Ultra Lightweight Encapsulation for Efficient IPv6 Transport

Bernhard Collini-Nocker, Hilmar Linder, Wolfram Stering
Department of Scientific Computing
Paris-Lodron University of Salzburg, Austria
Email: [bnocker, hlinder, wolfi]@cosy.sbg.ac.at

***Abstract* — The MPEG-2/DVB standards define several methods to encapsulate and carry IP datagrams in transport stream packets. In order to specify a flexible, yet efficient encapsulation the IETF has started the IP over DVB working group activity that developed the Ultra Light Encapsulation (ULE). ULE is currently in the working group last call procedure and may soon become an Internet standard. One specific goal of ULE is the efficient encapsulation of IPv6 datagrams. This paper describes the ULE encapsulation, its open source implementation, and presents the results of measurements that were carried out to compare the DVB standardized Multi-Protocol Encapsulation with ULE.**

I. INTRODUCTION

Digital Television Systems have received worldwide deployment (e.g. using the Digital Video Broadcasting (DVB) standards [1] from the European Telecommunications Standards Institute (ETSI) [2-4]). DVB builds on the work of the Motion Picture Experts Group (MPEG) standardized by the International Standards Organization (ISO) [5] to specify a complete transmission system. Satellites (DVB-S) in particular, may be used to extend the Internet service to areas not readily covered by other broadband technology.

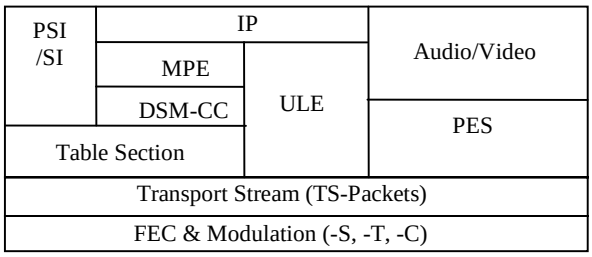

Figure 1: DVB/MPEG-2/IP Protocol Stack

The MPEG-2 transmission network [5] allows transmission of fixed-sized frames, known as "TS-Packets". TS-Packets may carry Packetized Elementary Streams (PES) of audio and video and, in the form of Tables (e.g. Program Specific Information/Service Information (PSI/SI) [5]), associated control information. Tables are sent by segmenting them into Sections and then fragmenting each section into a series of fixed-sized TS-packets. Sections may also carry Digital Storage Media Command and Control [6], DSM-CC data, such as carousels. A PC-based DVB Receiver card or Integrated Receiver Decoder (IRD) employs hardware, firmware, or software to receive TS-packets and to reassemble the PES and transmitted Sections.

A. IP over MPEG-2

An MPEG-2 transmission network may be used to transmit Internet Protocol (IP) packets [7] sourced locally or remotely over the MPEG-2 link for delivery to a local host (e.g., a PCI-based DVB Receiver card), or forwarded over other IP bearers to remote hosts (e.g., a DVB receiver card or set-top box with routing software). Such IP services are widely deployed today.

Protocol Data Units (PDUs) (e.g. IP packets, Ethernet frames or MPLS payloads) are first sent to an Encapsulator at the edge of the MPEG-2 network. This encapsulator uses a convergence protocol [8, 9] to form each PDU into a Subnetwork Data Unit [10] (SNDU) by adding header fields that carry protocol control information. The variable-sized SNDU is fragmented into a sequence of fixed sized TS-Packets. When the TS-Packets have been delivered over the MPEG-2 network, a Receiver reassembles the SNDUs and extracts the original PDUs. In the case of IP packets, these are then forwarded to the connected destination host or IP network.

The DVB standards specify a method known as Multi Protocol Encapsulation (MPE) [4] to support transmission of SNDUs within the control plane of MPEG-2. Conceptually MPE is layered (see Figure 1) as an extension above DSM-CC, using a table type 0x3E (a DSM-CC Section containing private data) with DSM-CC stream type 0x0A (MPE). MPE permits encapsulation of PDUs up to almost 4 KB. The basic MPE header format carries a MAC destination address, but no payload type field, this may lead to the assumption in most current Receiver driver software that the payload is IPv4 only.

B. Case for a New Encapsulation Algorithm

The MPE encapsulation is significantly more complex than convergence protocols such as ATM/AAL5 [11], and is generally considered to be neither lean nor particularly efficient in terms of processing requirements [9]. Other criticisms include the mandatory use of a destination MAC address, a lack of an optional source MAC address, the ordering of destination MAC address bytes, and the absence of a next header protocol type field in the base header.

When a type field is required (e.g. for IPv6), it is added by including an IEEE LLC/SNAP adaptation field. Presence of the LLC/SNAP field is indicated by a flag-bit in the MPE SNDU header. The efficiency of transmission and implementation are therefore impacted in scenarios employing bridging, multicasting and IPv6 transmission. Finally, the MPE Specification [4] provides several options for receiver and transmitter functionality and does not explicitly specify how a sender or receiver should behave, or how interoperable configurations should be selected.

Recently, an alternative to MPE has been proposed that is tailored to IP networks. The Ultra Light Encapsulation (ULE) [8] allows simple and efficient encapsulation of PDUs up to 32 KB and uses a mandatory CRC-32. A summary of the major differences of ULE

compared to MPE is:

- Support for an Ethertype like type field providing code points for IPv4, IPv6, MPLS, bridged frames, etc.
- Support for optional destination MAC address.
- Simple, unambiguous design for leaner and easier implementation.
- No "hidden" features, thus better interoperability.
- Improved efficiency (less overhead and less instructions).
- Flexibility through extension headers.

The ULE protocol is being developed by the IETF "IP over DVB" (ipdvb) Working Group, and is one of a number of work items intended to enhance performance and interoperability while improving support for automated configuration for IP networks using MPEG-2 transmission [7]. A key goal of the activity leading to the definition of ULE was the need to precisely define the behavior of the protocol – such that any compliant ULE Receiver is able to receive data from any compliant ULE sender. Other intended work items include the definition of protocols to resolve IP addresses to TS-Logical Channels and support for automated discovery of network neighbor addresses within an IP network. These topics are not discussed in this paper.

The remainder of this paper is organized as follows: Section 2 presents the criteria for an efficient encapsulation and will argue for the ULE header compared to the MPE header. Section 3 describes the implementation of the open source DVB receive drivers. Section 4 compares the transmission efficiency of ULE with MPE. Finally, the findings are summarized and future research work in the IP over DVB area is discussed.

II. ULE ENCAPSULATION

A. Encapsulation Method

An SNDU is transmitted over an MPEG-2 transmission network by placing it either in the payload of a single TS Packet, or if required, fragmenting it into a series of TS Packets. Where there is sufficient space, ULE permits a single TS Packet to carry more than one SNDU (or part thereof), sometimes known as Packing. All TS Packets comprising a SNDU are assigned the same PID, to facilitate reassembly at the Receivers.

The ULE specification [8] follows that for Private Data in MPEG-2 [5]. That is, ULE transmits SNDUs directly over the Transport Stream layer (Figure 1). This is sometimes called "Data Piping". The header of each TS Packet carries a one bit Payload Unit Start Indicator (PUSI) field. The PUSI identifies the start of a Payload Unit (SNDU) within the MPEG-2 TS Packet payload. In ULE, the following PUSI values are defined (in compliance with [5]:

0: The TS Packet does not contain the start of a SNDU, but contains the continuation, or end of a SNDU.

1: The TS Packet contains the start of a SNDU, and a one byte Payload Pointer follows the last byte of the TS Packet header.

If a SNDU finishes before the end of a TS Packet payload, but it is not intended to start another Payload Unit, a stuffing procedure fills the remainder of the TS Packet payload with bytes with a value 0xFF [5], known as Padding.

B. SNDU Format

PDUs (i.e. IPv6 packets) are encapsulated using ULE to form a SNDU. The encapsulation format to be used for PDUs is illustrated below:

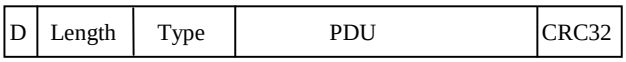

Figure 2: SNDU Encapsulation

The interpretation of the SNDU header fields is as follows:

- Destination Address Present (D) Field: this 1-bit value of 0 indicates the presence of a destination MAC address following the Type field. A value of 1 indicates that a Destination Address Field is not present (i.e. it is omitted).
- Length Field: A 15-bit value that indicates the length, in bytes, of the SNDU counted from the byte following the Type field, up to and including the CRC-32.
- Type Field: The 16-bit Type field indicates the type of payload carried in a SNDU, or the presence of a Next-Header. The set of values that may be assigned to this field is divided into two parts, similar to the allocations for Ethernet. A Type filed smaller than 1536 (decimal) indicates the presence of an extension header (see section II.E). The second set of ULE Type field values comprise the set of values greater than or equal to 1536 in decimal. In ULE, this value is identical to the corresponding type codes specified by the IEEE/DIX type assignments for Ethernet and recorded in the IANA EtherType registry.
- SNDU Destination Address Field (not shown in figure 2): The optional SNDU Destination Address Field is carried only when the D field is set to 0 and follows the type field. This may be used for IP unicast packets destined to routers that are sent using shared links (i.e., where the same link connects multiple Receivers). A sender may use a D field set to 1 for an IP unicast packet and/or multicast packets delivered to Receivers that are able to utilize a discriminator field (e.g. the IPv4/IPv6 destination address), which in combination with the PID value, could be interpreted as a link-level address.
- SNDU Trailer CRC: each SNDU carries a 32-bit CRC field in the last four bytes of the SNDU. The CRC-32 polynomial from the DSM-CC section [5] syntax is to be used. This complies with the IPv6 requirement that link-level frames should be error free.

When the first two bytes of a SNDU have the value 0xFFFF, this denotes an End Indicator. This indicates to the Receiver that there are no further SNDUs present within the current TS Packet.

C. SNDU Encapsulation Comparison

The SNDU header complexity usually dominates the volume of code needed to implement a Receiver and Encapsulator. Optional components can significantly impact performance (specifically execution cost increases with the number of bit-manipulations required to convert protocol variable to header field values).

The MPE SNDU header provides flexible delivery for a wide range of deployment scenarios. The penalty for this flexibility is a complex header structure with a range of encapsulation options leading to 4 basic header formats, ranging from 16 to 44 Bytes. This complexity can be disadvantageous when performing software-based encapsulation/decapsulation (e.g. in the driver software of a Receiver). In contrast, the SNDU header in ULE is simpler and comprises only 3 fixed fields. The small header reduces the number of instructions and code paths to be considered during reassembly of a particular Sub Network Data Unit (SNDU), and hence improves software processing efficiency. The mandatory presence of a destination link (MAC) address in MPE has been avoided in ULE through the introduction of the D field to improve transmission efficiency of IP-multicast and unicast in some specific scenarios [8]

(e.g. directly connected end-hosts). When receiver addressing is required, ULE adds an additional 6 byte SNDU destination address field, analogous to the MPE MAC address.

Table 1 compares the overhead of MPE and ULE (but ignores the impact of the MPEG-2 Payload Pointer, associated with fragmentation [5]). It shows that ULE reduces the header overhead by 4% - 10%. For IPv6 packets, the overall size of the base header in ULE and MPE does not differ significantly– although in ULE, when the destination address is omitted (D=1), or bridging is used, a considerable saving may be achieved. The LLC option of MPE increases overhead by 8 Byte.

Encapsulation	SNDU Header				Total SNDU Overhead including CRC
	Payload Type	Link Address	LLC	Bridging	
MPE	IPv4	6B	-	-	16
MPE, LLC	IPv6	6B	8B	-	24
MPE Bridging	Any	6B	8B	14B	44
ULE (D=0)	Any	6B	-	-	14
ULE (D=1)	Any	-	-	-	8
ULE(D=0) Bridging	Any	6B	-	14B	28
ULE(D=1) Bridging	Any	-	-	14B	22

Table 1: Summary of SNDU Overhead.

D. SNDU Fragmentation and Reassembly

Since PDUs are of variable size (up to 64 KB for IPv4, and potentially larger for IPv6), most SNDUs will require fragmentation. There are similarities between the MPE and ULE fragmentation process. A one-bit Payload Unit Start Indicator (PUSI) in the MPEG-2 TS-Packet header [5, 8] indicates a specific TS Packet carries the start of a new payload (i.e. SNDU). MPEG-2 [5] provides three options when SNDUs do not precisely align to the end of a TS-Packet payload: Padding, Packing, and Stuffing. Stuffing requires the additional use of Adaptation Field Control bits and an Adaptation Field. It is not used in ULE and is not further discussed here.

With Padding, each new SNDU starts in a new TS-Packet (as does each PES for Audio/Video), and therefore padding bytes fill any remaining payload within the final TS-Packet used to send a SNDU. To allow a Receiver to identify the start of each SNDU, the Encapsulator sets the PUSI in the first TS-Packet used, and directly follows this with a Payload Pointer byte, indicating the position of the first byte of the SNDU within the payload (usually 0).

Using Packing [4], an Encapsulator can utilize the remaining bytes following a previous SNDU to start sending a new SNDU. This removes the need for Padding, provided there is a continuous stream of PDUs arriving at the input of the Encapsulator. As in Padding, the PUSI Pointer indicates the start of the first SNDU within a specific TS-Packet, although subsequent SNDUs in the same TS-packet are found by offsetting the length of previous SNDUs. To control Packing, ULE introduces the concept of a Packing Threshold and/ payload packing time-out, which is a period of time that the Encapsulator is willing to defer transmission of a partially filled TS-Packet to accumulate more SNDUs, rather than use Padding. After the Packing timeout, the Encapsulator uses Padding to send the partially filled TS-Packet. A Packing Threshold of zero is equivalent to Padding.

E. Extension Headers

ULE extension headers were introduced to gain more flexibility and to allow future extensions to the ULE process without requiring changes to the already established ULE implementations.

In ULE, a Type field value less than 1536 (decimal) indicates an Extension Header. This field is organized as a 5-bit zero prefix, a 3-bit H-LEN field and an 8-bit H-Type field, as follows (see Figure 3):

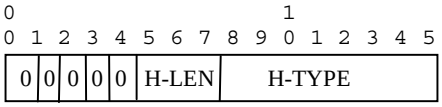

Figure 3: Structure of ULE Next-Header Field.

A H-LEN of zero indicates a Mandatory Extension Header. Each mandatory extension header has a pre-defined length that is not communicated in the H-LEN field [8]. The H-Type is a one byte field that is either one of 256 mandatory header extensions or one of 256 Optional Header Extensions. If these optional extension headers are not known to a Receiver, it may omit them and may still deliver the PDU.

The method used for extension headers follows closely the use of next-level headers in IPv6. Figure 4 shows an SNDU including two Extension Headers. The values of T1 and T2 are both less than 1536 Decimal, each indicates the presence of an Extension Header, rather than a directly following PDU. T3 has a value greater than 1535 indicating the EtherType of the PDU being carried.

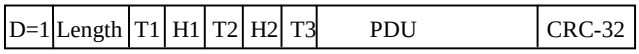

Figure 4: Encapsulation with two Extension Headers.

III. Implementation of the Driver

A. Hardware and Driver Architecture

The ULE decoder has been implemented as an extension to the well-known and widely used, open-source Linux DVB driver [14] which works for the PCI (and some USB) adapters based on a design originally done by Technotrend. Supported cards include the Hauppauge WinTV DVB-S Nova and Nexus, to name just two, but a number of others exist as well. Additionally, different revisions of that design are available, using different hardware for tuners or other components. Most – if not all of them – are supported well by the Linux driver.

B. Linux DVB Devices: The Driver API

The Linux DVB API offers control of the hardware components through currently six Unix-style character devices for video, audio, frontend, demux, CA and IP-over-DVB networking. The video and audio devices control the MPEG2 decoder hardware, the *frontend* device the tuner and DVB demodulator. The *demux* device allows control over the PES and section filters of the hardware. If the hardware does not support filtering, these filters can be implemented in software. Finally, the CA device controls all the conditional access capabilities of the hardware.

The driver already supports Linux network interfaces for MPE encoded streams (i.e. MPEG2/DVB sections) via its dvb_net module, which feeds the IP packets extracted from MPE sections received by a feeder instantiated via the *demux* API into the Linux network protocol stack.

C. The ULE Decoder

The new ULE decoder is tightly integrated with the dvb_net module. Upon creation of an ULE network interface, it uses the *demux* API to setup a TS feeder – one that delivers (blocks of) single TS packets – for the desired stream, referred to via its PID value. Then, as these TS packets are received, the decoder (an instance thereof is working for each network interface) performs the following steps for each TS packet:

1. Check the TS packet's validity: the TS sync byte (0x47) and the TS header's transport error indicator (TEI) bits are examined, bad packets are dropped.
2. If the decoder is in synchronization state: drop the TS packet, unless the payload unit start indicator (PUSI) is set, which marks the beginning of a new ULE SNDU; change to synchronized state.
3. If the decoder is synchronized and the PUSI flag is set: check, by examining the pointer field, if the current ULE SNDU can be completed with the data available in the current TS packet.
    a. If not, drop the current ULE SNDU, synchronize on the new one; apparently we missed one or more TS packets.
    b. Else, it is a packed payload; complete the current SNDU, and re-process the same TS cell to synchronize on the beginning of the new payload.
4. Else (not PUSI or no current ULE SNDU available): add the current TS packet's payload to the current ULE SNDU, creating a new one, if required.
5. If an ULE SNDU has been completed with the current TS packet's payload (known from the SNDU length field):
    a. verify the CRC32, drop the ULE SNDU, if this fails.
    b. check for the presence of ULE extension headers (by examining the ULE type field) and call the respective handler(s), which might drop the SNDU.
    c. pass the SNDU as Ethernet frame to the Linux kernel.

These steps give a rough overview of the decoder's algorithm, although, in reality some more error checking and handling of 'strange' situations have to be performed to achieve a robust decoder, especially with respect to packed payloads and re-synchronization in case of errors. In total, about 400 lines of code cope with the reassembly and decoding process and another 80 are required for extension header handling. Ideally, this whole functionality would be implemented in the firmware for the card, as is already the case with the section decoder. Then, only step 5 remains to be done in the driver's context. The ULE implementation is either available from the LinuxTV CVS [14] or from [15].

IV. MEASUREMENTS

A. Testbed

For the measurements a Linux based IP encapsulator from gcs [12] has been used. It provides virtual network interfaces corresponding to logical circuits, addressed by a specific packet identifier (PID) for the underlying transport multiplex. Each virtual network interface (dvbX) is characterized by its PID, its encapsulation and some interface parameters, out of which the payload packing time-out value has been used to set the time until when the driver waits for new packets being concatenated within the same TS packet before filling the TS packet with padding bytes.

The encapsulator has been configured in such a way that only dedicated traffic was routed to the respective dvb interface. The driver of the encapsulator can be queried and provides statistics about the amount of data bytes sent to the hardware interface and other information. Before each series of tests the driver and interface statistics have been reset.

B. Measurements

The measurements conducted concentrated on the effect of the payload packing timeout. For the IPv6 measurement iperf [13], version 2.0.1, was used to send an arbitrary number of IPv6 packets to an IP-multicast group. This traffic has been recorded with tcpdump [16] and played back on each of the dvb interfaces using tcpreplay [17]. IP-multicast addressing was used primarily to avoid sending feedback being necessary for TCP and UDP tests normally performed with iperf. Four interfaces have been configured, dvb0 with MPE (implicitly using LLC/SNAP for IPv6 transport), dvb1 with MPE and section packing (again implicitly using LLC/SNAP for IPv6 transport), dvb2 for ULE with Destination Address (DA) present, and dvb3 for ULE without Destination Address. The ULE interfaces were configured to always provide packing. Values of 10 ms and 100ms for the payload packing time-out were used for dvb1, dvb2, and dvb3.

The amount of encapsulated data is shown in the following tables. The column "Bytes sent" counts the number of bytes and the column "TS packets" shows the number of 188 TS packets that have been passed from the encapsulator module to the hardware. Finally the "%" column calculates the efficiency with respect to MPE.

The following tables show the transmission efficiency for IPv6 transport at 100Kbps, 1 Mbps, 2 Mbps and 10 Mbps respectively. The packing timeout was set to 100 ms; the transfer duration was 60 seconds.

Rate	DA	Interface	Bytes sent	TS packets	%
100Kbps	Yes	MPE	880404	4683	100
100Kbps	Yes	MPE-SP	880404	4683	100
100Kbps	Yes	ULEP	880404	4683	100
100Kbps	No	ULEP	880404	4683	100

Rate	DA	Interface	Bytes sent	TS packets	%
1Mbps	Yes	MPE	8756664	46578	100
1Mbps	Yes	MPE-SP	8054108	42841	91.977
1Mbps	Yes	ULEP	8000716	42557	91.367
1Mbps	No	ULEP	7969132	42389	91.006

Rate	DA	Interface	Bytes sent	TS packets	%
2Mbps	Yes	MPE	17507876	93127	100
2Mbps	Yes	MPE-SP	16103140	85655	91.977
2Mbps	Yes	ULE	15996356	85087	91.367
2Mbps	No	ULE	15933188	84751	91.006

Rate	DA	Interface	Bytes sent	TS packets	%
10Mbps	Yes	MPE	87522648	465546	100
10Mbps	Yes	MPE-SP	80499344	428188	91.975
10Mbps	Yes	ULE	79965612	425349	91.366
10Mbps	No	ULE	79650336	423672	91.001

Above tables clearly show that the load of a link strongly influences the payload packing efficiency. Due to the smaller header also ULE is also better than MPE. Although a saving of 1 percent does not sound a huge improvement this 1 percent easily corresponds

to an average of 400 kilobit/sec saving on a full transponder. In other words the 400 kbps correspond to roughly 1.2 GB of download volume per hour that can be saved.

In repeating the same series of tests with a payload packing time-out of 10ms, the following tables result. Of course the MPE cannot perform differently.

Rate	DA	Interface	Bytes sent	TS packets	%
1Mbps	Yes	MPE	8756664	46578	100
1Mbps	Yes	MPE-SP	8289860	44095	94.67
1Mbps	Yes	ULE	8289860	44095	94.67
1Mbps	No	ULE	8289860	44095	94.67

Rate	DA	Interface	Bytes sent	TS packets	%
2Mbps	Yes	MPE	17507876	93127	100
2Mbps	Yes	MPE-SP	16379876	87127	93.557
2Mbps	Yes	ULE	16127016	85782	92.113
2Mbps	No	ULE	16127016	85782	92.113

Rate	DA	Interface	Bytes sent	TS packets	%
10Mbps	Yes	MPE	87522648	465546	100
10Mbps	Yes	MPE-SP	80618348	428821	92.112
10Mbps	Yes	ULE	80190648	426546	91.623
10Mbps	No	ULE	80054160	425820	91.467

The longer the encapsulator waits for consecutive IPv6 packets the more efficient the overall encapsulation process gets. For real traffic the efficiency will heavily depend on the queue size of the respective interface, corresponding to the number of pending packets to be encapsulated. An adaptive Packing time-out depending on traffic load could help optimizing efficiency.

A reasonable value of 100 ms for the Packing time-out can lead to more than 7 percent of bandwidth saving without noticeable increasing the average delay for a satellite transmission for lightly loaded links. For cable and terrestrial transmissions that have a processing and transmission delay of a few milliseconds, again, the adaptation of the Packing time-out with respect to the traffic offered and queue sizes will be mandatory to optimize encapsulation efficiency.

In summary the above tables provide ample evidence that ULE efficiency of IP6 transport is significantly better than MPE without payload packing, and noticeably better than MPE with payload packing in performance for IPv6 transport, yet the results are heavily dependent on packing payload time-out.

V. Conclusion

This paper has compared the Multi-Protocol Encapsulation with the Ultra Lightweight Encapsulation, a new encapsulation protocol being defined by the ipdvb WG, which places packets directly into the MPEG-2 transport stream. Compared to Multi-Protocol Encapsulation it provides several functional improvements. These represent the primary reasons for developing ULE:

- For IP-multicast transport ULE saves 8 bytes of overhead and obsoletes the mandatory use of a destination MAC address.
- IPv6 support is natively provided by a code point, instead of going through the general purpose LLC/SNAP method used by MPE. Apart from the increased header size of IPv6 (40 bytes instead of 20 in the minimal case) ULE can save 16 bytes of encapsulation header (8 bytes less MPE plus 8 bytes LLC/SNAP) thus making IPv6 over ULE nearly as efficient than IPv4 over MPE.
- IPv4 unicast addressing has been difficult to configure in uni-directional transmission networks, requiring the knowledge about the MAC address of the receiving card in order to set up static ARP entries at the sending gateway. ULE allows sending an IP packet addressed to the destination IP address, instead, which removes the need for an operator to maintain a database of the hardware at the receiver side.
- The implementation of the ULE decapsulator code is straight forward and amounts for approx. 400 lines of code.

This paper shows that the currently implemented open source driver is able to map the expected theoretical results well to the practical results. It is further assumed that the availability of the ULE driver in the standard Linux Kernel (greater than 2.6.7) might further speed-up the movement from MPE to ULE.

Further work is certainly needed to optimize encapsulation efficiency for different traffic patterns corresponding to changing queue sizes, where the Packing time-out needs to be adopted accordingly.

The authors are continuing their efforts in the ipdvb working group to fine tune the encapsulation, its implementation, and provide input to assist for the associated networking functions, such as address resolution and routing. The goal is to provide a reasonable and useful series of standards.

Acknowledgment

This standardization activity was partially funded by the European Commission's Sixth Framework Programme, the Information Society Technologies (IST) Project BROADWAN and by the ESA project IPEncaps.

References

[1] DVB, "Digital Video Broadcasting Home Page." http://www.dvb.org.
[2] ETSI, "Digital Video Broadcasting (DVB); Interaction Channel for Satellite System Distribution," EN 301790 v.1.1.1 2000-07, 2000.
[3] ETSI, "Framing structure, channel coding and modulation for 11/12 GHz satellite services," ETSI, Draft DVB-S: ETS 300 421, 2000.
[4] ETSI, "Digital Video Broadcasting (DVB); DVB specification for data broadcasting," European Standard (Telecommunications series), ETSI EN 301 192 V1.3.1 2003.
[5] ISO/IEC, "ISO/IEC 13818: Part 1: Information Technology - Generic Coding of Moving Pictures and Associated Audio Information", 2000.
[6] ISO/IEC, "ISO/IEC 13818: Part 6: Extensions for DSM-CC is a full software implementation," International Organization for Standardization and International Electrotechnical Commission 1995.
[7] M.J. Montpetit, G. Fairhurst, H. D. Clausen, B. Collini-Nocker, and H. Linder, " A Framework for transmission of IP datagrams over MPEG-2 Networks," IETF Work-in-Progress, Internet Draft, draft-ietf-ipdvb-arch-03.txt, 2004.
[8] G. Fairhurst and B. Collini-Nocker, "Ultra Lightweight Encapsulation (ULE) for transmission of IP datagrams over MPEG-2/DVB networks," IETF Work-in-Progress, Internet Draft, draft-ietf-ipdvb-ule-05.txt, 2005.
[9] H. Clausen, H. Linder, and B. Collini-Nocker, "Internet over Direct Broadcast Satellites," IEEE Communications Magazine, vol. 37, pp. 146-151, 1999.
[10] P. Karn (Editor), "Advice for Internet Subnetwork Designers," IETF Work-in-Progress, Internet Draft (BCP), draft-ietf-pilc-link-design-15.txt, 2004.
[11] J. Heinanen, "Multiprotocol Encapsulation over ATM Adaptation Layer 5," Network Working Group, RFC1483, 1993.
[12] Open DVB Gateway: http://www.gcs-salzburg.at/.
[13] Iperf: http://dast.nlanr.net/Projects/Iperf/
[14] LinuxTV: http://www.linuxtv.org
[15] Network Research: http://www.network-research.org
[16] tcpdump: www.tcpdump.org
[17] tcpreplay: tcpreplay.sourceforge.net