about all `ESGNamespacePrefix` and `ESGXMLFragmentType` data contained in the current ESG. The `ESGXMLFragmentType` announces the type of ESG XML fragment as one of the types defined in table 3.1.

Since ESG data is transmitted in the form of independent XML fragments the fragments are encapsulated into aggregated containers for efficient transport. Aggregating the fragments into containers also enables efficient management including creation, update,

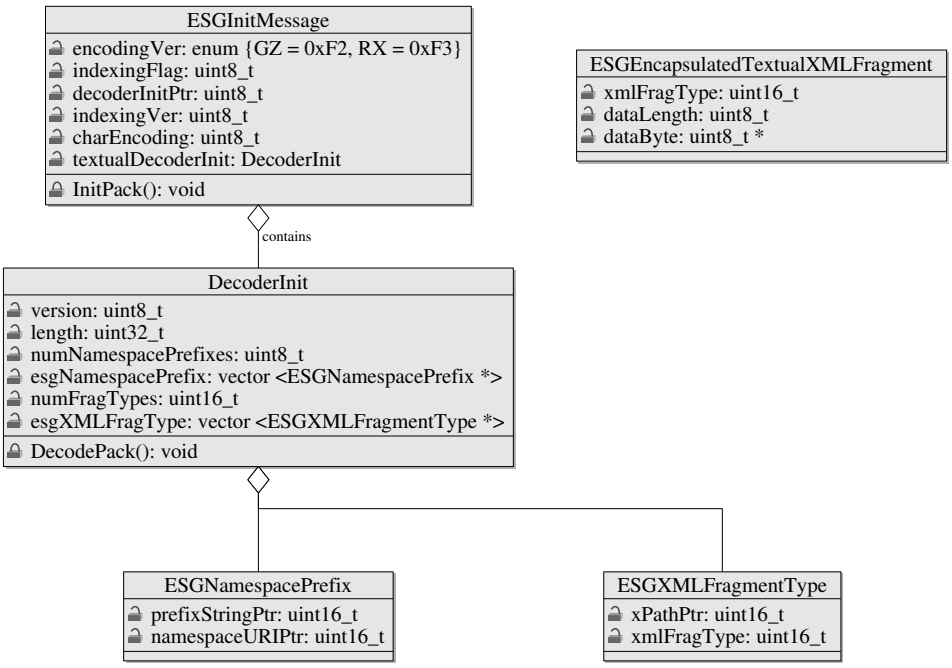


Figure 3.4: Class diagram for ESGRepresentation.

deletion and processing of the fragments. Each ESG fragment is encapsulated by the `ESGEncapsulatedTextualXMLFragment` class. Once the fragments have been represented in the form of the ESG textual XML fragment, they are put into containers before being transported over the medium. Figure 3.5 represents the class diagram of ESG encapsulation. We now describe the classes represented in the encapsulation process. The `ESGContainer` class has a `ESGContainerHeader` that defines the number and type of structures carried in the container and a data body that carries an ESG structure. The `ESGContainerHeader` class stores the structures information in ascending order of `StructType` and `StructId`. The valid values for `StructType` and `StructId` are shown in table 3.2.

One of the structures carried in the `ESGContainer` is the ESG fragment management information structure. The structure is represented in the `ESGFragmentManagementInfo` class which encapsulates information about the identification of fragments carried in the current container. The `ESGEncapsulationHeader` class contains information about the format and interpretation of the `fragRef` field of the `ESGEncapsulationEntry` class. In our implementation, the only accepted value for the `fragMgmtRefFmt` field in the `ESGEncapsulationHeader`

Table 3.1: List of values for `xmlFragType` for different ESG XML Fragments.

Value	ESG XML Fragment Type
0x0020	esg:ESGMain Fragment
0x0021	esg:Content Fragment
0x0022	esg:ScheduleEvent Fragment
0x0023	esg:Service Fragment
0x0024	esg:ServiceBundle Fragment
0x0025	esg:Acquisition fragment
0x0026	esg:PurchaseChannel Fragment

Table 3.2: Structure types `ContType` and their valid structure IDs `ContId`.

Structure Type	Structure ID	Description
0x01	0x00	Fragment Management Information
0x02	0x00	Data Repository of type String
0x03	0x0 to 0xFF	Reserved
0x04	0x0 to 0xFE	Used to identify a specific instance of an index structure, within a container
0x05	0x0 to 0xFE	Used to identify a specific instance of an multi-filed sub-index structure, within a container.
0xE0	0x00	ESG Data Repository
0xE1	0xFF	ESG Session Partition Declaration
0xE2	0x00	ESG Init Message

class is `0x21` indicating a generic ESG fragment reference. This fragment reference is represented by the `ESGFragmentReference` class. The `ESGFragmentReference` class contains the `fragType` field indicating the type of the fragment and the offset of the referenced fragment from the start of the `ESGDataRepository`. We only support encapsulated ESG XML fragments and so the only valid value for `fragType` is `0x00` citeDVBIPDCESG. The `ESGFragmentManagementInfo` class also defines a vector of `ESGEncapsulationEntry` objects. Each of these objects contain a pointer to an `ESGFragmentReference` object. The `ESGEncapsulationEntry` class also contains `fragVer` and `fragId` fields. A increment in the `fragVer` field within an ESG session indicates that the XML fragment being referred to by the current entry is being updated. The `fragId` field uniquely identifies an ESG XML fragment within an ESG fragment stream. The vector `ESGEncapEntry` are sorted in the order of ascending `fragId` to enable efficient location of the fragment by the terminal. The ESG encapsulation process also includes the `ESGDataRepository` class which carries the

actual ESG XML fragments. In our implementation, the `ESGDataRepository` contains an object of the `ESGEncapsulatedTextualXMLFragment` class (defined in figure 3.4) since we support only textual or GZipped XML fragments.

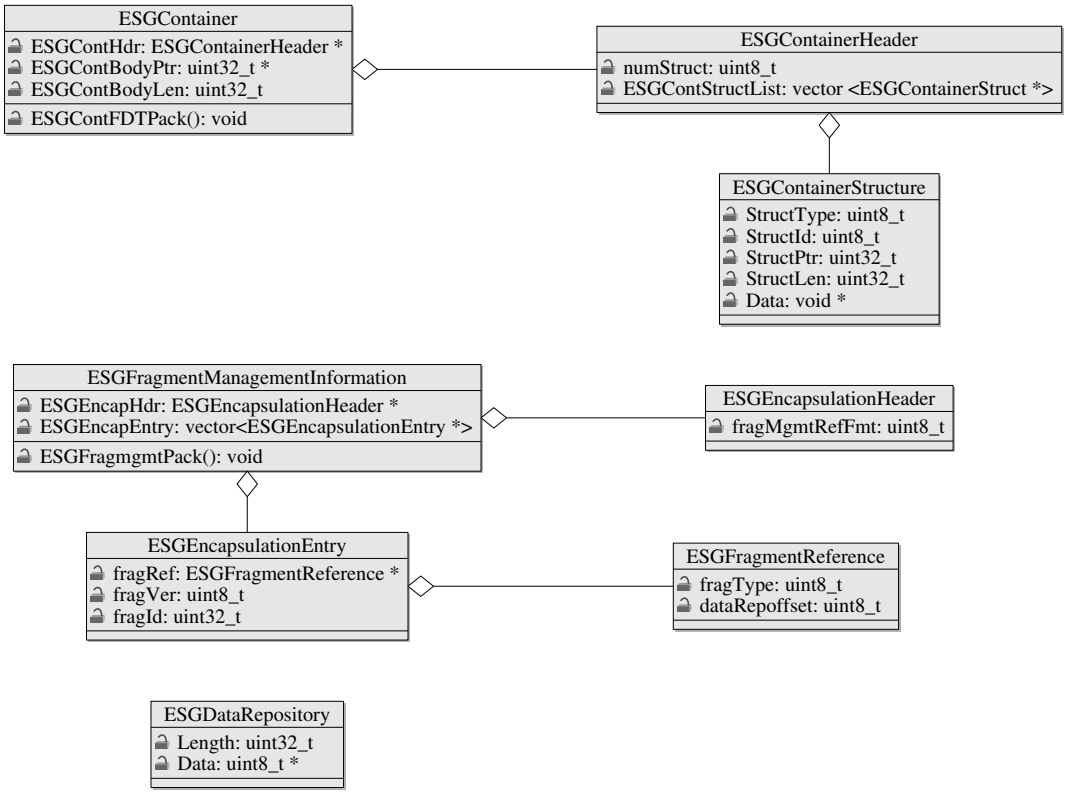


Figure 3.5: Class diagram for ESG Encapsulation.

Finally, the containers of encapsulated ESG data are transported as files in Transport Objects (TO) over FLUTE sessions [17, 44]. ESG containers may be transported in multiple FLUTE sessions distributed over several transport flows when using the multiple stream mode transport mechanism. We implement single stream mode ESG transport. The signaling of ESG containers in FLUTE sessions is dependent on the fields in the File Delivery Table (FDT) specifications. FDTs carry information like the location, version and encoding of the containers. For example, to indicate files as ESG containers the attribute `Content-Type` is set to `application/vnd.dvb.esgcontainer` and to indicate that the container has been compressed using GZIP the `Content-Encoding` attribute is set to `gzip`. An ESG container

is identified by the container ID which is a unique value in a given ESG fragment stream. This container ID is used to form the unique URI of the form <context>:<Container textunderscore ID> where the "<context>" tag is replaced by "urn:dvb:ipdc:esg:cid". An example, URI used in our implementation looks like the following:

urn:dvb:ipdc:esg:cid:23

An example of FDT used to initiate ESG bootstrap is shown in listing 3.3. FDTs for other ESG containers carrying ESG XML fragments are defined similarly.

Listings 3.3: FDT used to initiate the ESG bootstrap session.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <FDT-Instance xmlns="urn:dvb:ipdc:cdp:flute:fdt:2005"
     Expires="4294967295" xmlns:ef="urn:dvb:ipdc:
     esg_flute_extension:2005" ef:FullFDT="true">
3    <File Content-Length="19" Content-Location="
       ESGAccessDescriptor.bin" Content-Type="application/vnd.
       dvb.ipdcesgaccess" TOI="16777279" Transfer-Length="19"/
       >
4    <File Content-Length="430" Content-Location="
       ESGProviderDiscovery.xml" Content-Type="text/xml" TOI="
       16777280" Transfer-Length="430"/>
5 </FDT-Instance>
```

3.1.2.3 ESG Update

The URI defined in the previous subsection is the unique identifier for a file carrying an ESG container and is defined as the value of the `Content-Location` field in the FDT. The versioning information of the file is inferred by mapping the unique `Content-Location` URI to the Transport Object Identifier (TOI) using the FDT. A receiver terminal may receive the same file with multiple TOI values and in such a case, the terminal uses the file with the highest TOI value. The high TOI value signals that the file with that TOI value is the latest version. For updating ESG fragments, the ESG fragment indexing structure may be used to efficiently update individual ESG fragments when updates are available on them.

We summarize the ESG process flow and usage of PSI/SI tables to retrieve ESG information at a receiver terminal below:

1. Receiver terminal retrieves NIT table from the TS. The terminal looks for `descriptor_tag` value of 0x4a when parsing the `network_descriptors` to look for the A value of 0x0B against the `linkageType` field in the `linkage_descriptor` in NIT indicates that an INT is present in the current stream. The `service_id` (for example, say `0x2BBB`) in NIT identifies INT. This `service_id` in SI corresponds to the `program_number` in PSI. The receiver then looks up this `program_number` in PAT to determine the corresponding `PMT_PID` for INT.

2. From PMT, the receiver verifies whether the values of `stream_type` and `data_broadcast_id` are 0x05 and 0x000B respectively. The presence of these values indicate that the PMT corresponds to INT. The `elementary_pid` indicates the PID of the elementary stream carrying INT. The receiver retrieves the INT by looking for `table_id` 0x4C in the elementary stream.

3. Once the INT is available, the receiver then looks for the IP address 224.0.23.14 in the `target_descriptor` to find the `service_id` of the stream in the `operational_descriptor`. In our implementation this `service_id` is `0x0555`. This `service_id` is used by the receiver to look up the corresponding `PMT_PID` in the PAT.

4. In the PMT, if the `stream_type` is 0x90 and the `data_broadcast_id` is 0x0005, the PMT indicates an MPE section. The IP address above signals the `ESG_Bootstrap` service. The `elementary_pid` in this PMT indicates the `ESG_Bootstrap` service. The receiver then extracts the `ESG_Bootstrap` service from the TS.

5. Once the ESG Bootstrap has been retrieved, the receiver uses the TOI to retrieve the two bootstrap files, `ESGAccessDescriptor.bin` and `ESGProviderDiscover.xml`. From these two files, the receiver obtains the Transport Session Identifier (TSI) and uses the TOI value of 0 to get the FDTs for ESG containers transported over FLUTE.

6. The FDT may contain different types of files like GZip type ESG data files, SDP files, JPEG images or video files. The image and video files are used for presenting event information to the user in an attractive manner. We implemented four ESG data models in our testbed - Content fragment, Schedule Event fragment, Service fragment and Acquisition fragment.

7. Content, Schedule Event and Service fragments are used to present program and service information like program names, program synopsis, program start time, program end time, service language and so on in human readable format. The Acquisition fragment provides content access information. It indicates among others the content type and SDP information. We transport the SDP [28] inline, it can be carried in a separate file and pointed to via SDP URI.

8. The value of the attribute `c` in the SDP file indicates a multicast IP address. This is the destination address and is indicated in a `target_descriptor` in the INT. This INT entry is used to obtain the `service_id` which is then used to look up the `PMT_PID` from the PAT. The IP stream can thus be retrieved using the `elementary_pid` obtained from PMT.

3.2 ESG Server Implementation

In this section, we list and describe important sections of the implementation of PSI/SI tables and ESG server implementation. We list only a few code excerpts for reasons of brevity.

3.2.1 PSI/SI Tables Implementation

The design for PSI/SI tables implementation is shown in figure 3.2. We implemented the seven major classes for PSI/SI tables. These classes are summarized in table 3.3 along with their purpose.

The Network Information Table is a System Information (SI) table that communicates the IP stream parameters and the physical network parameters like code rate, frequency, bandwidth etc. It has a PID value of 0x0010 as indicated in table 2.1 and a table ID value of 0x40. As mentioned earlier, the terminal becomes aware of a presence of an IP/MAC Notification Table (INT) and IP streams carrying different services using the linkage descriptor defined in this table. In our implementation the network parameters are configurable and the user may enter desired values into an XML configuration file. The parameters are read from the XML file and populated into PSI/SI tables fields. NIT is the first table to be accessed and processed by a terminal in the DVB-H process flow. A code fragment

Table 3.3: Table describing the seven major classes used to implement PSI/SI tables

Class Name	Purpose
`Psisi`	Initiate and un-initiate priority queues for storing PSI/SI tables. The subclass `Table` also creates raw TS packets to be put in the transport stream delivered to terminals.
`PmtTable`	Implements the Program Map Table. Objects of this class are mapped to programs using the `programNumber` field by the `PatTable` class.
`PatTable`	Implements the Program Association Table. Associates a vector of `Patprogram` to available `PmtTable` objects.
`TdtTable`	Implements the Time and Date Table notifying terminals of current time and date on the server.
`SdtTable`	Implements the Service Description Table and maps service definitions to program definitions on the platform.
`IntTable`	Implements the IP/MAC Notification Table and maps `serviceId` to IP addresses of video streams.
`NitTable`	Implements the Network Information Table. The NIT carries information about the physical attributes of the network carrying the transport streams.

for populating the NIT with configuration parameters is shown in listing 3.4. The non-configurable parameters are populated as per the recommendations of the document ETSI TS 102 470 [25]. Once the parameters have been populated, the tables are prepared for network transmission. As mentioned in subsection 3.1.1, each table class has a corresponding `Pack` function. The `Pack` function converts the values populated in each table into network byte order for transmission.

Listings 3.4: Network Information Table attribute population.

```
1 networkId     = mtv.cfg_mgr->getNetworkId();//should show 0
    x66 in the TS for current stream;
2 networkName   = mtv.cfg_mgr->getNetworkName();
3 serviceId     = mtv.cfg_mgr->getServiceId(); // for INT.
4 platformId    = mtv.cfg_mgr->getPlatformId();//should show 0
    x66 for current stream
5 languageCode = ((lgI=lg.find(mtv.cfg_mgr->getLanguage()))->
    second);//en
6 platformName = mtv.cfg_mgr->getPlatformName(); // SYTE-MTV-5
7 frequency     = mtv.cfg_mgr->getCarrierFrequency();//690 MHz;
8 bandwidth     = (Bandwidth)mtv.cfg_mgr->getBandwidth();//MHZ8
    ;
```

```
9  constellation= ((constlI=constl.find(mtv.cfg_mgr->
       getConstellation()))->second);
10 hierarchy    = ((hierI=hier.find(mtv.cfg_mgr->getHierarchy()
       ))->second); //DISABLED;
11 codeRateHP   = (CodeRateHP)mtv.cfg_mgr->getCodeRateHP(); //
       CR4;
12 codeRateLP   = (CodeRateLP)mtv.cfg_mgr->getCodeRateLP(); //
       NAR;
13 guardInterval= (GuardInterval)mtv.cfg_mgr->getGuardInterval
       (); //GI8;
14 transmitMode = ((trnsMI=trnsM.find(mtv.cfg_mgr->
       getTransmitMode()))->second); //TM8K;
15 transportStreamId = mtv.cfg_mgr->getTransportStreamId();//
       should 1001 for current stream;
```

An excerpt from `PMTPack()` function is shown in listing 3.5. In the listing, the vector `IntAssociation` contains entries of the platform descriptor loop which contains the target descriptor loop and the operational descriptor loop. The target descriptor loop defines IP/MAC addresses of devices while the operational loop defines the actions to be performed on them. The mask values used to convert each field into the network byte order were decided from the size definitions of each field as specified in [22, 26, 25].

Listings 3.5: IP/MAC Notification Table Pack function.

```
1  for (vector<IntAssociation*>::iterator iter = associations.
       begin();iter!=associations.end();iter++) {
2    Ip = ((*iter)->GetVideoIp());
3    fec = ((*iter)->GetFec());
4    fSz = (uint8_t)((*iter)->GetFramesize());
5    //Target desc loop
6    Table::Header[headidx] = (((0x00<<4)&0xf0)| ((
       targetDescLoopLen>>8)&0x0f));
7    Table::Header[(headidx+1)] = (targetDescLoopLen & 0xff);
8    Table::Header[(headidx+2)] = (targetIPSlashDescTag & 0xff)
       ; //Target IP Slash desc tag
9    Table::Header[(headidx+3)] = (targetIPSlashDescLen & 0xff)
       ; //Target IP Slash desc length
10   memcpy(Table::Header+headidx+4,&Ip,4); //Ip address
11   Table::Header[(headidx+8)] = (0x14&0xff); //Ip mask
12   //Operation desc loop
```

```
13    Table::Header[(headidx+9)] = (((0x00<<4)&0xf0) | ((
         operationDescLoopLen>>8)&0x0f));
14    Table::Header[(headidx+10)]= (operationDescLoopLen & 0xff)
         ;
15    Table::Header[(headidx+11)]= (timeslicefecdescTag&0xff);
         //Time slice fec desc tag
16    Table::Header[(headidx+12)]= (timeslicefecdescLen&0xff);
         //Time slice fec desc len
17    Table::Header[(headidx+13)]= (((0x01<<7)&0x80)| ((fec<<5)
         &0x60) | ((0x03<<3)&0x18) | (fSz&0x07)); //Time slicing
          used | fec |reserved |
18    Table::Header[(headidx+14)]= (((*iter)->GetMaxDuration())
         &0xff);
19    Table::Header[(headidx+15)]= (((((*iter)->GetMaxAvgRate())
         <<4)&0xf0) | (0x00&0x0f)); //Avg Rate | time slice fec
         id
20    //IP MAC stream loc desc
21    Table::Header[(headidx+16)]= (IPMACstreamlocdescT&0xff);
         //Desc tag
22    Table::Header[(headidx+17)]= (IPMACstreamlocdescL&0xff);
         //Desc len
23    Table::Header[(headidx+18)]= ((((*iter)->GetNetworkId())
         >>8)&0xff);
24    Table::Header[(headidx+19)]= (((*iter)->GetNetworkId()) &
         0xff);
25    Table::Header[(headidx+20)]= ((0x01 >>8)&0xff); //Original
          network ID: Since the requests generated in the same
         network in the lab.
26    Table::Header[(headidx+21)]= (0x01 & 0xff);
27    Table::Header[(headidx+22)]= (((mtv.cfg_mgr->
         getTransportStreamId())>>8) & 0xff);
28    Table::Header[(headidx+23)]= ((mtv.cfg_mgr->
         getTransportStreamId()) & 0xff);
29    Table::Header[(headidx+24)]= ((((*iter)->GetSId())>>8)&0
         xff);//service ID
30    Table::Header[(headidx+25)]= (((*iter)->GetSId())&0xff);
31    Table::Header[(headidx+26)]= (((*iter)->GetComponentTag())
          & 0xff);
32    headidx += 27;
33  }
```

The creation of the PSI/SI tables is initiated in the `init` function of the `Psisi` class. The `init` functions read the PSIMAP file which contains a mapping that associates a PSIS/SI table name with its transmission frequency. The transmission frequency is used to decide the priority order of each table in the priority queue. The structure `Mtv::DereferenceMore` (not shown in this document) acts as the comparison class for the `Table` objects stored in the `tables` priority queue. The `Table` objects are packed into 188 byte raw TS packets [35] and then placed in the priority queue. The `Psisi::init` function is shown in listing 3.6.

Listings 3.6: `Psisi::init()` function initiating the creation of PSI/SI tables.

```
1  // PSI/SI tables  initialization function
2  void Psisi::initTables() {
3    std::string line;
4    std::ifstream in(PSIMAP); //PSIMAP contains table names
       and table IDs and frequency of transmission of each
       table.
5    while(std::getline(in, line)) {
6      if (!line.empty()) {
7        string::size_type pos = line.find_first_of(":", 0); //
           each line has two item separated by :
8        int freq = atoi(line.substr(0, pos).c_str());
9        string tablename = line.substr(pos + 1);
10       LOG_DEBUG("*PSISI Map information: READ: %s |
           Frequency:%d from tablename:%s",line.c_str(),freq,
           tablename.c_str());
11       Table* table = new Table(freq, tablename);
12       if (table == NULL) {
13         LOG_WARN1("Unable to allocate table, exiting program
             ");
14         exit(-1);
15       }
16       tables.push(table);
17       LOG_DEBUG("Added table: %s, current tables size: %d",
           tablename.c_str(), tables.size());
18     }
19   }
20 }
```

3.2.2 ESG Server Implementation

In this subsection we describe the implementation of ESG server. As discussed in subsection 3.1.2.1 the bootstrap process initiates the ESG session on a terminal. Two kinds of bootstrap descriptors are signaled to the terminal. The `ESGProviderDiscovery` descriptor is transmitted in an XML file as shown in listing 3.2. The `ESGAccessDescriptor` however, is transmitted as a binary file encoded in the network byte order. The `ESGAccessDescriptor` class defines `ESGAccessDescPack()` function to perform the encoding. Listing 3.7 shows the descriptor loop of `ESGAccessDescPack()` function that encodes each `ESGEntry` into `entries` vector containing entries for ESGs transmitted (class diagram shown in figure 3.3). The `AccessDescCont` is a character array holding byte chunks of ESGs defined and transmitted by the server.

Listings 3.7: `ESGAccessDescPack()` function defined in `ESGAccessDescriptor` class.

```
1  for (vector<ESGEntry*>::iterator iter = entries.begin();
       iter!=entries.end();iter++) {
2
3   AccessDescCont[headidx] = (version & 0xff);
4   AccessDescCont[(headidx+1)] = (entryLength & 0xff);
5   AccessDescCont[(headidx+2)] = (((multiSTransport << 7) & 0
       x80) | ((ip6 << 6) & 0x40) | 0x00 & 0x3f); //Multi
       Stream Transport | IPv6 | Reserved
6   AccessDescCont[(headidx+3)] = ((providerID >> 8) & 0xff);
       //Provider ID
7   AccessDescCont[(headidx+4)] = (providerID & 0xff);
8   memcpy(AccessDescCont+headidx+4,&ESGSourceIP,4); //
       Currently we only support IPv4 addresses
9   memcpy(AccessDescCont+headidx+8,&ESGDestIP,4); //Currently
        we only support IPv4 addresses
10  AccessDescCont[(headidx+12)] = ((port >> 8) & 0xff);
11  AccessDescCont[(headidx+13)] = (port & 0xff);
12  AccessDescCont[(headidx+14)] = ((tsi >> 8) & 0xff);
13  AccessDescCont[(headidx+15)] = (tsi & 0xff);
14  headidx += 16;
15 }
```

Listing 3.8 shows an excerpt from `ESGContFDTPack()` function defined in `ESGContainer` class. This function packs the ESG representation information from the `ESGInitMessage`

class and different data structures defined in the ESG encapsulation class diagram in figure 3.5. The data from this function is encapsulated in the `ESGDataRepository` class and transferred to the FLUTE application. We use a customized version MAD-FLUTE [42] application for transfer of ESG files over the FLUTE protocol. The customizations to MAD-FLUTE like adapting MAD-FLUTE to support DVB-IPDC and disabling of congestion control function was done as part of a previous work [40]. The listing below is an implementation of the parameters described in table 3.2.

Listings 3.8: Excerpt from `ESGContFDTPack()` function defined in `ESGContainer` class

```
1  if (StructType == 0x01 && StructId == 0x00)
2    StructPtr = (void *)fragManInfo; // fragManInfo is a
       pointer to an object of
       ESGFragmentManagementInformation class.
3  else if (StructType == 0x02 && StructId == 0x00); // TV
     anytime specification related
4  else if (StructType == 0x03); // Reserved
5  else if (StructType == 0x04); // Not required in this
     implementation
6  else if (StructType == 0x05); // Not required in this
     implementation
7  else if (StructType == 0xE0 && StructId == 0x00)
8    StructPtr = (void*)dataRepo; // dataRepo is a pointer to
       an object of ESGDataRepository
9  else if (StructType == 0xE1 && StructId == 0xFF)
10   StructPtr = (void*) sessPart; // Since we implement a
       single stream mode of transport, this variable is a
       dummy.
11 else if (StructType == 0xE2 && StructId == 0x00)
12   StructPtr = (void *) initMessage; // initMessage is a
       pointer to an object of the ESGInitMessage class that
       carries the ESG representation information.
```

3.3 ESG Server Validation

We use several tools to validate the ESG server. The litmus test though, is the trial run of the ESG server software on the mobile TV testbed. In our testbed, we broadcast the transport streams using low cost DVB-H antennas and use real cell phones (Nokia N96)

with DVB-H receivers to validate the stream. Our apparatus for creating a real DVB-H network is shown in Figure 3.6.

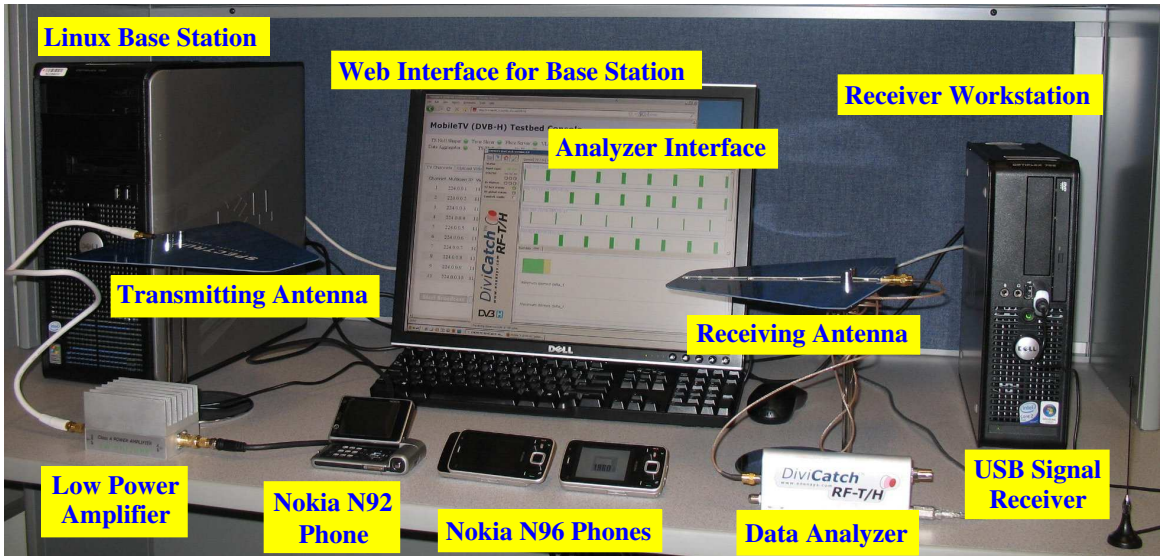


Figure 3.6: DVB-H testbed apparatus.

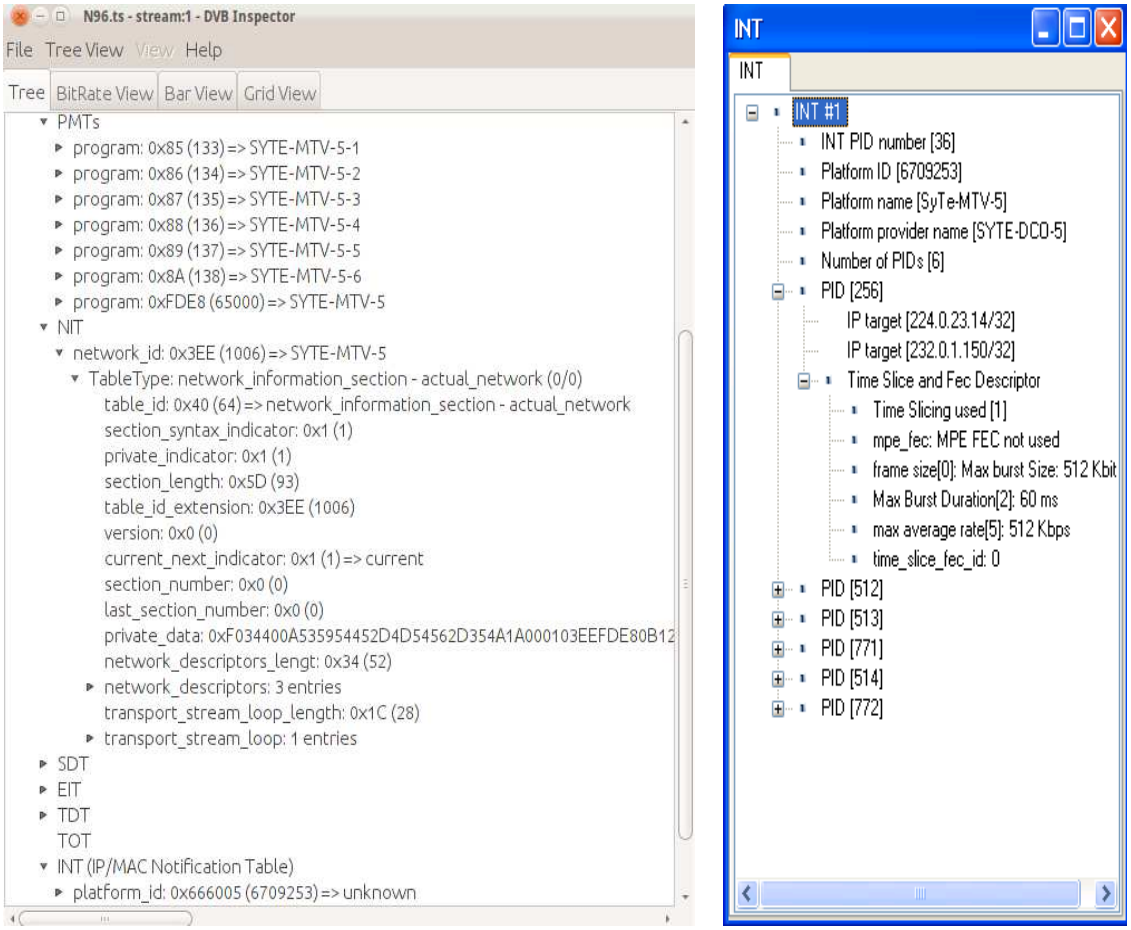
Conventional software testing methods with testing metrics are difficult to implement on our system. We devised two distinct methods of validating our implementation. An offline method where we generate a TS file from DVB-H stream and analyze the file using DVB-H analysis tools to verify the parameters of our ESG server. In this method, we also validate our system through an online analysis tool using which we capture a live DVH-H stream broadcasted from our local base station and analyze the stream in real time to verify our ESG implementation.

In the offline method, we begin by testing the creation of the valid PSI/SI tables. To validate the PSI/SI tables, we create a transport stream (TS) file from the DVB-H stream carrying ESG and service information. We then analyze this file with a DVB-H analyzer software. We verify the TS file with dvbSAM [11] analyzer software which verifies if the given TS file is valid. If the TS file is not valid, dvbSAM does not process the TS file. Implementation errors with PSI/SI tables or invalid values in any of the PSI/SI tables are caught at this stage. Once it is established that the TS file has no errors, we validate the service and program specific information encoded in the TS file against the configuration parameters specified in the configuration file. This is done with an open source Java based

analyzer DVB Inspector [10].

In the online method or testing our software, we generate real DVB-H signals using the apparatus shown in figure 3.6. We generate the DVB-H transport stream on the Linux base station, use the transmitting antenna and the low power amplifier to tranmit the signals. We then use a Nokia N96 phone that has a built-in DVB-H receiver to receive the transport stream transmitted from our base station. Using this phone, we're able to tune into services and use the transmitted ESG generated by our ESG server implementation to select and view channels. Using the phone replicates a real life field scenario where a user tunes into a multiple channel mobile TV service. Using the phone however, limits the analysis of the signals transmitted. In our testbed apparatus, we also have a receiving antenna which captures the signals transmitted by the base station. These captured signals then pass through an analyzer hardware and software, Divicatch RF-T/H[9]. The analyzer software allows us to visualize the transport stream data in several different ways. The analyzer shows the bursts as they are sent over the network. A screenshot of DVB-H bursts is shown in figure 3.8. In the figure, the green bars represent bursts of transport stream being broadcasted. We note that there are four available services seen in the four rows of bursts in the figure. The bursts in the first row represent ESG information and are thus thinner than the other three rows which represent mulimedia content bursts. Also, the services generated by our base station in the transport stream multiplexes the mulimedia channels and hence the bursts in the third and fourth rows are aligned. The color of the green bars indicates that the bursts have no errors. Bursts with errors are seen with yellow and red bars varying with the recoverability of the error.

We vary the configuration of the parameters in the PSI tables and use DVB Inspector to verify the changes are reflected in the TS file. For example, we vary the PIDs of programs and figure 3.7 shows this variation. In figure 3.7(b) we see that the included programs have serviceId (also referred to as the program number) ranging from 133 through 136 for the four available programs (the programs with serviceId 137 and 138 relate to ESG information) and in figure 3.7(a) we note that there are three programs with serviceId ranging from 512 through 514 (the programs with serviceId 771 and 772 relate to ESG information). It is possible to verify the values of parameters transmitted in PSI/SI tables with both the Divicatch analyzer as well as DVB Inspector. While Divicatch captures data from a live broadcast stream, DVB Inspector can only verify transport stream files. We verify the transport stream in both applications. Figure 3.7 shows screenshots of PSI/SI tables as



(a) PMT on DVB Inspector (b) INT on Divicatch

Figure 3.7: PSI/SI tables validation

seen on Divicatch and DVB Inspector respectively. We note that there are seven programs listed under PMT in the screenshot from DVB Inspector. The last program on the list with program number 0xFDE8 indicates that there is an INT in the given stream. The fifth stream with program number 0x89 is the ESG bootstrap stream while the sixth program with program number 0x8A is the ESG stream. Figures 3.7(a) and 3.7(b) are screenshots taken during distinct tests visualizing distinct TS files. Of the remaining four programs three programs with PIDs 513, 514 and 512 are visible as services in figure 3.7. The burst row with PID 256 in the figure represents the ESG bootstrap stream.

The Divicatch software can validate ESG information as well. Figure 3.9 shows the ESG

Figure 3.8: DVB-H bursts on Divicatch RF-T/H analyzer

channel received by the analyzer on the broadcast network. As it can be seen in the figure, the ESG type is chosen as IPDC. Once the correct ESG standard is chosen, the analyzer is able to identify the IP platform on the current stream that carries the ESG. Listening at the well known port number and the IP from bootstrap, the analyzer starts receiving the ESG information. On a successful bootstrap, the button against `Bootstrap state` field in the figure turns green. In the message flow window, the bootstrap files are indicated while the ESG service providers window reads information from the ESG XML files. At the analysis level, receiving the ESG XML files concludes the verification of our ESG server implementation. Beyond this point, to verify that the ESG XML files are transmitted in the right format, we use the Nokia N96 phone to view the available channel information after tuning into the available DVB-H signals in our lab (generated by our base station). After tuning into the broadcast, we verify that the phone is able to select and view multimedia content served through the available channels. The visibility of different channels on the phone validates the implementation of the ESG server in the context of representation. The

ability to tune in to each channel validates the implementation of the ESG server at the system level and verifies that the correct transmission of PSI/SI tables.

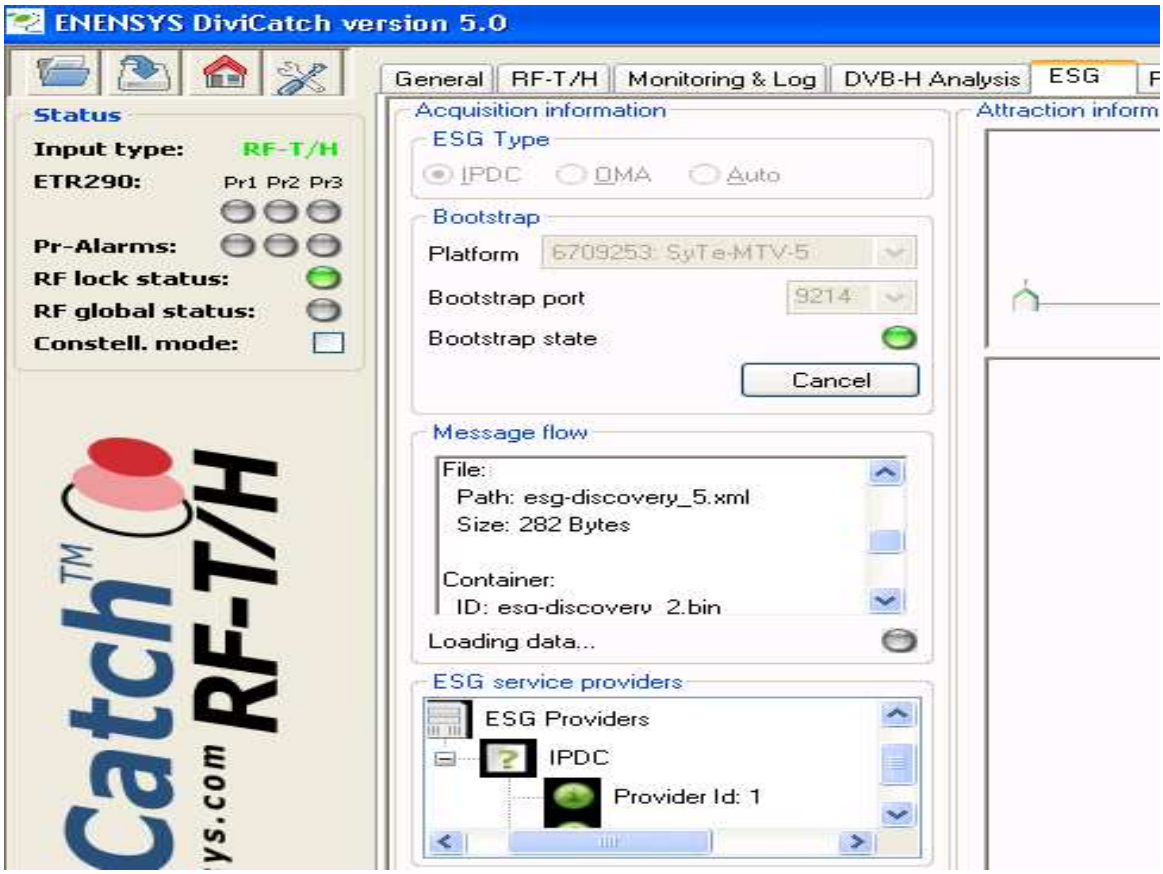


Figure 3.9: ESG information from transport stream on Divicatch RF-T/H analyzer

Chapter 4

Conclusion

In this chapter, we first summarize the contributions of this project. Then, we describe some future extensions of this work.

4.1 Conclusions

We designed and implemented a C++ based Electronic Service Guide server for mobile TV testbed based on DVB-H. We followed the DVB-IPDC standard for designing our ESG server as it has provisions to operate over any IP based multimedia delivery mechanism. IPDC also has provisions for collaborative delivery of multimedia with cellular technologies like 3G/UMTS. In order to implement ESG server over DVB-H, it is also required to implement the link layer signaling mechanism for DVB-H namely, PSI/SI tables. The testbed originally integrates python based open source to implement PSI/SI tables. PSI/SI tables implemented as part of this project are designed to interface with uniform and consistent program configurations on the testbed and ESG server. The PSI/SI tables and ESG server are used in combination by a mobile terminal receiving multimedia streams to present TV programming information including descriptions and schedules to the user and to obtain physical network parameters to tune into available services.

We validated our implementation using several verification tools to ensure error free delivery of transport stream. We used our live mobile TV testbed apparatus to reproduce real life field scenarios to verify our implementation. When running the system with our low power antennas, we were able to get consistent programming information on the mobile terminal (Nokia N96 phone) as configured in the ESG server. We used online USB based

transport stream analyzers as well as offline analyzers to monitor errors in the implementation. This implementation provides an easy to configure, consistent end-to-end mobile TV testbed.

4.2 Future Work

The work in this project can be extended in several directions. Some of them are summarized below:

- Hybrid delivery mechanisms are an attractive avenue considering the exponentially increasing video traffic on mobile devices. Because this project uses the DVB-IPDC specification, extensions to this project can be used to experiment with collaborative delivery of mobile video using DVB-H along with 3G/UMTS, LTE, WiMAX or other network technologies.
- With the ratification of the DVB-IPDC 2.0 standard, this work can be extended to implement additional features like interactive on demand services over unicast and provisions for sending notifications to terminals.
- A similar implementation of an ATSC [4] based mobile TV testbed can be designed and implemented based on the design of this project.
- A larger multiple technology testbed that can choose different network technologies from a list and test compatibility of unicast and broadcast networks can also be developed as an extension to this work.

Bibliography

[1] CMMB technology overview, 2009.

[2] Open Mobile Alliance. Service Guide for Mobile Broadcast Services: v1.0, February 2009.

[3] Association of Radio Industries and Businesses (ARIB); Transmission System for Digital Satellite Broadcasting:ARIB STD–B20, May 2001.

[4] Advanced Television Systems Committee (ATSC); ATSC–Mobile DTV Standard, Part 1 ATSC Mobile Digital Television System, June 2011.

[5] Hsin-Ta Chiao, Chi-Te Tseng, Jhih-Wei Jiang, and Hsin-An Hou. Hybrid streaming delivery over DVB-H broadcast and WiMAX mobile networks. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2010 IEEE 6th International Conference on*, pages 398 –405, oct. 2010.

[6] S. Cho, G. Lee, B. Bae, K. Yang, C. Ahn, S. Lee, and C. Ahn. System and services of Terrestrial Digital Multimedia Broadcasting (T-DMB). In *IEEE Transactions on Broadcasting, 53(1):171178*, March 2007.

[7] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011 – 2016, February 2012.

[8] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informational), may 1996.

[9] Divi Catch RF-T/H transport stream analyzer, http://www.enensys.com/, 2009.

[10] DVB Inspector v0.0.3, http://www.digitalekabeltelevisie.nl/dvb_inspector/, 2011.

[11] dvbSAM Standard Edition v5.0, www.decontis.com, 2011.

[12] Digital Video Broadcasting (DVB); Framing Structure, channel coding and modulation for Satellite Services to Handheld devices (SH) below 3 GHz. European Telecommunications Standards Institute (ETSI) Standard EN 302 583 Ver. 1.1.2, February.

[13] Digital Video Broadcasting (DVB); DVB framing structure, channel coding and modulation for digital terrestrial television. European Telecommunications Standards Institute (ETSI) Standard EN 300 744 Ver. 1.5.1, June 2004.

[14] Digital Video Broadcasting (DVB); Transmission System for Handheld Terminals (DVB-H). European Telecommunications Standards Institute (ETSI) Standard EN 302 304 Ver. 1.1.1, November 2004.

[15] Digital Audio Broadcasting (DAB); Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers, European Telecommunications Standards Institute (ETSI) Standard EN 302 401 Ver. 1.4.1, January 2006.

[16] Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Architecture. European Telecommunications Standards Institute (ETSI) Standard EN 102 69 Ver. 1.1.1, May 2006.

[17] Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocol. European Telecommunications Standards Institute (ETSI) Standard EN 102 472 Ver. 1.1.1, November 2006.

[18] Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Electronic Service Guide (ESG). European Telecommunications Standards Institute (ETSI) Standard EN 102 471 Ver. 1.2.1, November 2006.

[19] Digital Video Broadcasting (DVB); DVB-H Implementation Guidelines. European Telecommunications Standards Institute (ETSI) Standard EN 102 377 Ver. 1.3.1, May 2007.

[20] Digital Video Broadcasting (DVB); IPDC over DVB-H: Electronic Service Guide (ESG) Implementation Guidelines. European Telecommunications Standards Institute (ETSI) Standard EN 102 592 Ver. 1.1.1, October 2007.

[21] Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 2: System aspects in a uni-directional environment. European Telecommunications Standards Institute (ETSI) Standard TS 102 822-3-2 Ver. 1.5.1, May 2009.

[22] Digital Video Broadcasting (DVB); DVB Specification for Data Broadcasting. European Telecommunications Standards Institute (ETSI) Standard EN 301 192 Ver. 1.5.1, November 2009.

[23] Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocol. European Telecommunications Standards Institute (ETSI) Standard EN 102 472 Ver. 1.3.1, June 2009.

[24] Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Electronic Service Guide (ESG). European Telecommunications Standards Institute (ETSI) Standard EN 102 471 Ver. 1.3.1, April 2009.

[25] Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Program Specific Information (PSI)/Service Information (SI). European Telecommunications Standards Institute (ETSI) Standard EN 102 470 Ver. 1.2.1, March 2009.

[26] Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems. European Telecommunications Standards Institute (ETSI) Standard EN 300 468 Ver. 1.11.1, April 2010.

[27] A. Hammershøj, G. Pedersen, and R. Tadayoni. Open source end-2-end DVB-H mobile TV services and network infrastructure - The DVB-H pilot in Denmark. In *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pages 644 –648, may 2009.

[28] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), jul 2006.

[29] Lawrence Harte and Steve Clee. *Introduction to Mobile Video: How to Send Live TV and Streaming Video to 2G and 3G Multimedia Telephones*. Althos Publishing, 2008.

[30] Mohamed Hefeeda and Cheng-Hsin Hsu. Design and evaluation of a testbed for mobile tv networks. *ACM Transactions on Multimedia Computation, Communications and Applications*, 8(1):3:1–3:23, feb 2012.

[31] C. Heuck. An Analytical Approach for Performance Evaluation of Hybrid (Broadcast/-Mobile) Networks. *Broadcasting, IEEE Transactions on*, 56(1):9 –18, march 2010.

[32] Chen-Chiung Hsieh, Chao-Hsien Lin, and Wen-Tsung Chang. Design and implementation of the interactive multimedia broadcasting services in DVB-H. *Consumer Electronics, IEEE Transactions on*, 55(4):1779 –1787, november 2009.

[33] International Organization for Standardization (ISO); Information technology – Multimedia content description interface – Part 1 : Systems. ISO/IEC 15938-1, 2002.

[34] International Organization for Standardization (ISO); Information technology – Coding of audio-visual objects Part 10 : Advanced Video Coding. ISO/IEC 14496-10, May 2003.

[35] International Organization for Standardization (ISO); Information technology – Generic coding of moving pictures and associated audio information : Systems. ISO/IEC 13818-1:2007, October 2007.

[36] Tero Jokela and Jani Väre. Simulations of PSI/SI Transmission in DVB-H Systems. In *IEEE International Symposium on Broadband Multimedia and Broadcasting 2007*, 2007.

[37] Erkki Aaltonen Jyrki T.J. Penttinen, Petri Jolma and Jani Väre. *The DVB-H Handbook: The Functioning and Planning of Mobile TV*. John Wiley and Sons Ltd., 2009.

[38] Michael Kornfeld and Gunther May. DVB-H and IP Datacast–Broadcast to Handheld Devices. *Broadcasting, IEEE Transactions on*, 53(1):161 –170, March 2007.

[39] Amitabh Kumar. *Implementing Mobile TV (Second Edition): ATSC Mobile DTV, MediaFLO, DVB-H/SH, DMB, WiMAX, 3G Systems, and Rich Media Applications*. Elsevier Inc., 2010.

[40] Yi Liu and Mohamed Hefeeda. Video streaming over cooperative wireless networks. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, MMSys '10, pages 99–110, New York, NY, USA, 2010. ACM.

[41] T. Lohmar and U. Horn. Hybrid Broadcast-Unicast distribution of Mobile TV over 3G Networks. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 850 –851, nov. 2006.

[42] MAD-FLUTE project, http://mad.cs.tut.fi/, 2009.

[43] M. Oksanen and I. Defee. Mobile broadcast testbed development and results. In *Proceedings of the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting 2006, Las Vegas, NV, USA*, April 2006.

[44] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. FLUTE - File Delivery over Unidirectional Transport. RFC 3926 (Experimental), oct 2004.

[45] Thilo Pape and Veronika Karnowski. Which Place for Mobile Television in Everyday Life? In Corinne Martin and Thilo Pape, editors, *Images in Mobile Communication*, pages 101–120. VS Verlag fr Sozialwissenschaften, 2012. 10.1007/978-3-531-93190-6-6.

[46] C. Sattler. Seamless unicast and broadcast services to meet increasing mobile tv and video consumption, 2009.

[47] R. Schatz, N. Jordan, and S. Wagner. Beyond Broadcast–A Hybrid Testbed for Mobile TV 2.0 Services. In *Networking, 2007. ICN '07. Sixth International Conference on*, page 87, april 2007.

[48] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), jul 2003. Updated by RFCs 5506, 5761, 6051, 6222.

[49] W3C Recommendation: XML Schema, May 2001.