System and Signal Monitoring for IPTV Set-Top-Box Systems

Dávid Károly
NETvisor Ltd.,
Petzvál J. u. 56, Budapest, 1119, Hungary
Phone: (36)-(1)-371-2700, Fax: (36)-(1)-204-1664, karoly.david@netvisor.hu

Tivadar Szemethy, Árpád Bakay
Eötvös Loránd University, Joint Center for Information Sciences Research and Education
Pázmány P. sétány 1/C., Budapest, 1117, Hungary
{tiv,arpad} @ikkk.inf.elte.hu

Abstract – In IPTV systems, a set-top-box decodes digital multimedia streams into legacy video signals for the end user. Also, it provides a versatile user interface for applications like TV program guides, movie rentals, gaming, shopping etc. For providers, remote monitoring of these systems' parameters, such as performance, hardware, processes is crucial for overall service reliability. Another important issue is the (remote) monitoring of the stream-decoding process in order to measure "user experience", i.e. the quality of the multimedia stream delivered over the IP network. Our solution addresses these requirements, applicable for most set-top-box types available on the market today.

I. Introduction

Internet Television (IPTV) is a set of standards for delivering digital television channels (service) over an Internet Protocol transfer network. This generally includes encoding the video stream using advanced, high-compression digital algorithms (such as MPEG-2 and MPEG-4 AVC / H.264) and encapsulating the resulting digital stream into UDP packets. At the point of reception, the payload is extracted, audio and video streams are decoded, and displayed on the end user's TV set. As legacy TV sets are incapable of receiving this IP stream, providers deploy a *set-top-box* unit on the customers' premises, which connects to the IP network and outputs legacy video signal for the TV set. The schema of a typical IPTV network is shown in Fig. 1.

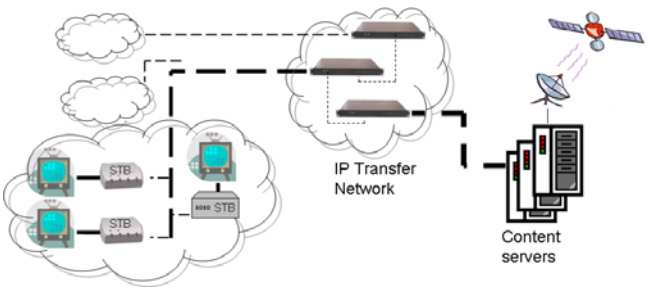

Fig. 1. Typical IPTV network setup.

TABLE I
Bandwidth requirements for IPTV technologies

Technology	Bandwidth (Mbps)	
	MPEG-2	MPEG-4 AVC
SDTV	3-4	1-2
HDTV	20-25	10-16
DVD Video On Demand	6 Mbps and up (unicast)	

Note that IPTV merely uses IP technology as a standardized and generally available means of transmission, these networks are usually not connected to the global Internet. This is mainly due to a) security and b) quality assurance reasons, the first of them being obvious. Let us now focus on the quality assurance aspect! Multimedia streaming protocols require high bandwidth, guaranteed-delay connections (Table I. shows typical bandwidth requirements for some of the protocols used today). It is far more important for the delay being constant than low, as video streaming has to deliver a constant frame-rate.

Thus, there are two main problems to overcome:

1. The transfer IP network has to provide a steady, constant stream of packets with as little *error* and *jitter* as possible
2. The set-top-box has to provide the means for the constant-rate reception and decoding of the stream, and at the same time it has to process and execute user commands (such as channel switch, pause, record etc.)

Set-top-boxes are embedded systems. They have very strict performance requirements, they have to be flexible and update-able in order to follow ever changing user demands, and finally, they have to be affordable enough to be deployed in every household.

II. SET-TOP-BOX ARCHITECTURE AND SYSTEM MONITORING

In this section, we describe a common set-top-box hardware and software architecture. During our tests, we used STBs of this kind from vendors Motorola and Tilgin. Typical systems monitoring requirements from network operators are also briefly discussed.

A. Hardware architecture

The Motorola 1900 STB is built around the highly application-oriented STb7100 "System-on-Chip" video decoder. It includes the following hardware components and interfaces:

- 266MHz ST40 CPU (SH4 compatible, supported by Linux and Windows CE)
- MPEG-2/MPEG-4 compliant video decoder DSP
- Audio decoder hardware
- High-performance 2D display engine supporting the composition of graphics and video
- Dual memory interface for SDRAM and Flash
- 100Mbps Ethernet interface
- Infrared UART for the remote controller
- Smart Card interfaces for pay-per-view applications
- Hardware watchdog
- Additional interfaces not used in the STBs studied (such as USB, SATA, modem ports)

Thus, only a minimal amount of glue hardware is necessary to build an STB.

In a typical configuration, the STB has 40-80MB of SDRAM, and 32-64MB of Flash memory. The Flash memory is used to store configuration data and to accelerate the boot process. The boot image (normally acquired via network boot) is stored in the Flash, thus it does not need to be downloaded upon subsequent reboots.

B. Software

All STBs we studied used a Linux-based embedded OS. The responsibilities of this OS include:

1. Receive and the stream via the Linux TCP/IP stack and pre-process it for efficient DMA
2. Deliver (DMA) the received packets for the DSP, which performs the audio/video decoding and produces output (TV) signals.
3. Perform user interaction by receiving remote controller input and displaying the GUI, which is usually a modified web browser, using JavaScript controls.

As computation-intensive and timing-critical decoding is performed in hardware, real-time requirements for the OS are less stringent. Thus, vendors provide best-effort solutions by only slightly modifying the standard Linux kernel (i.e. no provisions for hard real-time). This makes additional software development (such as enhanced GUI, gaming and shopping applications) easier, as standard (cross)-compilers and development environments can be used. On the other hand, developers have to be extremely cautious not to over-burden the STB, as there are no strict limits (e.g. for CPU usage or memory utilization) defined for software modules.

For this reason, when implementing our monitoring modules, we put extra emphasis on measuring our modules' own resource usage. In general, we found that while watching an MPEG-2 stream, the typical CPU utilization (without monitoring) was around 25-30% on the Motorola STB.

Another consequence of using a standard Linux kernel: process priorities do not work as expected by RT systems developers. For example, Motorola uses the standard Linux scheduler, which also accounts for fairness. Thus, it is not always true that the process with the highest (fixed) priority will run first. We feel that this might not be the best approach. For example, it is desirable that the media player process (responsible for the constant feed of the decoder HW) runs as soon as UDP packets arrive. It will run only for a minimum amount of time, (to set up DMA), but the system's configuration does not guarantee that it will run first. In Motorola's setup, it does not even have the highest priority. This clearly illustrates the best effort nature of the system.

C. Systems Monitoring

For overall service reliability, network operators monitor the STBs as well as the network connecting them to the media servers. In systems monitoring, the primary concerns are (prediction of) hardware malfunction and isolation of networking problems.

STBs lack any meaningful self-diagnosis capability thus the only indication of hardware problems are frequent reboots (STBs do have hardware watchdogs) and poor picture quality. Thus, monitoring system uptime is a useful indicator. Monitoring other, OS-level indicators such as CPU load, available memory etc, is also useful for software troubleshooting. Much more important is application-level monitoring, i.e. measuring the quality of IPTV service. In the following section, we discuss our solution for monitoring the IPTV signal.

III. MONITORING AND ANALYSIS OF THE MULTIMEDIA STREAM

Video content is usually transmitted as an MPEG-2 transport stream encapsulated in UDP packets. Although the canonical method of sending TV channels over IP would be encapsulating each elementary stream (audio, video, subtitles) in a separate RTP/UDP flow, this approach is rarely used. The other option is to multiplex the elementary streams into one MPEG-2 transport stream [15]. The MPEG-2 transport stream consists of 188-byte packets, called Transport Stream (TS) packets (Fig. 2). This transport stream is then encapsulated into UDP packets (without RTP headers) and sent over multicast to the clients. Each UDP packet contains seven transport stream packets, amounting to a total size of 1316 bytes.

Inside the transport stream, elementary streams are multiplexed using PID (Packet Identifier) fields. The PID is

used to distinguish between various audio, video and other elementary streams. Besides these elementary streams, some PIDs carry metadata, describing the content of the transport stream. Metadata content is called Program Specific Information (PSI) in MPEG terminology and it describes the TV channels contained in the transport stream. The main PSI data structure is the Program Association Table. It is found at PID 0 and it contains the list of TV channels present in the transport stream, along with their PID. Each channel has a Program Map Table, identified by the PID given in the PMT. This table contains the PID-s for audio, video and other elementary streams of this channel. IPTV streams however hardly ever contain more than one channel, so most of these tables contain redundant information.

The transport stream packet has a 4-byte header, an optional adaptation field and a payload area. The header contains the PID of the packet, a 4-bit continuity counter and various flags. One of these flags indicates if there is an adaptation field present. The adaptation field is a variable length field, containing optional fields. Most important is the Program Clock Reference (PCR) field, which is a timestamp of 90 KHz units, generated by the encoder. PCR must be sent every 100 msec. It is used for clock recovery at the receiver's side: presentation timestamps of the elementary streams are based on its value. Therefore it is easy to see that having correct PCR values is crucial for decoding the transport stream.

In our solution, we measure the following network characteristics of the packet stream:

- **Packet count, bandwidth.** This is the simplest possible metric, yet it is a useful indicator. It can show absence of transmission or incorrectly set bit rate. Also it is basic information that we can relate other metrics to, e.g. packet loss is meaningful only when compared to the actual number of packets.
- **Estimated packet loss.** Trivially, packet loss is a major factor in video quality degradation [1]. By comparing continuity counters in consecutively received packets, it is possible to infer if there has been packet loss, and estimate the number of lost packets.
- **Packet loss burstiness.** According to Mohamed and Rubino [3], the effect of packet loss depends on how bursty it is. They have concluded a surprising result: considering a fixed packet loss rate, burstier loss means less degradation in video quality. Measuring burstiness is done by comparing the number of lost packets to the number of packet loss events. The greater this ratio, the more bursty is the packet loss.
- **UDP inter-arrival jitter.** It has been shown ([2]) that jitter can have an effect on video quality similar to packet loss. Inter-arrival jitter is calculated by averaging the absolute difference between inter-arrival gaps. We also record minimum, maximum and average inter-arrival gaps over the sampling period.
- **PCR characteristics [4].** The following PCR features are examined. PCR jitter is basically the inaccuracy of successive PCR values. Given two PCR values received and the time elapsed between them, it is possible to determine what the second PCR should be. By comparing it to the actual value received, it is possible to evaluate the inaccuracy of PCR. PCR jitter is calculated by averaging those differences over the sampling period. PCR discontinuities and missing PCR-s are also counted. A PCR discontinuity is an event when the received PCR differs by more than 100 msec from the expected value. Missing PCR is an interval when PCR is not received for 100 msec.

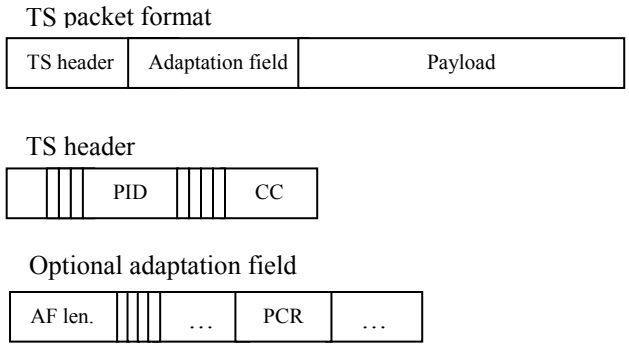

Fig. 2. IPTV packet format.

IV. IMPLEMENTATION AND TEST RESULTS

A. Assumptions about network operation

Consider the network schema shown in Fig. 1. This model suggests that stream quality degradation might come from three sources. The first source is at the headend. In the headend, external video sources (mostly DVB channels) are transcoded and converted to IPTV streams, i.e. multicast UDP stream of encapsulated transport stream packets. It is crucial that the headend produces a well-formed, good-quality output stream. The next potential source for degradation is the network. Bandwidth allocation must be done in order to deliver the stream from the headend station to the customer premises. This requires that QoS parameters i.e. flow characteristics and forwarding priorities are correctly set in network equipment such as routers and switches. The last factor is the set top box itself. It is possible that even though the video input is correct and the stream is transmitted properly, the media player process can introduce unexpected playback errors.

In the headend, we assume monitoring provided either by automatic means or simply by operators watching video quality on a video wall. Therefore, any error or quality degradation in the input (mostly DVB) channels can be spotted. Also, with this scenario headend miscofiguration, transcoding or encapsulation errors can be detected since they would become apparent on the monitoring screens. Thus, we can assume that the headend station will hardly ever emit incorrect streams and even so, the problem would be quickly detected and remedied.

The network is the next factor that can induce deficiencies to the video stream. Network characteristics that affect video playback are discussed in the previous section. In order to ensure these characteristics, network equipment must enforce appropriate QoS guarantees. This can be done e.g. by using DiffServ traffic classes or MPLS Class of Service. These flow specifications mainly rely on queuing and packet priority algorithms to accomplish class dependent processing. So we can suppose that errors in transmission - if they happen - are likely to be caused by network device misconfiguration due to incorrectly set QoS parameters. In this case, the set top box will receive a degraded video stream, affected by packet loss and jitter induced by the network.

The last factor is the set top box itself. It may happen that the stream is transmitted properly but the media player process cannot decode the stream adequately. This can be a performance problem, e.g. insufficient capacity to decode HD H.264 video or a parsing error, e.g. the media player is unable to interpret an optional field or a stream with a minor glitch. We have experienced a situation when a stream contained erroneous metadata (the PAT and PMT tables had continuity counter errors). In this case, some media player software and some of the tested set top boxes could decode it properly, some could not. In this case, tuning a "less tolerant" set top box to this channel the playback was fair for a few minutes but then the picture became jerky and decoding errors occurred. After about 5 minutes, the picture was completely frozen. It was also observed that during this, the memory usage and CPU load of the media player process increased drastically. Thus, such conditions can be detected by monitoring CPU load and comparing it to the media stream's bandwidth: excessively high processor load indicates playback errors.

These faults and sources of degradation can be analyzed and indicated by running a software module (agent) on the set top box, which accesses the transport stream's UDP packets. This agent computes the aforementioned packet loss and jitter metrics and set top box processor and memory usage indicators. We implemented two versions of this monitoring agent, as discussed in the following sections. For development, we used a concrete platform (Motorola 1920 STB), but the agent code is highly portable and could be ported to any Linux-based set top box without much difficulty. As discussed next, the main implementation problem was to gain efficient access to the UDP stream. Once the packet contents have been made available, computing the above metrics and indicators becomes trivial.

B. Implementing measurements with raw sockets

The first version of our measurement software used a raw UNIX socket. The raw AF_PACKET socket is essentially a packet snooping interface, which allows a user-space program to analyze all traffic passing through the kernel's link layer interface [9]. We decided to use this raw socket because the IP address for the multicast group of the active TV channel is not known *a priori*. With the raw socket, all UDP packets of suitable size can be captured and filtered using a LPF [10] packet filter attached to the socket. Thus the measurement process can receive the same stream that the media player is displaying.

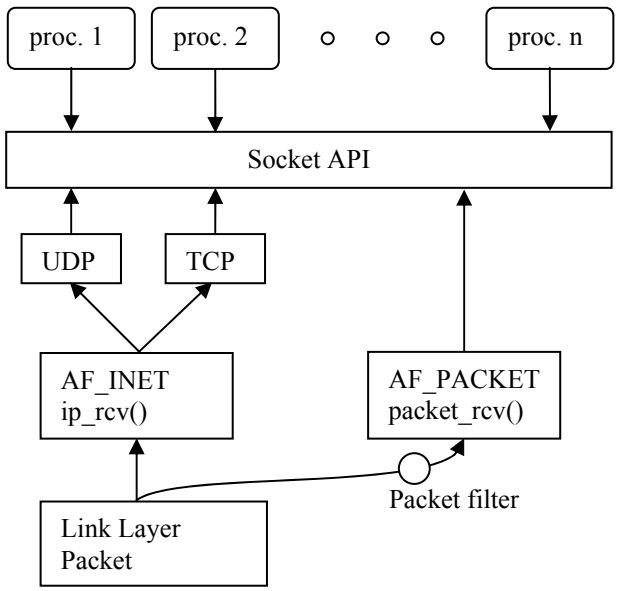

Fig. 3. Packet snooping with the AF_PACKET socket type.

Packet snooping through raw sockets is implemented in the AF_PACKET component. This special socket type allows a user mode process to completely bypass network and transport layer processing inside the kernel and directly read the raw packets delivered by the network driver (see Fig. 3). The packet snooping interface resembles a network protocol. It is accessible via the usual socket API, by specifying the PF_PACKET protocol family at socket creation. But it is not a real protocol: the created raw socket will receive all packets, bypassing upper layer processing. Because network traffic analysis programs rarely need to get all packets, a packet filter can be attached to the socket. The packet filter is specified in a kind of byte code. Upon packet reception, this byte code is evaluated and the packet is passed on to the upper layers only if the filter accepts it.

Using the above mentioned packet snooping and filtering techniques, we implemented the measurement program as follows. On startup, a packet snooping socket is created. A packet filter is attached to the socket, only allowing UDP packets of 1316 byte length, with multicast destination address in the IP header. This way the measurement process gets only the IPTV packets, all other traffic is dropped by the kernel.

Two severe drawbacks of this method have been found. The first problem is the inaccuracy of jitter measurements. In this case, there are two processes waiting for the incoming UDP packets: the media player and the measurement process. When a packet arrives both process becomes runnable. The scheduler will wake up one process while the other has to wait for the CPU. Sometimes the process to awaken first will be the media player, sometimes it will be the measurement. Using priority-based scheduling won't solve this problem either. Giving priority to the measurement process over the media player is not a feasible way because in this case the media player's buffers could starve, resulting in jerky playback. Giving priority to the media player isn't any better for the accuracy of measurements: we cannot know if the media player uses up

its time slice or it just enqueues the received packet to a decode buffer. So the measurement process will receive the packets with varying delay, i.e. this situation adds further jitter, that is not introduced by the network but the measurement itself.

The other problem is the heavy CPU load caused by kernel to user space copying. By examining function *packet_recvmsg* in the Linux kernel source file *net/packet/af_packet.c* one can see that function *skb_copy_datagram_iovec* gets called which in turn calls *copy_to_user* via function *memcpy_toiovec*. It is clear that this kind of snooping is inefficient because of the aforementioned kernel to user memory copies take significant amount of time. Our experience supports this argument. It has been found during testing in an IPTV pilot network that the CPU load imposed by the measurement process is proportional to the stream's bandwidth. CPU load of the measurement process was 5% for 2.5 Mbps, 12% for 6.5 Mbps. When watching a 10 Mbps HDTV channel, CPU load of the measurement process went up as high as 23%. Even user interface responsiveness issues were experienced.

It follows from the above mentioned reasons that the packet snooping method is not a viable approach. We needed a more accurate and efficient way to capture the UDP stream.

C. Implementing measurements in a kernel module

In order to have more accurate measurements and avoid excessive CPU load caused by kernel to user space copies, we developed a Linux kernel module that evaluates most of the indicators in kernel space and only reports aggregated values to the process running in user space. This kernel module registers itself with the kernel's networking core, so for each IP packet, a callback function gets called. The callback implemented in the kernel module examines the packet and updates the packet loss, jitter, etc. metrics if appropriate. Packet copies are eliminated as the callback function only gets a pointer to the buffer containing the packet. Timing is also improved because the timestamp field of the receive buffer is available.

Capturing received packets inside the kernel is achieved by registering our module's callback with the networking core. The low level packet reception mechanism is outlined on Fig. 4. When a packet arrives, the network driver loads the packet into an *sk_buf* socket buffer structure [5]. The network driver calls function *netif_rx* [7]. This function only does some minimal processing: it adds a timestamp to the socket buffer and inserts the socket buffer to a queue and schedules NET_RX *softirq* to be executed at a later time. Eventually the NET_RX *softirq* will be dispatched: function *net_rx_action* gets called. This function goes through the lists *ptype_all* and *ptype_base* and calls the appropriate callback functions. List *ptype_all* contains functions that are called for each incoming packet. *Ptype_base* is an array of lists containing callback functions for specific ethertypes, i.e. one list for IP, one list for IPX etc. Callback functions called this way will run in a *softirq* context. *Softirq*'s are similar to interrupts but they are scheduled in a more flexible way. When executing a *softirq*, *in_irq()* returns false but *in_softirq()* returns true.

To handle IPTV streams we have to add our callback to the list *ptype_base* for handling IP datagrams. This can be done with the *dev_add_pack* call [8]. In this situation there will be two callback functions for handling incoming IP datagrams. One callback is the original IP processing, the other one is the IPTV measurement kernel module. It is clear that the reference-counted socket buffers' mechanism eliminates the need for copies and the timestamps generated by *netif_rx* are more accurate than timing info acquired by a user mode process.

Using Linux best practices, communication between the kernel module and the process in user space is done through the */proc* filesystem. A text file is added to the directory tree in */proc*. This text file contains aggregated values for the metrics, in 'key=value' format. Since this file is read once in a few seconds, conversions and parsing overhead can be ignored.

In terms of CPU load, the kernel module is more difficult to measure. It is clear that the load imposed by the user mode process is marginal as it does not include time spent with packet processing in the kernel. But querying idle CPU time, we found that even with high bit-rate HDTV streams, there was only a few percent change. Also, user interface responsiveness is no longer an issue. Thus it can be concluded that performing the measurements in a kernel module callback function solved the performance problem.

Consequently, the kernel module has proven a viable approach to perform the required measurements. It also provides more accurate jitter values, similar to those acquired by a standard linux PC running a stand-alone measurement process.

D. SNMP interface

Results of the above analysis and diagnostics procedures are collected by the central network management application. In order to make them available, the Simple Network Management Protocol (SNMP) [11] is a straightforward choice: it is a standard, lightweight protocol, designed for monitoring network-attached devices. SNMP exposes management data as variables of the managed systems, which represent the system status and configuration. These variables can be queried (and some of them set) by managing applications. Managed systems run an *SNMP agent* process, which is responsible for this communication.

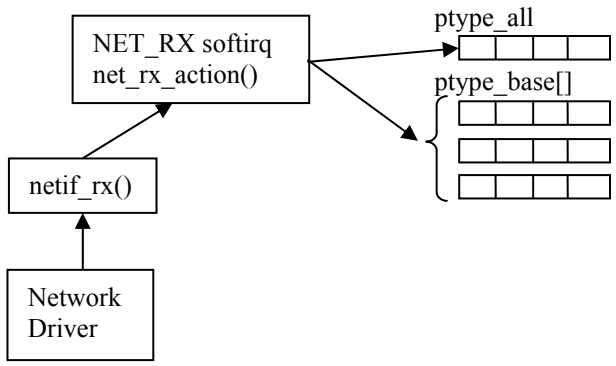

Fig.4. Packet reception in the Linux kernel.

SNMP defines a standard set of variables concerning OS operating conditions, such as *uptime*, *system load*, *memory usage*, number *of active processes* etc. As most STBs use a Linux-based OS, the straightforward solution is to port a Unix SNMP package (such as the open-source Net-SNMP [12]) to the STB's embedded OS. According to our experiences, this was fairly easy for the particular STB (Motorola) chosen. In order to reduce the memory footprint of the *snmpd* process (the SNMP agent), we performed some basic optimizations, mostly to omit support for SNMP variables and protocol versions not needed in this particular application.

The next task is to define and implement an SNMP interface for our application-specific needs. This includes defining and coding a SNMP MIB structure for the IPTV-specific variables.

As per SNMP best practices and recommendations, we decided to maintain this information as an SNMP table (*nviptvTransportStreamTable*). This enables turning on and off stream monitoring (analysis) on demand, using the following sequence:

1. The *snmpd* on the STB receives a *CREATE_ROW* SNMP command
2. *snmpd* creates and registers the variables backing a row *nviptvTransportStreamTable*, corresponding to the analysis results of the channel being watched, and enables packet capture and analysis
3. Queries for the table's variables are answered and the analysis code is invoked periodically.
4. Upon an eventual *DELETE_ROW* command *snmpd* stops packet capture and deallocates the variables

In the current implementation, the table has at most one row, representing the metrics of the channel (stream) being watched on the STB.

This scheme enables future STBs supporting multiple channels (e.g. one being watched, another being recorded).

Table II. shows the member variables of *nviptvTransportStreamTableEntry*, constituting one row of the table. These variables correspond to computed values discussed in Section III.

E. Test results

Tests were conducted in two settings. The first setting was within a corporate intranet, using pure Ethernet connectivity. This network included a CableWorld ASI-to-IP converter headend transmitting a DVB stream over IP multicast. The set top box was connected to the headend through a fast Ethernet switch. The second setting was within the IPTV test network of a major Hungarian provider. This network included Tandberg transcoding headend equipment. The multicast traffic was directed through an IP router.

TABLE II

Variables exposed over SNMP in *nviptvTransportStreamTableEntry*

Variable Name	Syntax	Description
DestAddress	STRING	Destination IP address of the multicast stream (identifies the channel)
Port	INTEGER	Destination UDP Port of the multicast stream (identifies the channel)
Control	RowStatus	SNMPv2 Row Status. Controls creation, activation and deletion of the *nviptvTransportStreamEntry* row. This is a service field for SNMP.
SamplingPeriod	INTEGER	Atomic time period for the data collection / analysis algorithm
WindowPeriod	INTEGER	Period used to compute averages and counters, an integer multiple of SamplingPeriod
InPackets	COUNTER	Number of input transport packets on the stream
InBytes	COUNTER	Number of input bytes on the stream
TSPackets	COUNTER	Number of input Transport Stream packets on the stream
IAJitter	INTEGER	Average of the inter-arrival jitter of the stream, computed similarly to that of RTCP, in usec
IAMinJitter	INTEGER	Minimum of the inter-arrival jitter of the stream, in usec, within a WindowPeriod
IAMaxJitter	INTEGER	Maximum of the inter-arrival jitter of the stream, in usec, within a WindowPeriod
PCRJitter	INTEGER	Average of the PCR jitter of the stream, in usec, within a WindowPeriod
PCRMinJitter	INTEGER	Minimum of the PCR jitter of the stream, in usec, within a WindowPeriod
PCRMaxJitter	INTEGER	Maximum of the PCR jitter of the stream, in usec, within a WindowPeriod
PacketLoss	COUNTER	Number of lost Transport packets
MultiPacketLoss	COUNTER	Maximum number of consecutive Transport packets lost
PCRDicontinuityCount	COUNTER	Number of events when the PCR counter jumps more than 100ms
PCRMissing	COUNTER	Number of PCR timeouts (PCR not received for 100ms)

In both configurations, it was confirmed that running the measurement process on the set top box does not impose severe load, nor does it degrade playback quality. It was also reported that CPU load remains small even for higher bandwidth HDTV streams (up to 10Mbps).

In the first network, we conducted an experiment of comparing the data gathered by the set top box with the measurements ran on standalone PCs. Figure 5. shows jitter values acquired this way. It can be inferred that the set top box provides jitter values that don't differ significantly from other measurements, so they can be regarded as valid.

An incorrectly configured headend was also spotted by examining the measured values. Average packet loss rate was 31 packets/sec, and CPU load was excessively high. Playback was jerky, and the picture was completely frozen after cca. 5 minutes. It turned out, that the cause of this error was a headend misconfiguration, as described in section IV.A.

Figure 6 shows data gathered by the set top box in the intranet, while watching a DVB channel. Average bandwidth was 3.8 Mbps, UDP jitter was about 250 µsec, PCR jitter was around 1.2 msec, and packet loss was negligible. Playback quality was fair.

Figure 7 shows data gathered by the set top box in the IPTV test network. The stream was generated by a Tandberg transcoding IPTV headend. Bandwidth was steadily 3,2 Mbps; UDP jitter was about 300 µsec, PCR jitter was below 2.5 msec. Sparse packet losses occurred. A few glitches (mainly blocking) were observable in video playback.

From the above mentioned results, it can be inferred that measuring stream characteristics is successfully solved by our software. It collects valid data while not interfering with video playback. In order to be able to fully understand and interpret these data, further experiments and evaluations are needed. Boundary values and reference intervals can be determined by comparing acquired data with subjective opinion scores. On the other hand, it is already clear that observing CPU load and similar operating characteristics also provides a useful indication.

V. CONCLUSIONS AND FUTURE WORK

Our promising test results show the feasibility of the approach. Depending on the available resources on the STB (mainly CPU cycles to run the analysis), our approach is well scalable. For the network operator, "device health" information (even as simple as uptime) and basic network statistics (such as Ethernet counters) are extremely helpful.

The IPTV provider is more interested in stream-related information. As we have shown, from measuring transport stream characteristics, such as packet loss, information relevant to the customer's experience can be deduced. Further analyzing the stream is more costly in terms of STB CPU cycles, but is certainly feasible. Currently, we are working on developing analysis methods for higher-level behavior (such as channel switch delay). This particular metric is a good characterization for the IPTV service in general, and provides information about other critical aspects of the system, like appropriate IGMP (multicast routing management) performance.

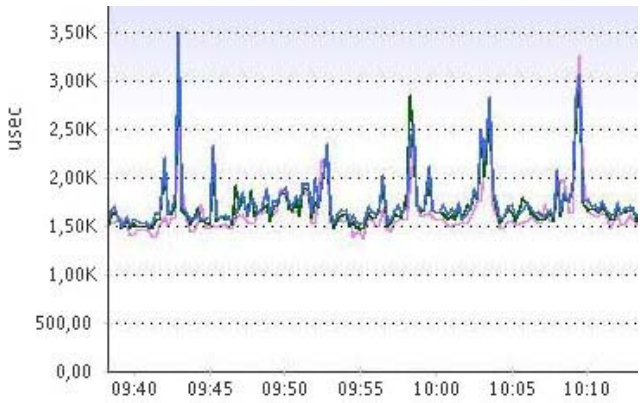

Fig.5. Comparing measured jitter values for a stream
blue – PC, Linux 2.4, green – set top box, red – PC, Linux 2.6.

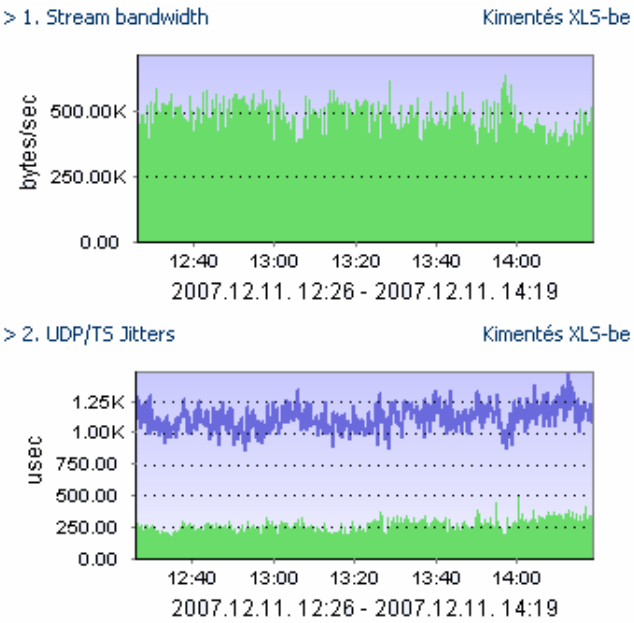

Fig.6. Measurement results in a corporate intranet.
A. Packets/sec, blue – TS packets, green – UDP packets
B. Jitters, blue – inter-arrival jitter, green – PCR jitter

Industrial vendors providing test and measurement equipment for IPTV networks generally take two different approaches:

1. Provide small, handheld "line testers" field technicians can use to check the (physical) availability of the service. For example, Argus Telecommunications [13] offers small, rugged testers of a mobile phone's form factor. These devices are highly portable, easy to operate, but lack higher-level analysis capabilities and also lack a high-definition screen which would be quite useful in this application. Additionally, since these devices are manufactured in limited numbers, they are relatively costly.
2. Another approach is to provide analysis software to be run on dedicated, high-performance workstations, connected to the IPTV network. These are able to monitor hundreds of channels simultaneously and analyze the streams in depth, sometimes using sophisticated AI methods (such as neural nets) to deduce user experience. The Agama Analyzer, provided by Agama Technologies [14] is a representative example of this class.

The main drawback of this approach is that due to the high cost, operators cannot afford running analyzers at arbitrary endpoints.

Our approach fits nicely between these two: it provides analysis capability at the endpoint, without the need to purchase additional equipment. The resources of a set-top-box are certainly insufficient to perform really deep analysis. In our experience though, the vast majority of IPTV problems are caused by transfer network deficiencies, and these are detectable using relatively simple analysis.

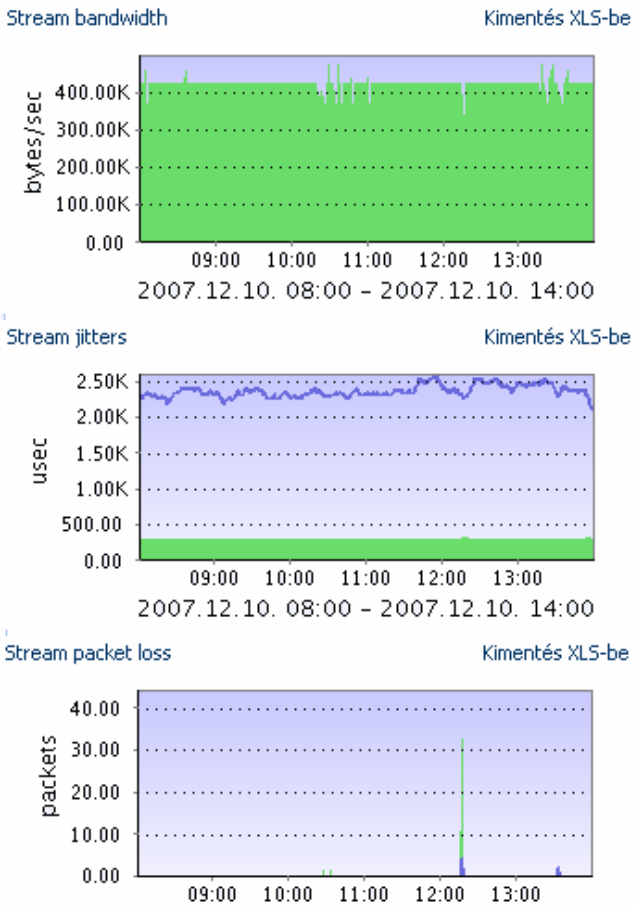

Fig.7. Measurement results in a provider's IPTV test network.
A. Stream bandwidth, bytes/sec
B. Jitters, blue – inter-arrival jitter, green – PCR jitter
C. Packet loss

ACKNOWLEDGMENT

The research presented in this paper was supported by the *Joint Center for Information Sciences Research and Education* (IKKK, at Eötvös Loránd University).

Additionally, we'd like to thank Invitel Ltd, of Hungary, and Smart-Com Ltd. of Slovenia for their help with providing test equipment and consultation. We would also like to thank Mr. Róbert De Vescovi of CableWorld Ltd. for providing support with IPTV headend configuration.

REFERENCES

[1.] Jill M. Boyce, Robert D. Gaglianello, "Packet loss effects on MPEG video sent over the public Internet", *Proceedings of the sixth ACM international conference on Multimedia,* pp. 181-190, 1998.

[2.] Mark Claypool and Jonathan Tanner, "The Effects of Jitter on the Perceptual Quality of Video", *Proceedings of ACM Multimedia Conference,* 1999.

[3.] S. Mohamed, G. Rubino, "A study of real-time packet video quality using random neural networks", *IEEE Trans. Circuits Syst. Video Technol.* 12 (12) (2002) pp. 1071-1083.

[4.] ETSI EN TR 101 290. *Digital Video Broadcasting (DVB); Measurement guidelines for DVB systems.*

[5.] Alan Cox, "Network Buffers and Memory Management", *Linux Journal,* issue 29, September 1996.

[6.] Jamal Hadi Salim, Robert Olsson, Alexey Kuznetsov, "Beyond Softnet", *Proceedings of the 5th Annual Linux Showcase & Conference,* 2001.

[7.] Alessandro Rubini and Jonathan Corbet, *Linux Device Drivers,* 2nd Edition, O'Reilly, 2001.

[8.] Gianluca Insolvibile, "Inside the Linux Packet Filter", *Linux Journal,* Volume 2002, Issue 94 (February 2002).

[9.] Gianluca Insolvibile, "Inside the Linux packet filter, part II", *Linux Journal,* Volume 2002, Issue 95 (March 2002).

[10.] Patrick Th. Eugster, "Linux socket filter: sniffing bytes over the network", *Linux Journal,* Volume 2001, Issue 86 (June 2001)

[11.] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2,* 3rd Edition, Addison Wesley, 1998. ISBN 978-0201485349.

[12.] Net-SNMP. http://net-snmp.sourceforge.net/

[13.] Argus Telecommunications Product site at http://www.argus.info/

[14.] Agama Technologies website at http://www.agama.tv/

[15.] ISO International Standard. *Information technology - Generic coding of moving pictures and associated audio information - Part 1: Systems Specification.* ISO/IEC 13818-2, first edition 1996-05-15 edition.