An SNMP Agent for a DTV Data Server

by

Dinkar Bhat
David Catapano
James Kenealy
Gomer Thomas

Abstract

This paper presents a framework for remote control and monitoring of a DTV data server using the Simple Network Management Protocol (SNMP). This standards-based framework consists of a remote manager, agent(s) that access the properties of the data server, and a management protocol that specifies the format of transfer of information between agents and the manager. The information to be accessed by the agent is provided in the form of a Management Information Base (MIB). We describe the process of developing a MIB for a generic DTV data server. The SNMP framework clearly separates management of the data server from the function of data serving, and also facilitates a consistent user interface to monitor and control diverse sub-systems in a broadcast plant, possibly provided by different vendors.
Introduction

With the advent of digital television, broadcasters are seeking ways to maximize the revenue they can generate with their digital bandwidth, in part to offset the large cost for upgrading their infrastructure. Data broadcasting offers new revenue opportunities. However, for reliable and efficient operation many issues have to be dealt with, including the need for controlling and monitoring their data server(s). We present a framework based on the Simple Network Management Protocol (SNMP) for remote control of a data server.

The organization of this paper is as follows. First, we present an overview of DTV data broadcasting, describe a typical station environment, and discuss the motivation for SNMP-based control in this environment. We then discuss the architecture and features of a data-server, and the specific remote control features needed. Then, we describe SNMP and the framework that it provides for data server management, give an example MIB, and present the related agent implementation architecture. Finally, we discuss future work on how the MIB may be extended.

Overview of DTV Data Broadcasting

First, an overview of DTV data broadcasting is in order. In an ATSC DTV stream, a fixed bandwidth of 19.39 Mb/s is available. Part of the stream is used for video and audio packets. Data packets can occupy some or all of the remaining bandwidth.

A DTV data server as shown in figure 1 inserts data packets into the broadcast stream. A data server may insert data directly in the form of MPEG-2 packets, or it may first send IP packets to an IP-MPEG2 gateway, which encapsulates them into MPEG-2 packets and then inserts them into the broadcast stream.

![Figure 1. DTV Data Server Injecting Data](image)

During a high-definition (HD) television broadcast, video and audio occupy a substantial portion of the overall bandwidth, and therefore only a modest portion of the transport stream can be used for data broadcasting. On the other hand, if a single standard definition (SD) virtual channel is broadcast, more bandwidth is available for data broadcasting. More than one standard definition program can be multiplexed in a stream, in which case less bandwidth becomes available for data. Some stations broadcast high-definition television for part of the day, and standard-definition for the rest, and hence
differing amounts of bandwidth may be available for data broadcasting over the period. Moreover, there may be second by second variations in the bandwidth used for the video and audio, because of variations in the video characteristics.

Under the changing bandwidth scenario, data services can be classified into profiles, according to the bandwidth that a data server needs to allocate to them. For instance, while one data service may require a guaranteed maximum bandwidth, another may only require “best-effort” or opportunistic bandwidth1. (i.e., the multiplexer attempts to fill the stream with data as and when bandwidth becomes available). Multiple data streams with different profiles may be in the same transport stream.

**Station Environment**

In a broadcast station, there are several sub-systems required for creating a broadcast stream. They include: a) encoders for audio/video, b) a traffic system which creates playlists containing precise time schedules of audio/video/data events, including intermediary commercials, c) data server(s) d) an automation system that actually triggers the encoders and data server(s), e) a PSIP generator which is responsible for inserting program guide information, and f) the multiplexer. These sub-systems must work in perfect tandem.

**Figure 2. Station Environment**

Presented at 35th Annual SMPTE Advanced Motion Imaging Conference, Feb 8-10, 2001
Motivation for Remote Control

Operators of transmission equipment including the data server are not necessarily located close to the actual system installations. In fact, the data servers may be distributed over multiple stations in a station network, and they may have to be monitored and controlled by one control center far away from any of the data servers.

A standards-based remote monitoring and control approach like one based on SNMP is desirable because it allows uniform management of diverse items of equipment provided by different vendors. If each item of equipment was managed by a proprietary mechanism, then it might not be possible to have a consistent management interface for the overall system.

SNMP also helps in clearly separating management from the data server functionality. Thus, the data server software can be maintained independent of the management software. Also, management software can be made available by third parties. Hence the system integrator is saved the costs of developing custom management framework, and at the same time may utilize the expertise of experienced generic-SNMP software developers.

Thus, the motivation for SNMP-based management is both cost saving and uniform user controls across different vendor-supplied equipment/applications.

DTV Data Server

Conceptually, there are three major components in the data broadcasting system, namely the data provider, the data server and the data receiver (see Figure 3.)

The data provider provides data and schedules for data transmission from one or more data servers. Each data provider could be viewed as an "account" at the data-server, since the data-provider typically contracts for bandwidth to be utilized for data transmission. The data server receives data, possibly from more than one data provider, encodes data into MPEG-2 packets and hands the packets to a multiplexer, according to schedules specified by the data providers. The data receiver allows the user to tune to channels containing data streams, decode data, and launch players based on the type of data.

The data server software controls several key parameters for the data provider accounts:

- Enabling and disabling of transmission from data provider accounts,
- Bandwidth to be allocated to accounts, including the amount of guaranteed bandwidth and the amount of opportunistic bandwidth (both of which may be dependent on date and time of day),
- The output PID in the MPEG-2 transport stream to be used for each account,
- The output interfaces for data from each account (in situations where data from different accounts may be going into different broadcast streams),
- Input interface over which data may be received from each data-provider.

In addition, the data-server can include re-configurable global parameters, such as:

- The output interface to be used to transmit data to the multiplexer (in situations where all data goes into the same broadcast stream),

Presented at 35th Annual SMPTE Advanced Motion Imaging Conference, Feb 8-10, 2001
The time slice to be used for interleaving data from multiple data providers (for tuning).

Also, for metering and billing data providers, statistics like the total bandwidth actually consumed by different data provider accounts over specified periods should be gathered and reported.

Thus, there is a clear requirement for monitoring and control of the data server.

Figure 3 shows a particular software architecture with the three major components in data broadcasting. The Data Fab maps to the data provider, the Data Hub maps to the data server located at a broadcaster, and the Data Receiver corresponds to the data consumers. Remote control and monitor of the Data Hub through SNMP is the focus of this paper.

Figure 3. Three-Component Architecture for Data

SNMP Framework

The genesis of SNMP-based management is the Internet Engineering Task Force (IETF), an open organization to create standards for the Internet. The initial targets for this effort were TCP/IP routers and hosts. However, the SNMP-based management approach is inherently generic so that it can be used to manage many types of systems. It can be used with computer data networks, data servers, MPEG-2 gateways, etc. The SNMP model of a managed network (illustrated in Figure 4) consists of the following elements:

- One or more managed nodes, each containing a processing entity called an agent.
- At least one management station containing management applications or managers.
- Management information at each managed node that describes configuration, statistics and state of the node.
- A management protocol that the managers and agents use to exchange messages.

Different types of management information can be specified for a node:
- monitor-type like status indicators or counters,
- control-type which can initiate actions to occur, like starting a session
- configurable-type to fine-tune performance of the node.
In data broadcasting, an example of monitor-type variables would be bandwidth usage by different data provider accounts, an example of control variables would be starting and stopping of transmission from an account, and an example of configurable-type variables would be specifying the output interface to which the data server should transmit data.

There are three types of messages that occur in an SNMP managed network:

- request message generated by a manager for an agent to retrieve and/or modify management information of a node.
- response message generated by an agent to satisfy a request message from the manager.
- event message generated by an agent to report events to a manager.

Thus, the operations in SNMP are limited to retrieving and modifying management information.

Figure 4. Generic SNMP Framework for Management
Another highlight of SNMP is that message exchange between an agent and a manager only requires a simple connectionless transport service like UDP (User Datagram Protocol). Thus, the manager and the agent can be located on the Internet. In the data broadcasting architecture, the agent would be located at the data server, and the manager could be located on any machine as long as UDP packets can be routed from the agent to the manager, and vice versa.

Specifications containing definitions of management information are called Management Information Bases (MIBs). The rules for writing MIBs are defined in a collection of IETF documents called the Structure of Management Information (SMI). Each MIB consists of several modules that can be classified into two categories: managed object modules, and information modules. Managed object modules contain definitions of management information. Information modules provide supplementary information that the managed object module would use, and do not contain managed object definitions. A managed object module is the most complex and most important to design carefully since it models how a node is to be managed. In the next section, we describe an example managed object module for a data server. We leave out the information modules for clarity sake.

SNMP version 2, commonly referred to as SNMPv2, is the current version of SNMP, SNMPv1 being the earlier version. SNMPv2 offers a richer and more precise syntax for MIBs, however, SNMPv1 is more widely deployed, and hence MIBs written in this SNMPv1 are likely to be more portable. On the other hand, IETF-produced MIBs must be in SNMPv2. In our definition of the MIB, we use SNMPv2.

**Implementation of SNMP**

There are two stages in implementing an agent for a data-server, namely the design of the MIB, and the design of the agent software that includes methods to access management information from the data-server.

**Defining the MIB**

The first stage in designing a managed object MIB module for a node is to identify the components that comprise it, and the monitor-type, control-type, and configurable-type variables that define the components. This stage is called *object modeling*. Once the object model has been developed, it can be quite easily translated into a MIB module using SMI.

As mentioned earlier, there are several parameters that may have to be controlled and monitored in the data server. Here, for clarity, we define a subset of those parameters in the management information, and build an example MIB. A data-server may receive data and transmission schedules from one or more data-provider accounts; hence an account can be viewed as a component of the data-server. Let us say that we need to control enabling/disabling of each account, and we need to monitor the bandwidth consumed by each account. Further, let us suppose that the interface for data output from each account would need to be configured dynamically. Finally, in addition to enabling individual...
accounts, let us say that it is required to start and stop the data-server itself. For modeling, bandwidth consumption can be represented as a monitor-type object, the switch to enable an account/data-server can be seen as a control-type object, and the output-interface can be controlled through a configuration-type object. This model can be translated into the example MIB shown in appendix A.

There are other extensions to the MIB. For example, we did not describe events that can be generated by the agent. Typical events would include exceptions that the manager must be informed about, for instance, if the data server malfunctions, if the total bandwidth required by all accounts exceed the total bandwidth allowed, or the communications connection to the encoder is broken. It is quite important to model events in a MIB because they report unexpected problems at a node.

Once the MIB is generated, the next stage is to compile it. A compiler would help in validating the structures defined in the MIB, and producing data structures (in C, Java) that can be used in production of manager or agent code. Typically, compilers also provide graphical views of the MIB. There are several compilers available. A publicly available compiler is SMIC, provided by Bay Networks.

Agent Software

One can build a monolithic agent software architecture where the agent is part of the data server code. A better manageable and extensible architecture would be one in which the agent software and the data server software are clearly separate. In this case, the data server software has well-defined interfaces for SNMP management, and agent uses the published methods to access management information. The agent must include the following software:

- Methods to receive and transmit messages using a transport stack.
- Methods to access MIB-defined management information from the data-server. This could be achieved through a dispatch table that contains mappings to data server method routines based on management information identity.
- Methods for authenticating requests from manager(s) and decoding them.
- Methods to receive events from the data-server, and then generate event messages/traps to the manager.

One can further modularize the agent software. There may be multiple subagents, each responsible for a subset of the management information, and an encapsulating agent that receives requests from managers, and dispatches them appropriately to these subagents. The transport mechanism between the manager and the encapsulated agent is typically standard UDP, but the mechanism between the subagents and the encapsulated agent could be proprietary. One issue of this approach is that the management information has to be kept consistent across different subagents.

The kind of architecture used to implement the SNMP agent is transparent to the manager(s). The manager simply transmits and receives messages as if it is talking to one agent, even if the above multiple agent approach was used.
Summary

We discussed the SNMP framework for remote management. We presented an example MIB for a data server, and discussed its structure and syntax. SNMP offers a convenient framework for management and control of diverse sub-systems, even when different vendors provide the subsystems. The wide acceptance of SNMP and consequently the large expertise that is available in this area make SNMP attractive for system builders. Also, with support for SNMP being available in many languages, like Java², it becomes a viable alternative for software developers. Hence, it may be a good solution for the broadcast industry.

References


Appendix

A) The following is the definition of an example MIB for the data-server defined in section 5.

```
DATASERVER-CONTROLLER-MIB DEFINITIONS ::= BEGIN
-- definitions to be imported from other MIB modules defined elsewhere.
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Unsigned32, Counter32
FROM   SNMPv2-SMI
    dataServerSystem, dataServerModules
FROM   DATASERVER-GLOBAL-REG;

dataServerControllerMibModule MODULE-IDENTITY
    LAST-UPDATED  "0101081200Z"
    ORGANIZATION "Triveni"
    CONTACT-INFO
        "Dinkar Bhat
        email: dbhat@3veni.com"
    DESCRIPTION
        "The data server controller MIB"
    REVISION  "0101081200Z"
    DESCRIPTION  "First revision"
::== { dataServerModules 2}

-- root for items in the controller MIB module
dataServerControllerMIB       OBJECT IDENTIFIER ::= { dataServerSystem 1}

-- subtree for objects, and for each functional area
dataServerControllerObjs     OBJECT-IDENTIFIER ::= {dataServerControllerMIB 1}

-- the objects themselves

-- a) object that represents the state of the datahub (active or inactive). The object
-- is read-write since the manager can change the state of the hub.
dataHubState       OBJECT-TYPE
    SYNTAX       INTEGER { active(1), inactive(2) }
    MAX-ACCESS   read-write
    STATUS       current
    DESCRIPTION
        "The state of the data server. If the data server is turned on, then all
        accounts that comprise this server can transmit data".
::== { dataServerControllerObjs 1 }

-- b) the data provider accounts that are handled by the data server can be viewed as a
-- table with each row in the table corresponding to an account.
accountTable       OBJECT-TYPE
    SYNTAX       SEQUENCE OF AccountEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION  "The table of accounts handled by the data server"
::== { dataServerControllerObjs 2 }

-- c) definition of a row in the account table. Note that entries in the account table
-- accounts - cannot be created by SNMP operations.
```
accountEntry OBJECT-TYPE
SYNTAX AccountEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "A row in the account table"
INDEX {accountIndex} -- this identifies an account in the table.
::= {accountTable 1}

AccountEntry ::= SEQUENCE {
    accountIndex   Unsigned32, -- The index to the table of accounts
    accountBandwidth Counter32, -- Identifies the bandwidth used.
    accountState   INTEGER,  -- Indicates whether the account is enabled.
    accountInterface INTEGER -- Identifies the output interface
}

-- d) each of the entries in the above table have to be defined.

accountIndex OBJECT-TYPE
SYNTAX Unsigned32
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "The unique value that identifies this account"
::= { accountEntry 1}

accountBandwidth OBJECT-TYPE
SYNTAX Counter32
UNITS "bits per second (bps)"
MAX-ACCESS read-only
STATUS current
DESCRIPTION "The bandwidth utilized by the account so far"
::= { accountEntry 2}

accountState OBJECT-TYPE
SYNTAX INTEGER {enabled(1), disabled(2)}
MAX-ACCESS read-write
STATUS current
DESCRIPTION "Current status of the account. Note that the hub state takes Precedence over this object".
::= { accountEntry 3}

accountInterface OBJECT-TYPE
SYNTAX INTEGER
MAX-ACCESS read-write
STATUS current
DESCRIPTION "The output interface to which data is written by this account. The value would be an index into a look-up table".

-- we are now done with defining the objects for this module. At this point, one would
-- events that an agent would generate under exceptions, for instance, if encoding errors
-- are reported by the data server.

END